УДК 519.853.4, 519.833.3

ПАРАЛЛЕЛЬНЫЙ ГЛОБАЛЬНЫЙ ПОИСК РАВНОВЕСНЫХ СИТУАЦИЙ В БИМАТРИЧНЫХ ИГРАХ

И. Л. Васильев[1], **К. Б. Климентова**[1], **А. В. Орлов**[1]

Рассматривается задача поиска ситуаций равновесия по Нэшу в биматричных играх. Для решения этой задачи используется вариационный подход, базирующийся на теореме эквивалентности биматричной игры и билинейной задачи математического программирования специального вида. Последняя решается с помощью алгоритма, основанного на теории глобального поиска. Для решения вспомогательных задач линейного программирования, возникающих на шагах алгоритма, используется пакет ILOG CPLEX. Кроме того, осуществлена параллелизация алгоритма глобального поиска. Проведен вычислительный эксперимент по тестированию последовательного и параллельного алгоритмов. Полученные результаты подтверждают эффективность реализованной параллелизации. Работа выполнена при поддержке РФФИ (код проекта 05–01–00110), а также гранта Президента РФ (код проекта МК-6580.2006.1).

Ключевые слова: биматричные игры, равновесие по Нэшу, локальный поиск, глобальный поиск, параллельные вычисления, математическое программирование.

1. Введение. В последние годы все большее распространение получают технологии параллельных вычислений и связанное с ними программное обеспечение. Выявление и использование параллельных структур алгоритмов при разработке пакетов программ помогает существенно сократить время решения задач и, следовательно, увеличить их размерность (решение, скажем, невыпуклых задач оптимизации большой размерности является актуальной задачей исследования операций, возникающей на практике [1, 2, 4]).

С другой стороны, в области оптимизации можно отметить увеличение количества бесплатных и коммерческих пакетов программ, ориентированных на решение "базовых" задач оптимизации, например таких как задачи линейного программирования, выпуклые квадратичные задачи и др. Методы решения таких задач часто являются составными частями, "кирпичиками" алгоритмов, разрабатываемых для решения более сложных задач, поэтому скорость решения "базовых" задач, как правило, существенно сказывается на времени решения всей задачи в целом.

В настоящей статье рассматривается задача поиска ситуаций равновесия по Нэшу в биматричных играх [3 – 8]. Как известно [6], данная задача эквивалентна поиску глобального решения в билинейной задаче специального вида, которая, в свою очередь, является невыпуклой. Для ее решения используется алгоритм, разработанный в [7], который основан на теории глобального поиска, созданной А. С. Стрекаловским (см. [4, 8]). Этот алгоритм состоит из нескольких этапов, основным из которых является локальный поиск с использованием точек аппроксимации поверхности уровня выпуклой функции, задающей базовую невыпуклость в билинейной задаче [4, 5, 7, 8]. В свою очередь, локальный поиск заключается в последовательном решении специальных задач линейного программирования (ЛП) [5, 7, 8]. Таким образом, основными "кирпичиками" алгоритма глобального поиска являются задачи ЛП, на решение которых, как показывают предыдущие исследования [5, 7, 8], тратится бóльшая часть времени его работы. Следовательно, быстрое решение упомянутых задач ЛП является одним из важнейших элементов успешного глобального поиска. В настоящей работе для решения этой проблемы предлагается использовать известный коммерческий пакет ILOG CPLEX [10]. Специалисты оценивают его как один из самых эффективных в настоящее время пакетов по решению "базовых" задач.

Кроме того, было замечено, что исследуемый алгоритм глобального поиска обладает естественной параллельной структурой. Действительно, процедура локального поиска на каждой итерации алгоритма глобального поиска осуществляется многократно из взаимно независимых стартовых точек. Основываясь на этом факте, нами выполнена параллелизация алгоритма для биматричных игр.

Статья организована следующим образом. В разделе 2 напоминается постановка задачи поиска ситуаций равновесия по Нэшу и формулируется теорема эквивалентности. В разделе 3 изложена схема

[1] Институт динамики систем и теории управления Сибирского отделения РАН, ул. Лермонтова, 134, 664033, Иркутск; e-mail: vil@icc.ru, xenia.klimentova@icc.ru, anor@icc.ru

используемого метода локального поиска. Раздел 4 посвящен описанию всех этапов глобального поиска. В разделе 5 представлены результаты тестирования рассмотренного алгоритма глобального поиска с применением пакета ILOG CPLEX. В разделе 6 изложена схема параллелизации. И, наконец, раздел 7 посвящен численному тестированию параллельного алгоритма.

2. Постановка задачи. Рассмотрим биматричную игру $\Gamma(A, B)$, где A и B — матрицы выигрышей игроков размерности $(m \times n)$ [3, 6]. Будем понимать решение биматричной игры в смысле отыскания ситуации равновесия по Нэшу [5, 7, 8].

Определение. *Ситуацией равновесия* (*по Нэшу*) в биматричной игре $\Gamma(A, B)$ в смешанных стратегиях называется ситуация $(x^*, y^*) \in S_m \times S_n$, которая удовлетворяет следующим неравенствам:

$$\langle x^*, Ay^* \rangle \geqslant \langle x, Ay^* \rangle \quad \forall x \in S_m, \quad \langle x^*, By^* \rangle \geqslant \langle x^*, By \rangle \quad \forall y \in S_n.$$

Здесь S_m и S_n — канонические симплексы:

$$S_m \stackrel{\triangle}{=} \left\{ x = (x_1, \ldots, x_m) \in \mathbb{R}^m \mid x_i \geqslant 0, \sum_{i=1}^{m} x_i = 1 \right\}, \quad S_n \stackrel{\triangle}{=} \left\{ y = (y_1, \ldots, y_n) \in \mathbb{R}^n \mid y_j \geqslant 0, \sum_{j=1}^{n} y_j = 1 \right\}.$$

Множество всех таких ситуаций обозначается через $NE(\Gamma)$. Согласно теореме Нэша [3, 5], в любой биматричной игре $NE(\Gamma) \neq \varnothing$.

Как известно [6], поиск ситуации равновесия по Нэшу в биматричной игре равносилен поиску глобального решения следующей билинейной задачи математического программирования:

$$F(x, y, \alpha, \beta) \stackrel{\triangle}{=} \langle x, (A + B)y \rangle - \alpha - \beta \uparrow \max, \quad (x, y, \alpha, \beta) \in X \times Y \stackrel{\triangle}{=} D, \tag{$\mathcal{P}$}$$

где $X = \{(x, \beta) \in S_m \times \mathbb{R} | x^T B \leqslant \beta e_n\}$, $Y = \{(y, \alpha) \in S_n \times \mathbb{R} | Ay \leqslant \alpha e_m\}$, $e_p = (1, 1, \ldots, 1)^T \in \mathbb{R}^p$, $p = m, n$.

Теорема [6]. *Допустимая точка $(x^*, y^*, \alpha_*, \beta_*) \in \mathbb{R}^{m+n+2}$ является глобальным решением в задаче $(\mathcal{P})$ тогда и только тогда, когда $F(x^*, y^*, \alpha_*, \beta_*) = 0$ и $(x^*, y^*) \in NE(\Gamma)$. При этом величины α_*, β_* являются выигрышами первого и второго игроков соответственно: $\alpha_* = \langle x^*, Ay^* \rangle$, $\beta_* = \langle x^*, By^* \rangle$.*

Нетрудно видеть, что целевая функция задачи $(\mathcal{P})$ является невогнутой, а следовательно, сама задача $(\mathcal{P})$ является невыпуклой. Кроме того, отметим, что функция $F(\cdot)$ является d.c. функцией, т.е. представима в виде разности двух выпуклых функций [5, 7]. Следовательно, для ее решения можно применить известную стратегию глобального поиска для задач d.c. программирования [4, 5, 7], что и будет продемонстрировано в последующих разделах статьи.

Отметим, что, поскольку речь идет о реализации этой стратегии на вычислительных машинах, для обоснования полученных численных результатов необходимо использовать определение приближенной ситуации равновесия по Нэшу [5].

Определение. Ситуация $(x^0, y^0) \in S_m \times S_n$ называется *ситуацией δ-равновесия* (*по Нэшу*) в биматричной игре $\Gamma(A, B)$, если справедливы следующие неравенства: $\langle x^0, Ay^0 \rangle \geqslant \langle x, Ay^0 \rangle - \delta \ \ \forall x \in S_m$ и $\langle x^0, By^0 \rangle \geqslant \langle x^0, By \rangle - \delta \ \ \forall y \in S_n$.

Множество всех таких ситуаций обозначается через $NE(\Gamma, \delta)$. Известно [5], что с помощью приближенного численного решения задачи $(\mathcal{P})$ можно получить приближенную ситуацию равновесия в игре $\Gamma(A, B)$.

3. Специальные методы локального поиска. Как уже упоминалось, одним из основных модулей алгоритма глобального поиска [4, 8] является поиск критических точек или локальных решений в рассматриваемой невыпуклой задаче, при этом понятие критической точки зависит от структуры конкретной задачи и соответствующего метода локального поиска.

В работе [7] предложено следующее определение критической точки в задаче $(\mathcal{P})$, учитывающее ее специфику.

Определение. Четверку $(\widehat{x}, \widehat{y}, \widehat{\alpha}, \widehat{\beta})$, удовлетворяющую неравенствам

$$F(\widehat{x}, \widehat{y}, \widehat{\alpha}, \widehat{\beta}) \geqslant F(\widehat{x}, y, \alpha, \widehat{\beta}) \quad \forall (y, \alpha) \in Y, \quad F(\widehat{x}, \widehat{y}, \widehat{\alpha}, \widehat{\beta}) \geqslant F(x, \widehat{y}, \widehat{\alpha}, \beta) \quad \forall (x, \beta) \in X, \tag{3.1}$$

будем называть *критической точкой* задачи $(\mathcal{P})$.

Такая критическая точка является частично глобальным максимумом в задаче $(\mathcal{P})$ по парам переменных (x, β) и (y, α). Отметим также, что точка $(\widehat{x}, \widehat{y}, \widehat{\alpha}, \widehat{\beta})$ называется *приближенной критической точкой*, если неравенства (3.1) выполняются с некоторой заданной точностью.

Кроме того, в [5, 7, 8] разработаны два варианта специального метода локального поиска в задаче $(\mathcal{P})$. Они носят название X- и Y-процедур, поскольку для начала работы используют, соответственно, компоненты $x^0 \in \mathbb{R}^m$ и $y^0 \in \mathbb{R}^n$ стартовой точки $(x^0, y^0, \alpha_0, \beta_0)$. Смысл процедур состоит в последовательном решении задач линейного программирования (ЛП) по парам переменных. В случае X-процедуры сначала решается задача ЛП по переменным (y, α), затем — по переменным (x, β). В Y-процедуре эти задачи решаются в другом порядке. Можно показать [5, 7, 8], что за конечное число шагов процесса будет получена приближенная критическая точка задачи $(\mathcal{P})$.

На рис. 1 для примера приведена вычислительная схема Y-процедуры.

Пусть даны начальное приближение $y^0 \in \mathbb{R}^n$, последовательность чисел $\{\rho_s\} > 0$, $s = 1, 2, \ldots$, такие, что $\sum\limits_{s=0}^{\infty} \rho_s < +\infty$, и константы точности $\tau > 0$ и $\tau_1 > 0$, причем $\tau_1 \leqslant \tau/2$ и $\rho_s \leqslant \tau/2 \ \forall s = 1, 2, \ldots$.

Шаг 0. Инициализировать $s := 0$, $y^s := y^0$.
Шаг 1. Найти $\rho_s/2$-решение (x^{s+1}, β_{s+1}) задачи линейного программирования:
$\langle x, (A + B)y^s\rangle - \beta \uparrow \max\limits_{(x,\beta)}$, $(x, \beta) \in X$.
Шаг 2. Найти $\rho_s/2$-решение (y^{s+1}, α_{s+1}) задачи линейного программирования:
$\langle x^{s+1}(A + B), y\rangle - \alpha \uparrow \max\limits_{(y,\alpha)}$, $(y, \alpha) \in Y$.
Шаг 3. Если
$F(x^{s+1}, y^{s+1}, \alpha_{s+1}, \beta_{s+1}) - F(x^{s+1}, y^s, \alpha_s, \beta_{s+1}) \leqslant \tau_1$,
то Stop. При этом $(x^{s+1}, y^s, \alpha_s, \beta_{s+1})$ — τ-критическая точка задачи $(\mathcal{P})$.
Иначе $s := s + 1$ и перейти на шаг 1.

Рис. 1. Y-процедура

4. Алгоритм глобального поиска. Как уже упоминалось в разделе 2, целевая функция задачи $(\mathcal{P})$ является d.c. функцией. Однако для решения конкретных невыпуклых задач с помощью подхода, предложенного в [4], знания только этого факта недостаточно. Необходимо иметь явное разложение целевой функции задачи в виде разности двух выпуклых. Как известно, такие разложения можно производить неединственным способом [4, 9].

В качестве одного из наиболее естественных d.c. представлений билинейной целевой функции задачи $(\mathcal{P})$ можно выбрать, например, следующее [7, 8]: $F(x, y, \alpha, \beta) = f(x, y) - g(x, y, \alpha, \beta)$, где

$$f(x, y) = \frac{1}{4}\left(\|x + Ay\|^2 + \|xB + y\|^2\right), \quad g(x, y, \alpha, \beta) = \frac{1}{4}\left(\|x - Ay\|^2 + \|xB - y\|^2\right) + \alpha + \beta. \tag{4.1}$$

Здесь f и g — выпуклые по совокупности своих переменных функции.

С использованием d.c. разложения (4.1) далее представим один из вариантов алгоритма глобального поиска ситуаций равновесия в биматричных играх, базирующегося на теореме эквивалентности, сформулированной в разделе 2, и стратегии глобального поиска для задач d.c. программирования [4]. Другие варианты алгоритма могут быть найдены в [5, 7, 8]. Основными этапами этого алгоритма, наряду с локальным поиском, являются построение аппроксимации поверхности уровня и одномерный поиск. Схема алгоритма глобального поиска представлена на рис. 2.

Пусть даны точка $(x^0, y^0, \alpha_0, \beta_0) \in \mathbb{R}^{m+n+2}$, числовые последовательности $\{\tau_k\}$, $\{\delta_k\}$: τ_k, $\delta_k > 0$, $k = 0, 1, 2, \ldots$, $\tau_k \downarrow 0$, $\delta_k \downarrow 0$, $(k \to \infty)$, числа $\gamma_- \triangleq \inf(g, D)$ и $\gamma_+ \triangleq \sup(g, D)$, множество направлений $\mathrm{Dir} = \left\{(u^1, v^1), \ldots, (u^N, v^N) \in \mathbb{R}^{m+n} \,|\, (u^s, v^s) \neq 0, s = 1, \ldots, N\right\}$ и параметры алгоритма ν и q. Пусть $\varepsilon > 0$ — заданная точность решения задачи.

Представленная выше схема алгоритма глобального поиска в задаче $(\mathcal{P})$ не является алгоритмом в общепринятом смысле, поскольку некоторые ее этапы не конкретизированы. Дополнительно необходимо реализовать следующие элементы глобального поиска:

1) поиск чисел γ_- и γ_+, задающих границы отрезка одномерного поиска;

2) выбор параметров алгоритма ν и q; первый параметр регулирует точность выполнения неравенства (4.2) на рис. 2, следующего из условий глобальной оптимальности для задачи $(\mathcal{P})$, второй — определяет количество отрезков разбиения интервала одномерного поиска $[\gamma_-, \gamma_+]$;

Шаг 0. Положить $k := 1$, $(\bar{x}^k, \bar{y}^k, \bar{\alpha}_k, \bar{\beta}_k) := (x^0, y^0, \alpha_0, \beta_0)$,
$s := 1$, $\gamma := \gamma_-$, $\triangle\gamma = (\gamma_+ - \gamma_-)/q$.
Шаг 1. Начиная из точки $(\bar{x}^k, \bar{y}^k, \bar{\alpha}_k, \bar{\beta}_k) \in D$, с помощью
X- или Y-процедуры построить τ_k-критическую точку
$(x^k, y^k, \alpha_k, \beta_k) \in D$ в задаче $(\mathcal{P})$. Положить $\zeta_k := F(x^k, y^k, \alpha_k, \beta_k)$.
Шаг 2. Если $\zeta_k \geqslant -\varepsilon$, то Stop, при этом $(x^k, y^k) \in NE(\Gamma, \varepsilon)$.
Шаг 3. По точке $(u^s, v^s) \in$ Dir построить точку $(\bar{u}^s, \bar{v}^s)$ аппроксимации
поверхности уровня функции f, такую, что $f(\bar{u}^s, \bar{v}^s) = \gamma + \zeta_k$.
Вычислить $\bar{\alpha}_s = \max\limits_i (A\bar{v}^s)_i = \alpha(\bar{v}^s)$, $\bar{\beta}_s = \max\limits_j (\bar{u}^s B)_j = \beta(\bar{u}^s)$.
Шаг 4. Если

$$g(\bar{u}^s, \bar{v}^s, \bar{\alpha}_s, \bar{\beta}_s) > \gamma + \nu\gamma, \qquad (4.2)$$

то положить $s := s + 1$ и перейти на шаг 3.
Шаг 5. Начиная из точки $(\bar{u}^s, \bar{v}^s, \bar{\alpha}_s, \bar{\beta}_s)$, построить
δ_k-критическую точку $(\widehat{x}^s, \widehat{y}^s, \widehat{\alpha}_s, \widehat{\beta}_s) \in D$ в задаче $(\mathcal{P})$.
Шаг 6. Если $F(\widehat{x}^s, \widehat{y}^s, \widehat{\alpha}_s, \widehat{\beta}_s) \geqslant -\varepsilon$, то Stop, при этом $(\widehat{x}^s, \widehat{y}^s) \in NE(\Gamma, \varepsilon)$.
Шаг 7. Если $F(\widehat{x}^s, \widehat{y}^s, \widehat{\alpha}_s, \widehat{\beta}_s) \leqslant \zeta_k + \varepsilon$, $s < N$, то положить $s := s + 1$
и перейти на шаг 3.
Шаг 8. Если $F(\widehat{x}^s, \widehat{y}^s, \widehat{\alpha}_s, \widehat{\beta}_s) \leqslant \zeta_k + \varepsilon$, $s = N$ и $\gamma < \gamma_+$, то положить
$\gamma := \gamma + \triangle\gamma$, $s := 1$ и перейти на шаг 3.
Шаг 9. Если $F(\widehat{x}^s, \widehat{y}^s, \widehat{\alpha}_s, \widehat{\beta}_s) > \zeta_k + \varepsilon$, то положить
$(\bar{x}^{k+1}, \bar{y}^{k+1}, \bar{\alpha}_{k+1}, \bar{\beta}_{k+1}) := (\widehat{x}^s, \widehat{y}^s, \widehat{\alpha}_s, \widehat{\beta}_s)$, $\zeta_{k+1} := F(\widehat{x}^s, \widehat{y}^s, \widehat{\alpha}_s, \widehat{\beta}_s)$,
$k := k + 1$, $s := 1$, $\gamma = \gamma_-$ и перейти на шаг 3.
Шаг 10. Если $s = N$, $F(\widehat{x}^s, \widehat{y}^s, \widehat{\alpha}_s, \widehat{\beta}_s) \leqslant \zeta_k + \varepsilon$ $\forall\gamma \in [\gamma_-, \gamma_+]$ (т.е. одномерный
поиск по γ на отрезке $[\gamma_-, \gamma_+]$ закончен), то Stop.

Рис. 2. Схема алгоритма глобального поиска

3) выбор множества направлений Dir;

4) построение точек аппроксимации поверхности уровня с помощью заданного множества направлений.

Перечисленные модули алгоритма были реализованы с учетом имеющегося опыта решения подобного сорта задач [4, 5, 7].

1. Для вычисления интервала одномерного поиска $[\gamma_-, \gamma_+]$ следует решить две задачи: одну на минимум выпуклой функции, другую — на максимум этой же функции. Решение первой задачи может быть выполнено одним из известных методов квадратичного или выпуклого программирования [1]. Вместо максимизации функции g, которая оказывается неограниченной сверху на множестве D, с помощью перебора вершин канонических симплексов производилось решение релаксированной задачи

$$\frac{1}{4}\left(\|x - Ay\|^2 + \|xB - y\|^2\right) \uparrow \max, \quad x \in S_m, \quad y \in S_n. \qquad (4.3)$$

После этого значения переменных $\widetilde{\alpha}$ и $\widetilde{\beta}$ вычислялись по формулам $\widetilde{\alpha} = \max\limits_i (A\widetilde{y})_i$, $\widetilde{\beta} = \max\limits_j (\widetilde{x}B)_j$, где $\widetilde{x}$, $\widetilde{y}$ — решение задачи (4.3).

2. Были использованы следующие варианты значений параметров алгоритма: (а) $q = 10$, $\nu = 0.0$; (б) $q = 20$, $\nu = 0.05$; (в) $q = 100$, $\nu = 0.15$. Если требуется быстро отыскать некоторое приближение к глобальному решению задачи $(\mathcal{P})$, то можно применить вариант (а) выбора параметров. При увеличении ν и q (случаи (б) и (в)) происходит рост точности работы алгоритма, но при этом падает его скорость.

3. В рассматриваемой задаче $(\mathcal{P})$ хорошо зарекомендовали себя следующие множества направлений [5, 7, 8]:

$$\mathrm{Dir}\,1 = \left\{(e^i, e^j),\ i = 1, \ldots, m,\ j = 1, \ldots, n\right\}, \qquad (4.4)$$

$$\mathrm{Dir}\,2 = \left\{(e^i + x^k, e^j + y^k),\ i = 1, \ldots, m,\ j = 1, \ldots, n\right\}, \qquad (4.5)$$

$$\mathrm{Dir}\,3 = \left\{(a^j + e_m, b^i + e_n),\ i = 1, \ldots, m,\ j = 1, \ldots, n\right\}, \qquad (4.6)$$

где e^p $(p = i, j)$ — стандартные базисные векторы; x^k, y^k — векторы, входящие в текущую критическую точку, полученную во время локального поиска; $a^j \in \mathbb{R}^m$ — столбцы матрицы A; $b^i \in \mathbb{R}^n$ — строки матрицы B и e_p $(p = m, n)$ — векторы, состоящие из единиц.

Отметим, что в этих множествах используется как информация о целевой функции и допустимом множестве задачи (матрицы A и B, вершины канонических симплексов e^i и e^j), так и информация, полученная в процессе вычислений (текущая критическая точка). Как показывает предыдущий вычислительный опыт [4, 5, 7, 8], при построении множеств направлений необходимо использовать всю возможную информацию, заключенную в задаче.

Нетрудно видеть, что количество точек в аппроксимациях, построенных с помощью этих множеств направлений, достаточно велико и равно mn. Для сокращения количества точек использовался прием, предложенный в [5, 7]. Рассмотрим некоторое множество направлений, состоящее из mn точек: $\mathrm{Dir}\,0 = \{(u^i, v^j) \in \mathbb{R}^{m+n} \mid (u^i, v^j) \neq 0, i = 1, \ldots, m, j = 1, \ldots, n\}$. Найдем в матрице A две строки с максимальной суммой элементов. Обозначим их через i_1 и i_2. Аналогично в матрице B через j_1 и j_2 обозначим два столбца с максимальной суммой элементов. Тогда новое множество направлений строится по следующему правилу: $\mathrm{Cut}\,(\mathrm{Dir}\,0) = \{(u^{i_1}, v^j), (u^{i_2}, v^j), j = 1, \ldots, n; (u^i, v^{j_1}), (u^i, v^{j_2}), i = 1, \ldots, m\}$. Количество точек в множестве $\mathrm{Cut}\,(\mathrm{Dir}\,0)$ равно $2(m + n)$, и при $m > 4$ и $n > 4$ мощность этого множества становится меньше мощности множества $\mathrm{Dir}\,0$.

При реализации схемы алгоритма глобального поиска данный прием сокращения аппроксимаций был применен к описанным множествам направлений $\mathrm{Dir}\,1$, $\mathrm{Dir}\,2$ и $\mathrm{Dir}\,3$.

4. Для построения точек аппроксимации поверхности уровня на шаге 3 алгоритма использовался следующий стандартный подход [4, 5, 7], при котором по вектору $(u^s, v^s) \in \mathrm{Dir}$ строится коллинеарный ему вектор $(\bar{u}^s, \bar{v}^s)$, лежащий на требуемой поверхности уровня: $(\bar{u}^s, \bar{v}^s) = \lambda(u^s, v^s)$, $s = 1, \ldots, N$, $\lambda = \lambda_s(\zeta, \gamma) = \pm\sqrt{\dfrac{\gamma + \zeta}{f(u^s, v^s)}}$. Легко проверить, что $f(\bar{u}^s, \bar{v}^s) = \gamma + \zeta$. Чтобы предотвратить чрезмерное увеличение числа точек в построенной аппроксимации, при численном решении задачи $(\mathcal{P})$ использовалось только значение λ со знаком плюс.

Итак, все этапы глобального поиска описаны. Прежде чем переходить к разработке параллельного варианта алгоритма, представим результаты численного тестирования описанного последовательного варианта.

5. Численное тестирование последовательного варианта алгоритма. Тестирование алгоритма поиска приближенных ситуаций равновесия по Нэшу, рассмотренного в предыдущих разделах, производилось на сериях случайно сгенерированных задач размерности от 50×50 до 400×400. Элементы матриц A и B выбирались равномерно распределенными на отрезке $[-n, n]$, где $n = m$ — число чистых стратегий игроков в игре $\Gamma(A, B)$.

Программа, реализующая указанный алгоритм, написана на языке C++ для операционной системы Linux. Ее запуск производился на персональном компьютере с процессором Intel Pentium 4, 3.2 ГГц, 1 ГБ ОЗУ.

Стартовая точка для всех решаемых задач выбиралась следующим образом: $x_i^0 = \dfrac{1}{m}$, $i = 1, \ldots, m$; $y_j^0 = \dfrac{1}{n}$, $j = 1, \ldots, n$; $\alpha_0 = \max\limits_i (Ay^0)_i$; $\beta_0 = \max\limits_j (x^0 B)_j$. Задачи решались с точностью $\varepsilon = 10^{-4}$.

Для решения каждой задачи последовательно перебирались различные комбинации наборов параметров алгоритма q и ν и множеств направлений Dir (см. раздел 4). Если в результате перебора всех вариантов не была достигнута заданная точность, то задача считалась нерешенной.

Решение подзадач линейного программирования в процедурах локального поиска (раздел 3), а также решение квадратичных задач для поиска значений γ_- (раздел 4) осуществлялось с помощью пакета прикладных программ ILOG CPLEX 9.1 [10]. Этот пакет считается специалистами одним из лучших пакетов, предназначенных для решения такого сорта задач. Как уже упоминалось, применение современных разработок для решения вспомогательных задач в алгоритмах помогает существенно сокращать время работы.

Результаты решения задач представлены в табл. 1, в которой использованы следующие обозначения: Name — имя задачи, имеющее вид m-n_number, где $m \times n$ — размерность матриц в решаемых задачах, number — номер задачи в серии; F_0 — значение целевой функции в стартовой точке; X, Y — используемая при реализации алгоритма процедура локального поиска; Dir — номер множества направлений, с помощью которого была решена задача; LP — количество решенных задач линейного программирования; Loc — количество запусков процедуры локального поиска; Time — общее время работы программы в секундах.

Таблица 1

Последовательный алгоритм

Name	F_0	X				Y			
		Dir	LP	Loc	Time	Dir	LP	Loc	Time
50-50_1	-13.70	1	56	10	0.09	1	32	4	0.07
50-50_2	-17.36	1	12	2	0.06	1	16	2	0.06
50-50_3	-18.60	1	38	8	0.08	1	46	9	0.08
50-50_4	-19.50	1	18	2	0.06	1	18	2	0.06
50-50_5	-18.17	2	326	53	0.20	2	212	25	0.19
50-50_6	-19.73	2	142	16	0.13	2	72	10	0.11
50-50_7	-15.56	2	410	49	0.29	1	20	4	0.07
50-50_8	-24.73	2	614	61	0.30	1	12	2	0.06
50-50_9	-19.60	1	26	5	0.08	2	66	9	0.10
50-50_10	-19.34	1	10	2	0.07	2	50	8	0.11
70-70_1	-20.43	2	1890	193	1.79	2	210	20	0.46
70-70_2	-25.83	2	324	34	0.53	2	318	32	0.51
70-70_3	-22.90	2	632	62	0.74	2	106	12	0.30
70-70_4	-22.19	1	16	2	0.14	1	18	2	0.15
70-70_5	-24.03	2	690	83	0.80	1	10	1	0.02
70-70_6	-23.87	2	1062	137	1.06	2	1058	108	1.41
70-70_7	-22.33	2	30	6	0.2	1	18	2	0.15
70-70_8	-27.90	2	74	9	0.24	2	286	29	0.47
70-70_9	-25.35	1	18	3	0.16	2	508	61	0.86
70-70_10	-28.67	2	170	18	0.36	2	244	25	0.53
100-100_1	-24.44	2	504	45	1.35	1	38	4	0.53
100-100_2	-26.72	1	12	2	0.43	1	18	2	0.45
100-100_3	-27.01	1	28	4	0.48	1	20	4	0.48
100-100_4	-27.17	1	22	2	0.44	1	16	2	0.44
100-100_5	-26.81	1	18	2	0.43	1	40	5	0.54
150-150_1	-39.55	2	238	22	5.37	2	110	12	4.54
150-150_2	-42.51	2	114	12	4.28	2	648	52	9.01
150-150_3	-34.58	1	46	5	3.33	2	1576	130	17.24
150-150_4	-38.21	1	24	2	2.77	2	1970	159	19.04
150-150_5	-33.66	2	6560	583	33.37	2	612	56	9.10
200-200_1	-38.60	2	946	66	21.11	1	36	5	8.85
200-200_2	-47.65	2	1684	114	39.43	2	2680	236	50.90
200-200_3	-49.07	2	4506	401	74.74	2	282	24	14.71
200-200_4	-45.29	2	606	41	19.25	2	508	36	20.50
200-200_5	-37.10	1	22	2	7.52	1	24	2	7.70
300-300_1	-50.23	2	3022	226	160.86	2	2806	208	286.43
300-300_2	-64.47	1	52	3	41.38	1	46	2	38.19
300-300_3	-52.41	1	42	5	41.86	2	4940	342	461.11
300-300_4	-56.90	2	8172	596	491.93	2	7564	502	591.51
300-300_5	-54.25	2	2954	178	230.15	2	6798	395	557.58
400-400_1	-73.96	2	9046	606	755.92	1	58	5	105.78
400-400_2	-73.17	2	18924	1205	1281.93	2	3334	209	826.50
400-400_3	-64.05	1	28	3	102.40	1	58	4	108.43
400-400_4	-79.39	2	25930	980	1982.40	2	4782	258	1221.82
400-400_5	-58.86	2	3192	203	574.41	2	14564	891	3089.58

Прежде всего, по результатам вычислительного эксперимента отметим, что с заданной точностью были решены все рассмотренные задачи, причем для этого потребовался только первый набор параметров ($q = 10$, $\nu = 0.0$).

Кроме того, нетрудно заметить, что в ходе работы потребовались только множества направлений Dir 1 и Dir 2, что еще раз подтверждает их хорошие практические свойства.

Далее отметим существенное отличие по времени работы программы с применением X- и Y-процедур. Например, в задачах 300-300_3, 400-400_5 быстрее удалось найти решение с помощью X-процедуры,

в то время как на задачах 200-200_3, 400-400_1 быстрее было найдено решение с использованием Y-процедуры. При этом легко видеть, что время работы напрямую зависит от количества запусков процедуры локального поиска (столбец Loc), и, соответственно, от количества решенных задач линейного программирования (столбец LP). Таким образом, любая из этих процедур может быть использована при реализации алгоритма глобального поиска.

Хотелось бы также отметить значительное уменьшение времени решения и увеличение размерности решаемых задач по сравнению с предыдущими результатами, опубликованными в [7, 8], где размерность матриц не превышала 200. Что же касается времени решения, то, например, для задач размерностей 100×100 преимущество данной реализации составляет десятки тысяч раз. Такое существенное уменьшение времени работы можно объяснить рядом причин. Во-первых, реализована несколько иная схема алгоритма глобального поиска. В данном варианте алгоритма глобального поиска при переходе на новую поверхность уровня точки аппроксимации исследуются начиная с первой, в отличие от [7, 8], где при таком переходе рассмотренные ранее точки отбрасываются. Во-вторых, был применен другой порядок перебора параметров алгоритма ν и q и множеств направлений Dir, который оказался более эффективным. Однако основной выигрыш, на наш взгляд, был достигнут благодаря использованию пакета ILOG CPLEX для решения вспомогательных задач ЛП.

Таким образом, вычислительный эксперимент по тестированию рассмотренного варианта алгоритма глобального поиска подтверждает эффективность использования современных технологий для решения вспомогательных задач.

6. Параллельный алгоритм глобального поиска. Вернемся к алгоритму глобального поиска (см. раздел 4). Нетрудно заметить, что данный алгоритм имеет естественную параллельную структуру. Действительно, на каждой итерации на одной и той же поверхности уровня ведется поиск критических точек (шаг 5), начиная из взаимно независимых начальных приближений (шаг 3). Следовательно, данный процесс может быть параллелизован. При реализации параллельного алгоритма использовалась система обмена сообщениями MPI [11 – 14]. MPI-программа — это множество параллельных взаимодействующих процессов. Все процессы порождаются один раз, образуя параллельную часть программы. В ходе выполнения MPI-программы порождение дополнительных процессов или уничтожение существующих не допускается. Каждый процесс работает в своем адресном пространстве, никаких общих переменных или данных в MPI нет. Основным способом взаимодействия между процессами является явная посылка сообщений [14]. Для реализации такого взаимодействия использовалась стандартная модель Master–Slave [13]. В рамках этой модели выделяется один главный процесс — Master, осуществляющий управление всеми остальными, которые называются Slave-процессами. Каждый параллельный процесс обычно (и в данной работе) запускается на отдельном процессоре распределенной вычислительной системы. Поэтому далее будут использоваться понятия Master-процессор и Slave-процессор.

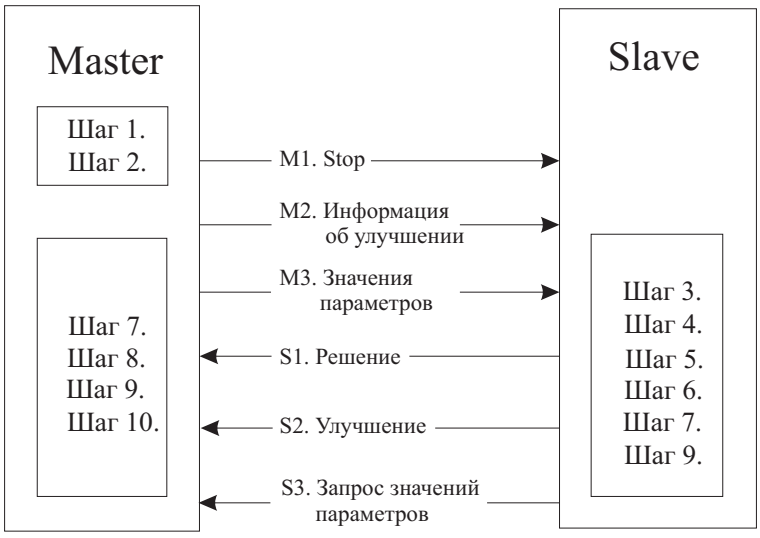

Рис. 3. Схема Master–Slave

Схема параллельного алгоритма глобального поиска представлена на рис. 3.

Из стартовой точки Master-процессор осуществляет локальный поиск (шаг 1 алгоритма глобального поиска). Если в результате не было найдено решение (ε-равновесие по Нэшу, шаг 2), то далее Master формирует множество параметров, необходимых для начала работы Slave-процессоров, а именно текущие значения γ, ζ_k, значения параметра ν, а также номера точек из множества направлений. Каждому Slave-процессору отсылается свой набор номеров точек. Отметим, что чем меньше точек в отсылаемых наборах, тем большее количество обменов сообщениями происходит между Master- и Slave-процессорами. Затем Master отсылает эти множества параметров Slave-процессорам. Получив их, каждый Slave-процессор по номерам последовательно вычисляет точки аппроксимации поверхности уровня (шаг 3), проверяет для

каждой условие (4.2) (шаг 4) и в случае, если точка удовлетворяет этому условию, осуществляет локальный поиск (шаг 5). После этого Slave-процессор проверяет, получено ли решение (шаг 6) или получено ли улучшение (шаг 9). Далее в зависимости от полученных результатов Slave-процессор выполняет одно из следующих действий.

S1. Если на шаге 6 было получено решение (ε-равновесие по Нэшу), то он останавливает свою работу и посылает это решение Master-процессору.

S2. Если получено улучшение, т.е. точка, лучшая по значению целевой функции, чем текущая критическая (шаг 9), то Slave-процессор отправляет новую точку и ожидает новый набор параметров.

S3. Переходит к следующей точке из имеющегося набора номеров точек множества направлений (шаг 7). Если все точки из имеющегося набора уже рассмотрены, то Slave-процессор отправляет запрос на отправку нового набора параметров для работы.

Master-процессор анализирует все приходящие к нему от Slave-процессоров сообщения и производит одно из следующих действий.

M1. Если на каком-либо из Slave-процессоров было получено решение, то Master отправляет всем остальным сигнал о завершении работы.

M2. В случае получения улучшения работа всех Slave-процессоров также останавливается и им рассылаются новая критическая точка и новые значения параметров.

M3. В случае же завершения каким-либо Slave-процессором работы без получения улучшения Master отправляет ему следующий набор параметров, необходимых для продолжения работы.

По завершении работы алгоритма на шаге 10 Master останавливает все Slave-процессоры.

При реализации параллельных алгоритмов, как правило, проводится исследование параллельного ускорения с целью определения эффективности реализованной параллелизации. Параллельное ускорение вычисляется по формуле t_0/t_i, $i = 1, 2 \ldots, n$, где t_0 — время работы последовательного алгоритма, а t_i — время решения задачи параллельным алгоритмом с использованием i процессоров. Говорят, что достигнуто линейное ускорение, если $t_0/t_i \approx i \ \ \forall i = 1, \ldots, n$. Такое ускорение свидетельствует о хорошей параллелизации алгоритма.

Таблица 2

Мощность наборов номеров точек

Subset	Slave	400-400_1			200-200_2		
		LP	Loc	Time	LP	Loc	Time
1	1	10584	601	853.4	1652	113	64.03
10	1	10584	601	849.35	1652	113	62.92
100	1	10584	601	849.09	1652	113	62.5
1000	1	10584	601	853.24	1652	113	63.72
10000	1	10584	601	855.29	1652	113	63.91
1	4	10608	603	275.81	1684	115	18.87
10	4	10606	602	273.94	1674	114	18.86
100	4	9324	520	213.5	2742	205	22.11
1000	4	23212	1480	546.18	468	35	5.31
10000	4	31214	2084	508.16	478	37	5.33
1	8	10670	607	167.95	1766	121	11.25
10	8	10806	612	161.11	1596	108	10.22
100	8	8804	491	119.86	6718	491	24.27
1000	8	29892	2024	321.77	334	30	3.41
10000	8	35828	3372	180.54	250	22	3.33

7. Тестирование параллельного алгоритма глобального поиска. Описанный в предыдущем разделе параллельный алгоритм глобального поиска так же, как и последовательный, был реализован на языке программирования C++ для Linux с помощью системы обмена сообщениями MPI. Тестирование проводилось на кластере МВС 1000/16, установленном в Институте динамики систем и теории управления СО РАН. Основные технические характеристики кластера таковы: 16 вычислительных узлов, 32 процессора Intel Xeon (2,667 GHz); пиковая производительность — 170 миллиардов операций в секунду; суммарный объем оперативной памяти на узлах — 32 GB; суммарный объем дисковой памяти на узлах — 1280 GB; интерконнект — Myrinet & Gigabit Ethernet. Дополнительная информация об используемом кластере может быть найдена по адресу www.mvs.icc.ru.

Таблица 3

Параллельное ускорение

Slave	LP	Loc	Time	Speedup	LP	Loc	Time	Speedup
	400-400_1				300-300_4			
0	9046	606	872.19	1.00	8172	596	636.99	1.00
1	10584	601	856.69	1.02	8168	595	632.39	1.01
2	10572	600	441.04	1.98	8184	596	322.94	1.97
3	10532	598	378.37	2.31	8208	598	280.52	2.27
4	10558	599	272.31	3.20	8216	598	199.08	3.20
5	10480	595	246.57	3.54	8198	597	182.47	3.49
6	10506	596	198.24	4.40	8142	592	142.24	4.48
7	10628	604	195.59	4.46	8090	588	133.15	4.78
8	10512	596	160.28	5.44	8144	592	114.26	5.57
	200-200_2				150-150_5			
0	1684	114	63.72	1.00	6560	583	46.54	1.00
1	1652	113	62.92	1.01	6556	582	46.33	1.00
2	1652	113	38.32	1.66	6656	592	22.64	2.06
3	1744	120	26.89	2.37	6722	599	16.67	2.79
4	1674	114	18.86	3.38	6852	611	13.02	3.57
5	1652	113	15.45	4.12	6868	615	13.59	3.42
6	1684	114	12.65	5.04	6942	620	10.62	4.38
7	1708	118	11.98	5.32	7216	647	8.97	5.19
8	1596	108	10.22	6.23	7332	659	8.13	5.72

Тестирование осуществлялось в несколько этапов. Сначала было проведено исследование двух параметров параллелизации. Это, во-первых, мощность наборов номеров точек множеств направлений Dir, распределяемых между Slave-процессорами, и, во-вторых, количество используемых Slave-процессоров. В связи с этим целью *первого этапа* было изучение зависимости скорости и эффективности работы алгоритма от мощности наборов. На *втором этапе* было проведено исследование параллельного ускорения (см. раздел 6), а также сравнение последовательного и параллельного алгоритмов. По результатам исследования параметров параллелизации на *третьем этапе* решались задачи больших размерностей.

Отметим, что при тестировании параллельного алгоритма использовалась только X-процедура локального поиска. Остальные детали реализации аналогичны последовательному алгоритму (см. раздел 5).

Итак, перейдем к описанию первого этапа тестирования параллельного алгоритма глобального поиска, на котором исследовалась зависимость времени решения задачи от мощности наборов номеров точек множеств направлений, распределяемых между Slave-процессорами. Исследуем эту зависимость на примере задач 400-400_1 и 200-200_2, которые были решены при тестировании последовательного алгоритма. Результаты представлены в табл. 2. В столбце Subset указано количество номеров точек в наборе, в столбце Slave — число используемых Slave-процессоров. Остальные обозначения имеют тот же смысл, что и в табл. 1.

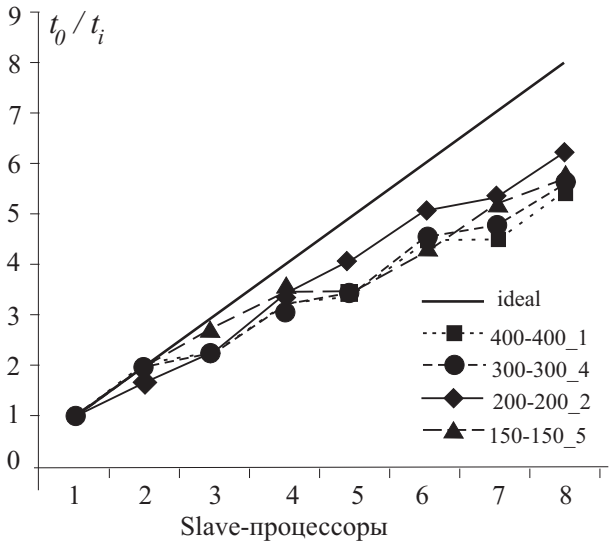

Рис. 4. Параллельное ускорение

В случае, когда используется один Slave-процессор, время работы программы при различной мощности наборов номеров остается практически неизменным. Таким образом, можно сделать вывод о том, что в данном случае время обмена данными между Master- и Slave-процессорами незначительно по сравнению со временем решения задачи в целом.

При использовании четырех и восьми Slave-процессоров возникают так называемые эффекты слу-

чайного “ускорения” и “замедления” [13], которые связаны с тем, что точка множества направлений, с помощью которой удается найти глобальное решение, может попасть в начало или в конец набора номеров точек, что приводит к сравнительно небольшому или, наоборот, неоправданно большому числу запусков процедуры локального поиска. Например, при отправке наборов по 100 номеров точек аппроксимации задача 400-400_1 решалась быстрее по сравнению с другими вариантами. Это связано с тем, что в данном случае для решения задачи потребовался лишь 491 запуск процедуры локального поиска (по сравнению с 601 в случае одного процессора). Для задачи же 200-200_2, напротив, при отправке 100 номеров точек задача решалась дольше всего.

В случае же 1000 и 10 000 точек в наборе наблюдается обратный эффект. Такое большое число исследуемых точек вызвало неоправданно большое количество запусков процедуры локального поиска для задачи 400-400_1, что существенно отразилось на времени решения. В то же время, задача 200-200_2 при тех же 1000 и 10 000 точек в наборах решалась намного быстрее, так как для этого потребовалось менее 40 запусков процедуры локального поиска во всех случаях (в случае одного процессора для этой задачи локальный поиск был запущен 113 раз). Таким образом, во избежание таких эффектов “ускорения” и “замедления” было решено выбрать мощность наборов номеров точек аппроксимации равной 10. При таком значении этого параметра указанных эффектов замечено не было, а время решения каждой задачи было немного меньше, чем при отсылке точек по одной.

Таблица 4

Задачи больших размерностей

Name	Dir	LP	Loc	Time
500-500_1	2	45978	2833	2407.51
500-500_2	2	28762	1803	1130.34
500-500_3	2	49164	2913	2315.12
500-500_4	2	34982	1894	1519.31
500-500_5	2	18648	1048	1375.84
600-600_1	2	61992	3749	6074.57
600-600_2	2	21514	1275	2457.79
600-600_3	2	24932	1236	3787.27
600-600_4	2	42578	2387	4488.33
600-600_5	1	38	2	79.23
700-700_1	1	56	7	171.77
700-700_2	2	1244	58	310.72
700-700_3	2	12428	711	1364.1
700-700_4	2	32092	1793	6054.11
700-700_5	2	45824	2331	9317.1
800-800_1	2	85302	4536	25371.26
800-800_2	2	68342	3280	15118.93
800-800_3	2	82202	4882	10497.71
800-800_4	2	51394	2795	14107.79
800-800_5	1	60	5	395.2
900-900_1	2	106388	5512	41440.38
900-900_2	2	11494	529	4857.9
900-900_3	1	44	2	319.38
900-900_4	2	114826	5889	47801.66
900-900_5	1	88	8	360.09
1000-1000_1	1	62	2	472.61
1000-1000_2	1	68	5	482.35
1000-1000_3	2	106212	5156	48721.54

Теперь, когда первый параметр зафиксирован, можно переходить к исследованию параллельного ускорения, т.е. к исследованию зависимости времени решения задачи от числа используемых процессоров. Поскольку рамки статьи не позволяют привести результаты в полном объеме, опишем их на примере четырех задач. Две из них уже рассматривались на первом этапе — задачи 400-400_1 и 200-200_2. Кроме того, были добавлены задачи 150-150_5 и 300-300_4. Результаты второго этапа параллельного вычислительного эксперимента представлены в табл. 3. Заметим, что здесь при Slave = 0 указано время решения этих задач с помощью последовательного алгоритма, запущенного на одном из узлов кластера. В столбце Speedup записано ускорение, которое, как уже упоминалось в разделе 6, вычисляется по формуле t_0/t_i, $i = 1, \ldots, 8$, где t_i — время решения задачи с i процессорами, а t_0 — время решения задачи последовательным алгоритмом.

По результатам, представленным в табл. 3, отметим, что время решения задачи с помощью последовательного варианта алгоритма и время работы параллельной программы, запущенной с одним Slave-процессором, отличаются незначительно, что свидетельствует о хороших качествах проведенной параллелизации. Что касается использования нескольких Slave-процессоров, на рис. 4 представлены графики зависимости ускорения от их числа. Здесь отметим, что, несмотря на то, что параллельного линейного ускорения достичь не удалось (жирная линия без маркеров), время решения для всех задач уменьшается весьма существенно. В случае восьми процессоров задачи решаются в 5–6 раз быстрее.

Наконец, на третьем этапе тестирования параллельного алгоритма решались задачи больших размерностей от 500×500 до 1000×1000 с помощью восьми Slave-процессоров (т.е. самым быстрым способом). Результаты решения задач этого этапа представлены в табл. 4.

Отметим, что здесь, так же как при тестировании последовательного алгоритма (см. раздел 5), все задачи были решены с помощью первого и второго множеств направлений (столбец Dir). Причем время решения тех задач, где использовалось только первое множество направлений, в несколько раз меньше

времени решения задач таких же размерностей, но для которых потребовалось второе множество направлений. Хотелось бы также отметить существенно меньшее количество решенных задач линейного программирования и меньшее число запусков процедуры локального поиска для этих задач. Особенно интересны задачи 900-900_3, 900-900_5, 1000-1000_1 и 1000-1000_2. Время решения этих задач чрезвычайно мало для таких больших размерностей. Таким образом, для параллельного алгоритма сохраняются тенденции, которые были отмечены при тестировании последовательного алгоритма.

Общее время решения всех задач данного этапа составляет 70 часов 13 минут. По примерной оценке, для решения этих задач последовательным алгоритмом потребовалось бы более 350 часов.

8. Заключение. В статье исследуется известный алгоритм глобального поиска ситуаций равновесия по Нэшу в биматричных играх [5, 7, 8]. Благодаря использованию современного программного комплекса ILOG CPLEX 9.1 для решения вспомогательных задач линейного программирования на шагах алгоритма при тестировании последовательного варианта алгоритма удалось существенно улучшить опубликованные в [7, 8] результаты по решению биматричных игр.

Кроме того, была проведена эффективная параллелизация рассмотренного алгоритма, позволившая уменьшить время решения задач в 5–6 раз при использовании восьми процессоров. Тестирование параллельного алгоритма проводилось на задачах больших размерностей. Удалось решить за разумное время биматричные игры размерности до 1000×1000.

Таким образом, результаты, представленные в статье, подтверждают возможность и эффективность параллелизации алгоритма глобального поиска, разрабатываемого для решения различных невыпуклых задач [4].

Авторы благодарят профессора А. С. Стрекаловского за ценные советы, позволившие улучшить содержание работы.

СПИСОК ЛИТЕРАТУРЫ

1. *Васильев Ф.П.* Методы оптимизации. М.: Факториал Пресс, 2002.
2. *Horst R., Tuy H.* Global optimization. Deterministic approaches. Berlin: Springer-Verlag, 1993.
3. *Васин А.А., Морозов В.В.* Теория игр и модели математической экономики. М.: МАКС Пресс, 2005.
4. *Стрекаловский А.С.* Элементы невыпуклой оптимизации. Новосибирск: Наука, 2003.
5. *Стрекаловский А.С., Орлов А.В.* Биматричные игры и билинейное программирование. М.: Физматлит, 2007 (в печати).
6. *Mills H.* Equilibrium points in finite games // J. of the Society for Industrial and Applied Mathematics. 1960. **8**, N 2. 397–402.
7. *Орлов А.В., Стрекаловский А.С.* О численном поиске ситуаций равновесия в биматричных играх // Журн. вычисл. матем. и матем. физики. 2005. **45**, № 6. 983–997.
8. *Strekalovsky A.S., Orlov A.V.* A new approach to nonconvex optimization // Вычислительные методы и программирование. 2007. **8**, № 2. Раздел 1. 160–176 (http://num-meth.srcc.msu.ru/).
9. *Hiriart-Urruty J.-B.* Generalized differentiability, duality and optimization for problems dealing with difference of convex functions // Lecture Notes in Economics and Mathematical Systems. Vol. 256. Berlin: Springer-Verlag, 1985. 37–69.
10. CPLEX user's manual. ILOG, Inc. Version 9.1. 2005 (www.ilog.com).
11. Installation and User's Guide to MPICH. Argonne National Laboratory, Univ. of Chicago. Argonne, 2003.
12. MPI standard 1.1. Univ. of Tennessee. Knoxville, 1995.
13. *Gropp W., Lusk E., Skjellum A.* Using MPI: portable parallel programming with the message-passing interface. Boston: MIT Press, 1999.
14. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб.: БХВ-Петербург, 2002.

Поступила в редакцию
06.06.2007