Библиотечка журнала
И Т № 7 2012
ПРИЛОЖЕНИЕ К ЖУРНАЛУ «ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ» № 7/2012

УДК 519.6

Карпенко А. П.

Популяционные алгоритмы глобальной поисковой оптимизации. Обзор новых и малоизвестных алгоритмов

Работа представляет собой обзор новых и малоизвестных популяционных алгоритмов, предназначенных для решения задачи глобальной непрерывной оптимизации. Представлены бактериальные, светлячковые, сорняковые, кукушкины, обезьяньи и ряд других алгоритмов, вдохновленных живой природой. Кроме того, в работе рассмотрены алгоритмы, инспирированные неживой природой — гармонические, гравитационные и электромагнитные, а также некоторые другие алгоритмы.

Ключевые слова: *глобальная непрерывная оптимизация; популяционные алгоритмы; алгоритмы, вдохновленные живой природой; алгоритмы, инспирированные неживой природой*

Karpenko A. P. *Population Algorithms for Global Continuous Optimization. Review of New and Little-Known Algorithms*

The paper presents a review of new and little-known population algorithms for global continuous optimization. We discuss bacterial optimization algorithms, firefly and glowworm swarm optimization algorithms, invasive weed optimization algorithm, cuckoo search algorithm, monkey search algorithm and so on. An additional we discuss some inspired of the lifeless nature algorithms: harmony search algorithm, gravitational search algorithm, electromagnetism-like algorithm and some other algorithms.

Keywords*: global continuous optimization; population algorithms; algorithms inspired of the wildlife; algorithms inspired of the lifeless nature*

СОДЕРЖАНИЕ

Введение

Многие задачи, возникающие в таких фундаментальных науках, как физика, химия и молекулярная биология, а также во многих прикладных науках, сводятся к задачам непрерывной глобальной оптимизации. Особенностями таких задач часто являются нелинейность, недифференцируемость, многоэкстремальность, овражность, отсутствие аналитического выражения и высокая вычислительная сложность оптимизируемых функций, высокая размерность пространства поиска, сложная топология области допустимых значений и т. д.

Для эффективного решения указанных задач в 80-х годах прошлого века начали интенсивно разрабатывать стохастические поисковые алгоритмы оптимизации. В данной работе мы рассматриваем класс таких алгоритмов, которые в разных публикациях называют поведенческими, интеллектуальными, метаэвристическими, вдохновленными (инспирированными) природой, роевыми, многоагентными, популяционными (*population*) и т. д. Будем использовать последний термин как, с нашей точки зрения, наиболее отвечающий сути этих алгоритмов.

Подавляющее большинство рассматриваемых алгоритмов представлено в англоязычной литературе, в которой вместо традиционного для русскоязычной литературы термина *метод* принято использовать термин *алгоритм*. Для того, чтобы избежать возможной неоднозначности идентификации рассматриваемых в работе объектов, мы также используем последний термин, хотя понимаем, что он с точки зрения русскоязычного читателя не совсем корректен.

Можно предложить несколько классификаций популяционных алгоритмов. В данной работе мы выделяем следующие классы таких алгоритмов:

- эволюционные алгоритмы, включая генетические алгоритмы;
- популяционные алгоритмы, вдохновленные живой природой;
- алгоритмы, вдохновленные неживой природой;
- алгоритмы, инспирированные человеческим обществом;
- прочие алгоритмы.

Эволюционные алгоритмы [1, 2], а также такие популяционные алгоритмы, вдохновленные живой природой, как алгоритмы роя частиц [3], колонии муравьев [4], роя медоносных пчел [5] и алгоритмы на основе искусственной иммунной системы [6—9] достаточно хорошо освещены в русскоязычной литературе. Данная работа представляет собой обзор большого числа других популяционных алгоритмов, почти не представленных в этой литературе.

Основные обозначения, используемые в работе, стандартизованы и почти всюду не совпадают с исходными авторскими обозначениями. Алгоритмы в разделах 2, 3 упорядочены по убыванию степени их разработанности и широте применения. Работа представляет собой некоторые переработанные главы учебного пособия, которое готовится автором к публикации.

1. Постановка задачи поисковой оптимизации и общая схема популяционных алгоритмов

Введем следующие обозначения.

$|A|$ — размерность вектора $A = (a_1, a_2, ..., a_{|A|})$;

$D \subset R^{|X|}$ — компактное множество допустимых значений вектора варьируемых параметров X (область поиска);

$E_\nu(a, b)$ — $(\nu \times 1)$-вектор независимых вещественных случайных чисел, распределенных по экспоненциальному закону

$$p(x) = \frac{1}{2b}\exp\left(-\frac{|x-a|}{b}\right),\ a,\ b > 0,\ x \in (-\infty, \infty);$$

$f(X)$ — скалярная целевая функция (критерий оптимальности) со значениями в пространстве R^1;

$f(X^{opt}) = f^{opt}$ — искомое оптимальное (минимальное или максимальное) значение целевой функции $f(X)$;

$G(X) = (g_1(X), g_2(X), ..., g_{|G|}(X))$ — $(|G| \times 1)$ — ограничивающая вектор-функция, с помощью которой задаем ограничения типа неравенств на вектор варьируемых параметров X;

$L_\nu(\lambda)$ — случайный $(\nu \times 1)$-вектор независимых вещественных случайных чисел, распределенных по закону Леви

$$p(x) = x^{-\lambda},\ \lambda \in (1; 3);$$

$N_\nu(m, \sigma)$ — $(\nu \times 1)$-вектор независимых вещественных случайных чисел, распределенных по нормальному закону с математическим ожиданием и средним квадратичным отклонением, равными m и σ соответственно;

$R^{|A|}$ — $|A|$-мерное арифметическое пространство;

$S = (s_i,\ i \in [1 : |S|])$ — текущая популяция;

$S' = (s'_i,\ i \in [1 : |S'|])$ — следующая популяция;

s^{best}, s^{worst} — лучший и худший (с точки зрения значения фитнесс-функции) агенты текущей популяции;

$\mathrm{sign}a$ — знак числа a;

$\hat{t}$ — максимальное допустимое число поколений (итераций);

t — номер текущего поколения (итерации);

$U_\nu(a; b)$ — $(\nu \times 1)$-вектор независимых вещественных случайных чисел, равномерно распределенных в интервале $[a; b]$; a, b — вещественные числа, $b > a$;

$U_\nu(n_1 : n_2)$ — $(\nu \times 1)$-вектор независимых целых случайных чисел, равномерно распределенных в интервале $[n_1 : n_2]$; n_1, n_2 — целые числа, $n_2 > n_1$;

$u_{\mathrm{sign}}^{\pm 1}$ — случайная величина с равной вероятностью принимающая значения $(-1; 1)$;

$X(t) = X = (x_1, x_2, ..., x_{|X|})$ — текущий ($|X| \times 1$)-вектор варьируемых параметров;

$X(t+1) = X' = (x_1', x_2', ..., x_{|X|}')$ — ($|X| \times 1$)-вектор варьируемых параметров на следующей итерации;

X^{opt} — искомый вектор оптимальных значений компонентов вектора варьируемых параметров X;

X^{best}, X^{worst} — текущие положения лучшего и худшего агентов популяции;

X_i^{best}, X_i^{worst} — лучшее и худшее положения агента s_i, $i \in [1 : |S|]$ на всех итерациях с начальной по текущую включительно;

$\varphi(X) = \varphi(s)$ — фитнесс-функции (функция приспособленности, приспособленность) агента s;

$\varphi^{best} = \varphi(X^{best})$, $\varphi^{worst} = \varphi(X^{worst})$ — лучшее и худшее значения фитнесс-функции;

$\varphi_i^{best} = \varphi(X_i^{best})$, $\varphi_i^{worst} = \varphi(X_i^{worst})$ — значения фитнесс-функции, соответствующие лучшему и худшему положениям агента s_i, $i \in [1 : |S|]$;

$\chi(x) = \begin{cases} 1, & x \geqslant 0, \\ 0, & x < 0 \end{cases}$ — функция Хевисайда;

$\|\bullet\|$ — некоторая векторная норма;

$\|\bullet\|_E$ — евклидова векторная норма;

$\|A_1, A_2\|_E = \sqrt{\sum_{i=1}^{|A|} (a_{1,i} - a_{2,i})^2}$ — евклидово расстояние (метрика) между целочисленными или вещественными векторами A_1, A_2, размерность которых равна $|A|$;

$\|A_1, A_2\|_M = \sum_{i=1}^{|A|} |a_{1,i} - a_{2,i}|$ — манхеттоновское расстояние между целочисленными или вещественными векторами A_1, A_2;

$A \otimes B = (a_1 b_1 \ a_2 b_2 \ ... \ a_{|A|} b_{|B|})$ — прямое произведение векторов A, B, размерность которых одинакова, т. е. $|A| = |B|$;

$\Pi = \{X | X^- \leqslant X \leqslant X^+\} = \{X | x_i^- \leqslant x_i \leqslant x_i^+, i \in [1 : |X|]\}$ — параллелепипед допустимых значений вектора варьируемых параметров.

Рассматриваем детерминированную задачу минимизации (максимизации)

$$\min_{X \in D \subset R^{|X|}} \left(\max_{X \in D \subset R^{|X|}} \right) f(X) = f(X^{opt}) = f^{opt}, \quad (1.1)$$

где

$$D = \{X | G(X) \geqslant 0\}. \quad (1.2)$$

Полагаем, что если целевая функция $f(X)$ подлежит минимизации (максимизации), то соответствующая фитнесс-функция $\varphi(X)$ также должна быть минимизирована (максимизирована). В качестве общего названия членов популяции используем термин "агент" (*agent*).

Общая схема популяционных алгоритмов включает в себя следующие этапы.

1. *Инициализация популяции.* В области поиска тем или иным образом создаем некоторое число начальных приближений к искомому решению задачи — инициализируем популяцию агентов.

2. *Миграция агентов популяции.* С помощью некоторого набора миграционных операторов, специфических для каждого из популяционных алгоритмов, перемещаем агентов в области поиска таким образом, чтобы, в конечном счете, приблизиться к искомому экстремуму целевой функции.

3. *Завершение поиска.* Проверяем выполнение условия окончания итераций и, если оно выполнено, завершаем вычисления, принимая лучшее из найденных положений агентов популяции в качестве приближенного решения задачи. Если указанные условия не выполнены, возвращаемся к выполнению этапа 2.

При инициализации популяции могут быть использованы детерминированные и случайные методы. Обычно агентов начальной популяции распределяют случайным образом равномерно по всей области поиска. Далее этап инициализации популяции обсуждаем только в том случае, если он имеет существенные особенности по сравнению с указанным распределением.

В качестве условия окончания поиска используют, как правило, условие достижения заданного числа итераций (*поколений*). Часто используют также условие стагнации (*stagnation*) алгоритма, когда лучшее достигнутое значение оптимизируемой функции не изменяется в течение заданного числа поколений. Могут быть использованы и другие условия, например, условие исчерпания процессорного времени, отпущенного на решение задачи. Далее, аналогично этапу инициализации популяции, обсуждаем условия окончания итераций только в том случае, если эти условия имеют существенные особенности.

Важнейшим понятием популяционных алгоритмов является понятие фитнесс-функции (*fitness-function*). Часто эту функцию называют функцией пригодности, функцией полезности, функцией приспособленности и т. д. Важность этой функции обусловлена тем обстоятельством, что с ее помощью оценивают "качество" агентов популяции. С самой общей точки зрения, в процессе миграции агенты движутся таким образом, чтобы приблизиться к глобальному минимуму (максимуму) фитнесс-функции.

Одной из основных проблем конструирования популяционных алгоритмов является проблема обеспечения баланса между интенсивностью поиска (скоростью сходимости алгоритма) и широтой поиска (диверсификацией поиска). Интенсификация (*intensification*) поиска требует быстрой сходимости алгоритма, что означает быстрое уменьшение разнообразия популяции. Напротив, диверсификация (*diversification*) поиска призвана обеспечить более широкий обзор пространства поиска и более высокую вероятность локализации глобального экстремума задачи. Диверсификация

требует сохранения разнообразия популяции в течение как можно большего числа поколений популяционного алгоритма. Наиболее развитыми механизмами решения проблемы обеспечения баланса между интенсивностью и широтой поиска являются механизмы адаптации (*adapting*) и самоадаптации (*self-adapting*) популяционных алгоритмов.

2. Популяционные алгоритмы, вдохновленные живой природой

В пунктах 2.1—2.5 рассматриваем бактериальные, светлячковые, сорняковые, кукушкины и обезьяньи алгоритмы соответственно. Пункт 2.6 содержит обзор некоторых других алгоритмов.

2.1. Бактериальная оптимизация

К настоящему времени разработано несколько алгоритмов оптимизации, объединяемых общих названием *бактериальная оптимизация* (*bacterial optimization*). Принято считать, что метод бактериальной оптимизации предложен Пассино (*K. M. Passino*) в 2002 г.

2.1.1. Бионические предпосылки [10]. Подвижные бактерии, такие как кишечная палочка или сальмонелла, продвигают себя с помощью вращающихся жгутиков. Чтобы двигаться вперед, все жгутики вращаются в одном направлении. При этом бактерия совершает движение, которое называют *плавание* (*swims*). При вращении жгутиков в разные стороны бактерия разворачивается — совершает движение *кувырок* (*tumble*).

Поведение бактерий обусловлено механизмом, который называется *бактериальным хемотáксисом* (*bacterial chemotaxis*) и представляет собой двигательную реакцию микроорганизмов на химический раздражитель. Данный механизм позволяет бактерии двигаться по направлениям к аттрактантам (чаще всего, питательным веществам) и от репеллентов (потенциально вредных для бактерии веществ).

Если выбранное бактерией направление движения соответствует увеличению концентрации аттрактанта (снижению концентрации репеллента), то время до следующего кувырка увеличивается. В силу малого размера бактерии сильное влияние на ее перемещения оказывает броуновское движение. В результате бактерия только в среднем движется в направлениях к полезным веществам и от вредных веществ.

В контексте задачи поисковой оптимизации бактериальный хемотаксис можно интерпретировать как механизм оптимизации использования бактерией известных пищевых ресурсов и поиска новых, потенциально более ценных областей.

2.1.2. Канонический алгоритм бактериальной оптимизации. Рассмотрим задачу многомерной глобальной безусловной максимизации.

Канонический *алгоритм бактериальной оптимизации* (*Bacterial Foraging Optimization, BFO*) основан на использовании трех следующих основных механизмов: хемотаксис, репродукция, ликвидация и рассеивание [10].

Пусть $X_{i,\,r,\,l}$ — $(|X| \times 1)$-вектор текущего положения бактерии $s_i \in S$ на итерации t (на t-м шаге хемотаксиса), r-м шаге репродукции и l-м шаге ликвидации и рассеивания. Здесь $i \in [1 : |S|]$, $t \in [1 : \hat{t}\,]$, $r \in [1 : \hat{t}^r]$, $l \in [1 : \hat{t}^l]$, где $|S|$ — четное число агентов в колонии бактерий S; $\hat{t}$, $\hat{t}^r$, $\hat{t}^l$ — общие числа итераций (шагов хемотаксиса), шагов репродукции, а также шагов ликвидации и рассеивания. Соответствующее значение фитнесс-функции обозначим $\varphi_{i,\,r,\,l}$.

Хемотаксис. Процедура хемотаксиса реализует в алгоритме *BFO* локальную оптимизацию Следующее положение $X'_{i,\,r,\,l}$ бактерии s_i определяет формула

$$X'_{i,\,r,\,l} = X_{i,\,r,\,l} + \lambda_i \frac{V_i}{\|V_i\|_E}, \qquad (2.1)$$

где V_i — текущий направляющий $(|X| \times 1)$-вектор шага хемотаксиса бактерии s_i; λ_i — текущее значение этого шага. При плавании бактерии на следующей итерации вектор V_i остается неизменным, т. е. имеет место равенство $V'_i = V_i$. При кувырке бактерии вектор V'_i представляет собой случайный вектор, компоненты которого имеют значения в интервале $[-1; 1]$. Другими словами, при кувырке имеет место равенство $V'_i = U_{|X|}(-1; 1)$. Плавание каждой из бактерий продолжается до тех пор, пока значения фитнесс-функции увеличиваются.

Значение шага хемотаксиса в формуле (2.1) может меняться в процессе поиска, уменьшаясь по некоторому закону с ростом числа итераций t.

Репродукция (*reproduction*). Механизм репродукции имеет своей целью ускорение сходимости алгоритма (за счет сужения области поиска). Назовем текущим *состоянием здоровья* (*health status*) h_i бактерии s_i сумму значений фитнесс-функции во всех точках ее траектории от первой до текущей итерации t:

$$h_i = \sum_{\tau = 1}^{t} \varphi_{i,\,r,\,l}(\tau).$$

Вычислим значения h_i, $i \in [1 : |S|]$, отсортируем все бактерии в порядке убывания состояний их здоровья и представим результат сортировки в виде линейного списка. Механизм репродукции состоит в том, что на $(r + 1)$-м шаге репродукции вторая половина агентов (наиболее слабых) исключается из указанного списка (погибает), а каждый из агентов первой (выжившей) половины списка расщепляется на два одинаковых

агента с одинаковыми координатами, равными координатам расщепленного агента.

Пусть, например, s_j, $j \in [1 : |S|]$, — один из выживших агентов, положение которого определяет вектор $X_{i,\,r,\,l}$. После репродукции этого агента получаем агентов s_j, s_k, $k = |S|/2 + j$, положения которых равны $X'_{j,\,(r+1),\,l} = X_{i,\,r,\,l}$, $X'_{k,\,(r+1),\,l} = X_{j,\,r,\,l}$.

В результате выполнения процедуры репродукции результирующее число бактерий в популяции остается низменным и равным $|S|$.

Ликвидация и рассеивание (*elimination and dispersal*). Рассмотренных процедур хемотаксиса и репродукции в общем случае недостаточно для отыскания глобального максимума многоэкстремальной целевой функции, поскольку эти процедуры не позволяют бактериям покидать найденные ими локальные максимумы этой функции. Процедура ликвидации и рассеивания призвана преодолеть этот недостаток.

Механизм ликвидации и рассеивания включается после выполнения определенного числа процедур репродукции и состоит в следующем. С заданной вероятностью ξ_e случайным образом выбираем n бактерий s_{i_1}, s_{i_2}, ..., s_{i_n} и уничтожаем их. Вместо каждой из уничтоженных бактерий в случайно выбранной точке пространства поиска создаем нового агента с тем же номером. Подчеркнем, что в результате выполнения операции ликвидации и рассеивания число бактерий в колонии также остается постоянным и равным $|S|$.

Более строго процедуру ликвидации и рассеивания определяет следующая последовательность шагов:

- полагаем $j = 1$;
- генерируем натуральное случайное число $i_j = U_1(1 : |S|)$ и вещественное случайное число $u_j = U_1(0; 1)$;
- если $u_j > \xi_e$, то новые координаты бактерии s_j определяем по формуле

$$x_{i_j,\,r,\,l,\,k} = U_1(x_k^-;\, x_k^+),\ k \in [1 : |X|], \qquad (2.2)$$

где x_k^-, x_k^+ — константы, определяющие диапазон возможных начальных координат бактерий;

- полагаем $j = j + 1$ и продолжаем итерационный процесс до тех пор, пока не будет уничтожено n бактерий.

Определение векторов начальных положений бактерий $X_{i,\,0,\,0}(0)$, $i \in [1 : |S|]$ выполняется по формуле, аналогичной формуле (2.2).

2.1.3. Кооперативная бактериальная оптимизация предполагает разделение процедуры оптимизации колонией бактерий на несколько этапов, каждый из которых занимает некоторую часть общего числа поколений и характеризуется различными значениями параметра λ. Известны два варианта *кооперативного бактериального алгоритма оптимизации* (*Cooperative Bacterial Foraging Optimization*, *CBFO*) — алгоритмы *CBFO-S* и *CBFO-H* [11]. Рассмотрим вначале первый из указанных алгоритмов.

Последовательный алгоритм бактериальной оптимизации (*CBFO-S*) представляет собой пример гибридизации по схеме препроцессор/постпроцессор [12]. Алгоритм использует несколько алгоритмов *BFO*, отличающихся значениями параметра λ и выполняющихся последовательно друг за другом. Выход предыдущего алгоритма *BFO* (лучшие позиции, найденные каждой из бактерий) поступают на вход следующего алгоритма *BFO*. На начальном этапе используем алгоритм *BFO* с большим значением параметра λ, обеспечивающим поиск по всему пространству поиска. В процессе поиска каждая из бактерий сохраняет в своей памяти координаты посещенных точек, а также соответствующие значения фитнесс-функции. Позиция с наибольшим значением фитнесс-функции объявляется *многообещающим участком* (*promising regions*).

При переходе на следующий этап (к выполнению следующего алгоритма *BFO*) текущее значение параметра λ по некоторому правилу уменьшаем и начинаем поиск из точек, принадлежащих найденным на предыдущем этапе многообещающим позициям. Поиск с данным значением λ продолжаем до выполнения условия перехода к следующему этапу. Затем повторяем процесс с еще меньшим значением параметра λ и так далее до выполнения условия окончания итераций.

Число этапов алгоритма $n_p < \hat{t}$ является свободным параметром. Числа шагов хемотаксиса в пределах каждого из этапов обычно полагают одинаковыми. Новое значение шага λ определяют, например, по правилу

$$\lambda' = \nu\lambda,$$

где $\nu \in (0; 1)$ — свободный коэффициент уменьшения шага алгоритма.

Развитием алгоритма *CBFO-S* можно считать *адаптивный алгоритм бактериальной оптимизации* (*Adaptive Bacterial Foraging Optimization, ABFO*), в котором шаг λ меняется, вообще говоря, на каждой итерации по правилу

$$\lambda' = \lambda(t + 1) = \sum_{\tau = 0}^{m} b_\tau \lambda(t - \tau),$$

где m — число предыдущих учитываемых шагов, b_τ — весовые коэффициенты, назначаемые лицом, принимающим решения.

Гибридный алгоритм бактериальной оптимизации (*CBFO-H*) [13] являет собой пример низкоуровневой гибридизации вложением [14, 15]. Алгоритм является многопопуляционным и выполняется в два этапа. На первом этапе используем алгоритм *BFO* с большим значением параметра λ, который после заданного числа итераций обнаруживает аналогично алгоритму

BFO-S многообещающие участки и передает их на следующий этап.

Основной особенностью второго этапа алгоритма *BFO-H* является разложение пространства поиска $R^{|X|}$ на $\frac{|X|}{2}$ подпространств, каждое из которых имеет размерность, равную двум. Поиск в каждом из указанных двумерных подпространств выполняет своя колония бактерий, совокупность которых образует мегаколонию $\mathbf{S} = \{S_i,\ i \in [1 : |X|/2]\}$, где $|X|/2 = |\mathbf{S}|$ — число колоний в мегаколонии.

2.1.4. Алгоритм, использующий эффект роения бактерий. Эффект роения бактерий наблюдается для нескольких разновидностей бактерий, включая бактерию кишечной палочки и бактерию сальмонеллы (лат. *Salmonella*), в полутвердой питательной среде. Эффект обусловлен локальным уменьшением концентрации аттрактанта вследствие его потребления бактериями. В результате возникает градиент этой концентрации, и в хаотическом перемещении бактерий появляется составляющая, направленная по градиенту. Кроме того, бактерии продолжают размножаться. Можно считать, что данный эффект объясняется наличием неявного канала связи между бактериями. В рассматриваемом алгоритме этот канал формализуют следующим образом.

Введем в рассмотрение величины d_{att}, w_{att}, определяющие размеры области наличия аттрактанта, а также аналогичные величины d_{rep}, w_{rep}, характеризующие размеры области репеллентов (имеются в виду двумерная модель популяции бактерий). Указанную выше связь между бактериями задаем функцией [16]

$$\tilde{\varphi}(X) = \sum_{i=1}^{|S|} [-d_{att}\exp(-w_{att}\|X - X_i\|_E^2) + d_{rep}\exp(-w_{rep}\|X - X_i\|_E^2)]. \quad (2.3)$$

Легко видеть, что алгебраическое значение первого слагаемого в формуле (2.3) возрастает с увеличением евклидова расстояния от точки X до точки X_i. Наоборот, второе слагаемое в этой формуле убывает с ростом указанного расстояния. Таким образом, первая сумма в формуле (2.3) формализует "притяжение" бактерий областью, в которой имеются запасы аттрактанта (а значит, и сосредоточены агенты популяции). Аналогично, вторая сумма в этой формуле формализует "отталкивание" бактерий этой областью. Отметим, что значение функции $\tilde{\varphi}(X)$ не зависит от значения целевой функции в точке X.

В качестве фитнесс-функции используем функцию

$$\varphi(X) := \varphi(X) + \tilde{\varphi}(X). \quad (2.4)$$

Варьируя значения параметров d_{att}, w_{att}, d_{rep}, w_{rep}, можно управлять поведением колонии бактерий. Так, уменьшение параметра d_{att} и увеличение параметра w_{rep}, а также увеличение d_{rep} и уменьшение w_{rep} увеличивает модифицированную фитнесс-функцию (2.4) и диверсифицирует поиск. Противоположные изменения указанных величин локализуют популяцию в небольших областях, обеспечивая высокую точность поиска.

2.1.5. Гибридные бактериальные алгоритмы. Известно большое число алгоритмов оптимизации, построенных на основе гибридизации бактериального алгоритма с другими популяционными алгоритмами. Рассмотрим в качестве примеров низкоуровневую гибридизацию [14, 15] бактериального алгоритма с генетическим алгоритмом (алгоритм *BFO-GA*) и алгоритмом роя частиц (алгоритм *BFO-PSO*).

Гибридный алгоритм *BFO-GA* [13] представляет собой расширенный генетическими операторами отбора, скрещивания и мутации алгоритм, использующий роение бактерий (п. 2.1.4). В качестве оператора отбора алгоритм *BFO-GA* использует алгоритм на основе метода рулетки. Скрещивание реализовано с помощью оператора арифметического кроссовера, а мутация — с помощью неравномерного мутатора Михалевича [1].

Указанные модифицирующие операторы выполняются после завершения процедур хемотаксиса и репродукции бактериального алгоритма перед процедурами ликвидации и рассеивания этого алгоритма (п. 2.1.2).

Основной целью гибридизации в алгоритме *BFO-PSO* было расширение возможностей агентов бактериального канонического алгоритма средствами обмена информацией между ними, позаимствованными в алгоритме *PSO* [17]. Таким образом, алгоритм *BFO-PSO* моделирует процессы хемотаксиса, репродукции, ликвидации и рассеивания колонии бактерий, а также процесс роения агентов.

2.2. Алгоритмы, вдохновленные роем светлячков

В мире насчитывается около двух тысяч видов светлячков, большинство из которых обладают способностью светиться, производя короткие и ритмичные вспышки. Считается, что основной функцией таких вспышек является привлечение особей противоположного пола и потенциальных жертв. Кроме того, сигнальные вспышки могут служить защитным механизмом предупреждения потенциальных хищников о том, что светлячок горек на вкус.

Известны два варианта популяционных алгоритмов оптимизации, инспирированных поведением светлячков — *алгоритм светлячков* (*firefly algorithm*) и *алгоритм оптимизации роем светлячков* (*Glowworm Swarm Optimization*, *GSO*). Основное различие между *firefly*- и *glowworm*-светлячками состоит в том, что вторые являются бескрылыми.

2.2.1 Алгоритм светлячков [18]. Алгоритм светлячков предложен в 2007 г. Янгом (*X.-Sh. Yang*). Алгоритм ис-

пользует следующую модель поведения светлячков: все светлячки могут привлекать друг друга независимо от своего пола; привлекательность светлячка для других особей пропорциональна его яркости; менее привлекательные светлячки перемещаются в направлении более привлекательного светлячка; яркость излучения данного светлячка, видимая другим светлячком, уменьшается с увеличением расстояния между светлячками; если светлячок не видит возле себя светлячка более яркого, чем он сам, то он перемещается случайным образом.

Яркость излучения светлячка $s_i \in S$, $i \in [1 : |S|]$, принимаем равной значению фитнесс-функции в его текущем положении.

Привлекательность светлячка s_i для светлячка s_j полагаем равной

$$\beta_{i,j} = \beta_0 \exp(-\gamma r_{i,j}^2),\ i, j \in [1 : |S|],\ i \neq j, \qquad (2.5)$$

где $r_{i,j}$ — расстояние между светлячками s_i, s_j; β_0 — взаимная привлекательность (*attractiveness*) светлячков при нулевом расстоянии между ними; γ — вещественная величина, имеющая смысл коэффициента поглощения света среды (*light absorption coefficient*).

Для ускорения и упрощения вычислений экспоненциальную функцию в выражении (2.5) можно заменить функцией $\frac{1}{1 + r_{i,j}^2}$. При этом указанное выражение приобретает вид

$$\beta_{i,j} = \frac{\beta_0}{1 + \gamma r_{i,j}^2},\ i, j \in [1 : |S|],\ i \neq j. \qquad (2.6)$$

Формулы (2.5), (2.6) определяют характерное расстояние $r_c = 1/\sqrt{\gamma}$, на котором привлекательность изменяется от β_0 до $\beta_0 \mathbf{e}^{-1}$ для случая (2.5) и от β_0 до $\beta_0/2$ для случая (2.6).

В качестве функции $\beta(r)$ могут быть использованы и другие монотонно убывающие функции, например, функция вида

$$\beta(r) = \beta_0 \mathbf{e}^{-\gamma r^n},\ n \geqslant 1.$$

Движение светлячка s_j, который притягивается более привлекательным светлячком s_i, определяет формула

$$X_j = X_i + \beta(r_{i,j})(X_j - X_i) + \alpha U_{|X|}(-1; 1),\quad i, j \in [1 : |S|],\ i \neq j, \qquad (2.7)$$

где α — свободный параметр рандомизации. Для обеспечения приемлемого баланса между диверсификацией и интенсификацией поиска значение параметра рандомизации рекомендуют уменьшать с ростом номера поколения, например, по формуле

$$\alpha = \alpha_\infty + (\alpha_0 - \alpha_\infty)\mathbf{e}^{-t},$$

где α_0 — начальное значение параметра рандомизации, α_∞ — его окончательное значение.

Для улучшения сходимости алгоритма в формуле (2.7) может быть использован еще один член вида

$$\lambda U(-1; 1) \otimes (X_j - X^{best}),$$

где λ — параметр, подобный параметру α.

Схема алгоритма светлячков для задачи глобальной безусловной минимизации фитнесс-функции имеет следующий вид:

1) инициализируем начальную популяцию светлячков $S = (s_i;\ i \in [1 : |S|])$ и вычисляем значения фитнесс-функции в начальных точках;

2) если $\varphi(X_j) < \varphi(X_i)$, то по формуле вида (2.7) перемещаем светлячка s_j в направлении светлячка s_i; $i, j \in [1 : |S|]$, $i \neq j$;

3) вычисляем значения фитнесс-функции в полученных точках X_i', $i \in [1 : |S|]$;

4) если условие окончания итераций не выполнено, то переходим к шагу 2.

В большинстве случаев можно использовать следующие значения основных свободных параметров алгоритма: $\beta_0 = 1$; $\alpha \in [0; 1]$; $\gamma = 1$.

2.2.2 Алгоритм оптимизации роем светлячков. Алгоритм *GSO* предложили в 2005 г. Кришнананд (*K.N. Krishnanand*) и Гхоус (*D. Ghose*) [19]. Состояние светлячка s_i, $i \in [1 : |S|]$, определяют следующие переменные: X_i — его текущее положение в пространстве поиска; l_i — уровень светимости (*luciferin level*); r_i — радиус окрестности (*neighborhood range*). Основным содержанием каждой итерации является обновление значений указанных переменных.

Уровень светимости светлячка s_i обновляем по формуле

$$l_i' = (1 - \rho)l_i + \gamma\varphi(X_i),\ i \in [1 : |S|],$$

где ρ, γ — положительные свободные параметры алгоритма, моделирующие распад флуоресцирующего вещества и привлекательность светлячка. Отличные от нуля значения параметра ρ обеспечивают алгоритм памятью. Параметр γ определяет относительные веса текущей светимости агента и значения его фитнесс-функции.

Светлячок s_j считается соседом светлячка s_i; $i, j \in [1 : |S|]$, $i \neq j$ при выполнении двух следующих условий: евклидово расстояние между этими светлячками не превышает текущий радиус окрестности r_i; текущий уровень светимости светлячка s_j превышает этот же уровень светлячка s_i, т. е. $l_j > l_i$. Если светлячок имеет несколько соседей, то случайным образом выбираем одного из них с вероятностью, пропорциональной уровням их светимости (правило рулетки).

Положим, что по рассмотренной схеме светлячком s_i выбран светлячок s_j. Тогда новое положение светлячка s_i определяет формула

$$X_i' = X_i + \lambda \frac{X_j - X_i}{\|X_j - X_i\|_E}, \; i, j \in [1 : |S|], \; i \neq j,$$

где λ — постоянное значение шага (свободный параметр алгоритма).

Новый радиус окрестности светлячка s_i определяем в соответствии с выражением

$$r_i' = \min(r_{\min}, \max(0, (r_i + \varepsilon(n - |N_i|)))), \; i \in [1 : |S|], \quad (2.8)$$

где N_i — текущее множество соседей светлячка s_i, $r_{\min}$ — минимально допустимый радиус окрестности, n — желательное число соседей, ε — положительная константа. Последние три величины являются свободными параметрами алгоритма.

Рассмотрим некоторые модификации представленного канонического алгоритма *GSO*.

Для диверсификации поиска в случае, когда число соседей $|N_i|$ светлячка s_i, $i \in [1 : |S|]$, в окрестности N_i менее желательного числа n, радиус окрестности $r_{\min}$ может быть по тому или иному правилу расширен (в простейшем случае, положим, на 5 %).

В той же ситуации возможно иное решение — случайное перемещение светлячка в пределах куба с центром в точке X_i и длиной ребра, равной шагу λ:

$$X_{i,j}' = X_{i,j} + \lambda(0{,}5 - U_1(0; 1)), \; j \in [1 : |X|].$$

Если значение фитнесс функции в новой точке X_i' лучше ее значения в точке X_i, то оставляем светлячка в точке X_i', в противном случае — возвращаем в прежнюю точку X_i.

Новый радиус r_i' окрестности светлячка s_i может вычисляться не по формуле (2.8), а по формуле вида

$$r_i' = \frac{r_{\max}}{1 + \varepsilon_r d_i},$$

где $r_{\max}$ — максимально допустимое значение этой величины, $\varepsilon_r > 0$ — заданная константа, d_i — средняя текущая плотность агентов в окрестности N_i:

$$d_i = \frac{|N_i|}{\pi (r_i)^2}.$$

Величины $r_{\max}$, ε_r представляют собой свободные параметры алгоритма.

Наиболее сложной является модификация алгоритма *GSO*, использующая специальный механизм организации агентов в группы. В каждой из групп в этом случае выделяют одного из агентов в качестве ее владельца (*master*), который определяет миграцию всех подчиненных (*slave*) агентов группы. В исходном состоянии подчиненные агенты равномерно распределены случайным образом в гиперсфере радиуса $r_d \approx 0{,}1 r_s$ с центром, совпадающим с начальным положением владельца группы. В процессе поиска величина r_d может варьироваться в диапазоне $[0{,}33\lambda;\; 0{,}1 r_s]$, где r_s — свободный параметр алгоритма.

2.3. Сорняковый алгоритм

Алгоритм сорняковой оптимизации (*Invasive Weed Optimization*, *IWO*) вдохновлен таким общераспространенным явлением, как колонизация сельскохозяйственных угодий сорняками. Алгоритм предложили в 2006 г. Мехрабиан (*A. R. Mehrabian*) и Лукас (*C. Lucas*) [20].

2.3.1. Биологические основы. Говоря простым языком, сорняк — это любое растение, растущее там, где оно нежелательно. В общем случае, любое растение может быть классифицировано как сорняк. Однако обычно термин используют по отношению к тем растениям, чьи экспансионистские свойства представляют серьезную угрозу для культивируемых растений.

Основным механизмом, определяющим динамику сообщества любых растений, является естественный отбор, из которого выделяют два крайних типа: *r*-отбор и *K*-отбор. Реальные стратегии отбора лежат между этими предельными типами.

Можно сказать, что девизом *r*-отбора являются слова "живи быстро, размножайся быстро, умирай молодым" (*live fast, reproduce quick, die young*). Данный тип отбора необходим для успеха в нестабильной, непредсказуемой окружающей среде. При *r*-отборе предпочтительны такие качества, как высокая плодовитость, маленький размер семян и приспособленность к рассеиванию их на большое расстояние.

K-отбор использует принцип "живи медленно, размножайся медленно, умирай в старости" (*live slow, reproduce slow, die old*). Этот тип отбора необходим для успеха в стабильной, предсказуемой окружающей среде, когда вероятно тяжелое соперничество за ограниченные ресурсы между конкурентно способными индивидуумами. Ситуация имеет место, если размер популяции в ареале обитания близок к максимуму, который он способен вместить. При *K*-отборе предпочтительны такие качества индивидов, как большой размер семян, длинная жизнь, небольшое потомство, за которым требуется интенсивный уход.

2.3.2. Схема алгоритма. Рассматриваем задачу глобальной безусловной минимизации. В алгоритме *IWO* модель поведения сорняков при колонизации учитывает следующие базовые свойства этого процесса:

1) распределение конечного числа семян по всей области поиска (инициализация популяции);

2) производство выросшими растениями семян в зависимости от приспособленности растений (воспроизводство);

3) размещение произведенных семян в случайном порядке по области поиска (пространственное распределение);

4) повторение шагов 2, 3 до тех пор, пока не достигнут заданный максимум числа растений;

5) отбор растений с более высокой приспособленностью, их воспроизводство и пространственное распределение (конкурентное исключение);

6) повторение шага 5 до выполнения условия окончания процесса.

Воспроизводство. В оригинальном алгоритме *IWO* число семян n_i^s, произведенных сорняком s_i; $i \in [1 : |S|]$, линейно зависит от его текущей приспособленности $\varphi_i = \varphi(X_i)$ и определяется формулой

$$n_i^s = \frac{n_{\max}^s - n_{\min}^s}{\varphi^{best} - \varphi^{worst}} \varphi_i + \frac{\varphi^{best} n_{\min}^s - \varphi^{worst} n_{\max}^s}{\varphi^{best} - \varphi^{worst}},$$

$n_{\min}^s$, $n_{\max}^s$ — заданные константы, представляющие собой минимальное и максимальное числа семян, которые могут быть произведены каждым из сорняком на текущей итерации.

Пространственное распределение. Произведенные сорняком s_i, $i \in [1 : |S|]$, семена распределяем в окрестности родительского растения в соответствии с нормальным законом распределения:

$$X_{i,j} = X_i + N_{|X|}(0, \sigma),\ i \in [1 : |S|],\ j \in [1 : n_i^s].$$

Стандартное отклонение σ в последнем выражении зависит от текущего номера поколения t, уменьшаясь с ростом этого номера по формуле

$$\sigma = \sigma(t) = \left(\frac{\hat{t} - t}{\hat{t}}\right)^m (\sigma_b - \sigma_e) + \sigma_e, \qquad (2.9)$$

где σ_b, σ_e — начальное и конечное значения стандартного отклонения σ, $m > 1$ — свободный параметр, определяющий характер функции $\sigma(t)$ (*параметр модуляции*).

Формула (2.9) обеспечивает уменьшение вероятности попадания семян вдали от родительского растения с ростом номера итераций, уменьшая тем самым диверсификационные и повышая интенсификационные свойства алгоритма. Можно сказать, что данная схема изменения стандартного отклонения σ реализует механизм перехода от *r*-отбора к *K*-отбору в процессе эволюции популяции.

Конкурентное исключение. Обозначим S_i^s популяцию сорняков, являющихся потомками растения s_i. Тогда до достижения всей популяцией сорняков своего максимального размера, равного $|S|$, новую популяцию S' формируем путем объединения текущей популяции S со всеми популяциями S_i^s:

$$S' = S \cup \left(\bigcup_{i=1}^{|S|} S_i^s \right).$$

Конкурентное исключение начинает функционировать после достижения популяцией размера $|S|$ и заключается в уничтожении сорняков с меньшей приспособленностью до достижения популяцией размера $|S|$. Таким образом, растения и их потомки оцениваются вместе, и тем, которые обладают лучшей приспособленностью, позволяют размножаться. Данный механизм позволяет растениям с меньшей приспособленностью воспроизводиться, и, если их потомки обладают хорошей приспособленностью, они могут выжить.

2.4. Кукушкин поиск

Алгоритм кукушкиного поиска (*Cuckoo Search, CS*) предложили Янг (*X.-Sh. Yang*) и Деб (*S. Deb*) в 2009 г. [21]. Алгоритм вдохновлен поведением кукушек в процессе вынужденного гнездового паразитизма.

2.4.1. Биологические предпосылки. Такие виды кукушек, как *Ani* и *Guira*, откладывают яйца в коллективные гнезда вместе с другими кукушками, хотя могут выбрасывать яйца конкурентов, чтобы увеличить вероятность вылупления их собственных птенцов. Целый ряд видов кукушек занимается гнездовым паразитизмом, подкладывая свои яйца в гнезда других птиц как своего вида, так и, часто, других видов.

Некоторые птицы могут конфликтовать с вторжением кукушек. К примеру, если хозяин гнезда обнаружит в нем яйца иного вида, то он либо выбросит эти яйца, либо просто покинет данное гнездо и соорудит на новом месте другое гнездо.

2.4.2. Схема алгоритма. В алгоритме *CS* каждое яйцо в гнезде представляет собой решение, а яйцо кукушки — новое решение. Цель заключается в использовании новых и потенциально лучших (кукушкиных) решений, чтобы заменить менее хорошие решения в гнездах. В простейшем варианте алгоритма в каждом гнезде находится по одному яйцу.

Положим, что речь идет о задаче глобальной безусловной максимизации. Алгоритм основан на следующих правилах: каждая кукушка откладывает одно яйцо за один раз в случайно выбранное гнездо; лучшие гнезда с яйцами высокого качества (высоким значением пригодности) переходят в следующее поколение; яйцо кукушки, отложенное в гнездо, может быть обнаружено хозяином с некоторой вероятностью ξ_a (0; 1) и удалено из гнезда. Схему алгоритма *CS* можно представить в следующем виде:

1) инициализируем популяцию $S = (s_i,\ i \in [1 : |S|])$ из $|S|$ хозяйских гнезд и кукушку, т. е. определяем начальные значения компонентов векторов X_i; $i \in [1 : |S|]$, и вектор начального положения кукушки X_c;

2) выполняем некоторое число случайных перемещений кукушки в пространстве поиска с помощью полетов Леви [21] и находим новое положение кукушки X_c;

3) случайным образом выбираем гнездо s_i; $i \in [1 : |S|]$, и, если $\varphi(X_c) > \varphi(X_i)$, то заменяем яйцо в этом гнезде на яйцо кукушки, т. е. полагаем $X_i = X_c$;

5) с вероятностью ξ_a удаляем из популяции некоторое число худших случайно выбранных гнезд (включая, быть может, гнездо s_i) и по правилам шага 1 строим такое же число новых гнезд;

6) если условие окончания итераций не выполнено, то переходим к шагу 2.

Полеты Леви кукушки реализуем по формуле

$$X'_c = X_c + V \otimes L_{|X|}(\lambda), \qquad (2.10)$$

где обычно полагают все компоненты вектора V одинаковыми и равными v: $V = (v_j = v, j \in [1 : |X|])$. Величина v должна быть связана с масштабами области поиска.

Известно несколько модификаций алгоритма *CS*. В каноническом алгоритме *CS* кукушка в своих полетах Леви никак не учитывает информацию о лучших найденных решениях. В *модифицированном алгоритме поиска кукушки* (*Modified Cuckoo Search*, *MCS*) компоненты вектора V вычисляют по формуле вида

$$V = U_1(0; 1)(X^{best} - X_k),\ k \in [1 : |S|],$$

где $X_k \neq X^{best}$ — случайно выбранное гнездо [22]. Так определенный вектор V обеспечивает большую вероятность полета кукушки к гнездам, имеющим высокую приспособленность.

Вместе с тем, в каноническом алгоритме *CS* вероятность ξ_a и параметры полета Леви являются фиксированными константами. В целях диверсификации поиска на ранних итерациях целесообразно использовать большие значения величин ξ_a, v. На завершающих итерациях для повышения точности локализации экстремума (интенсификации поиска) разумны меньшие значения указанных величин.

Улучшенный алгоритм поиска кукушки (*Improved Cuckoo Search, ICS*) использует динамические значения этих параметров:

$$\xi_a(t) = \xi_a^{\max} - \frac{t}{\hat{t}}(\xi_a^{\max} - \xi_a^{\min});$$

$$v(t) = v^{\max}\exp(d^t),\ d = \frac{1}{\hat{t}}\ln\left(\frac{v^{\min}}{v^{\max}}\right).$$

Здесь $\xi_a^{\min}$, $\xi_a^{\max}$, $v^{\min}$, $v^{\max}$ — заданные константы.

2.5. Алгоритмы, вдохновленные поведением обезьян

Известны два существенно различных алгоритма, вдохновленных некоторыми аспектами поведения обезьян в процессе поиска ими пищи — алгоритм обезьяньего поиска и обезьяний алгоритм.

2.5.1. Обезьяний поиск. *Алгоритм обезьяньего поиска* (*Monkey Search*, *MS*) предложили Мучерино (*A. Mucherino*) и Шереф (*O. Seref*) в 2008 г. [23]. Алгоритм вдохновлен поведением обезьяны, лазающей по дереву в поисках пищи. Обезьяне ставится в соответствие агент, который строит деревья решений для поиска экстремума в задаче глобальной максимизации.

В алгоритме *MS* максимальное количество пищи представляет собой желаемое решение, а ветви дерева представляют собой варианты выбора между соседними допустимыми решениями в рассматриваемой задаче оптимизации. Этот выбор может быть как полностью случайным, так и основанным на известных алгоритмах решения задачи глобальной оптимизации. Алгоритм использует бинарные деревья поиска, т. е. от каждой данной ветки (кроме ветвей, образующих вершину дерева) отходят две другие ветви с решениями, располагающимися на их концах.

Канонический алгоритм *MS* не является популяционным, поскольку предполагает поиск с помощью только одной обезьяны. Известны "многообезьяньи" модификации канонического алгоритма, что и позволяет нам включить данный алгоритм в рассмотрение.

Если в текущий момент времени обезьяна находится на конце некоторой ветви, то далее она с равной вероятностью перемещается по левой или правой исходящим ветвям. В точке пространства поиска, соответствующей концу ветви, на которой находится обезьяна, вычисляем значение фитнесс-функции. Если это решение лучше найденного ранее лучшего решения, то запоминаем его, и по рассмотренной схеме обезьяна продолжает движение вверх. Движение останавливаем при достижении обезьяной вершины дерева, определяемой максимально допустимой его высотой $l_{\max}$. Все посещенные обезьяной ветви дерева запоминаем.

Если не все пути в дереве исследованы, то всякий раз, после достижения обезьяной вершины дерева, она спускается до текущей лучшей точки и снова начинает движение вверх, возможно, проходя некоторые из уже пройденных ветвей.

Принципиальным в алгоритме *MS* является способ генерации ветвей дерева поиска, т. е. генерации решений, соседствующих с текущим положением обезьяны. Эти решения получают путем локального возмущения текущего решения, для чего могут быть использованы миграционные операторы любых популяционных алгоритмов. Например, если X, X^{best} — текущее и лучшее положения обезьяны, то два следующих ее возможных положения X'_1, X'_2 могут быть найдены путем скрещивания решений X, X^{best}. Известны модификации алгоритма *MS*, использующие для генерации ветвей алгоритмы колонии муравьев [4], поиска гармонии (п. 3.1) и т. д.

Для уменьшения вероятности преждевременной стагнации поиска может быть ограничено максимально допустимое число путей, подлежащих исследованию в данном дереве. Если это число достигнуто, то лучшее полученное решение дерева запоминаем. Затем исходя из *семени* (*seed*) — случайной точки пространства поиска — "выращиваем" новое дерево поиска.

После исследования по рассмотренной схеме заданного числа деревьев в качестве семян новых деревьев могут быть использованы случайные решения из списка лучших решений. Поиск останавливаем, когда исследовано все предопределенное число деревьев.

2.5.2. Обезьяний алгоритм. Данный алгоритм предложили Жао (*R. Zhao*) и Танг (*W. Tang*) в 2007 г. [24] Алгоритм моделирует поведение обезьян в процессе их лазания по горам в целях поиска пищи. Полагают, что обезьяны исходят из того, что чем выше гора, тем больше пищи на ее вершине. Местность, которую обследуют обезьяны, представляет собой ландшафт фитнесс-функции, так что решению соответствует самая высокая гора (рассматриваем задачу глобальной максимизации).

Из своего текущего положения каждая из обезьян движется вверх до тех пор, пока не достигнет вершины горы. Затем обезьяна делает серию локальных прыжков в случайном направлении в надежде найти более высокую гору, и движение вверх повторяется.

После выполнения некоторого числа подъемов и локальных прыжков обезьяна полагает, что в достаточной степени исследовала ландшафт в окрестности своего начального положения. Для того, чтобы обследовать новую область пространства поиска, обезьяна выполняет длинный глобальный прыжок.

Указанные выше действия повторяются заданное число раз. Решением задачи объявляется самая высокая из вершин, найденных данной популяцией обезьян.

Процесс движения вверх (*climb process*) представляет собой процесс локального поиска и может быть реализован многими способами. В оригинальном алгоритме авторы предлагают использовать алгоритм на основе процедуры стохастической аппроксимации [24].

Локальные прыжки (*watch-jump process*). Схема локальных прыжков для агента s_i, $i \in [1 : |S|]$, имеет следующий вид:

1) исходя из текущего положения агента X_i генерируем его новое возможное положение X_i' по формуле

$$x'_{i,j} = U_1((x_{i,j} - b); (x_{i,j} + b)),\ i \in [1 : |S|],\ j \in [1 : |X|],$$

т. е. полагаем, что по каждой из координат новое положение агента представляет собой случайную величину, равномерно распределенную в интервале $[(x_{i,j} - b); (x_{i,j} + b)]$. Здесь $b > 0$ — максимально возможная длина прыжка вдоль соответствующего координатного направления (свободный параметр алгоритма);

2) если точка X_i' является допустимой (принадлежит области допустимых значений D) и $\varphi(X_i') \geqslant \varphi(X_i)$, то полагаем $X_i = X_i'$ и завершаем для данного агента локальные прыжки, в противном случае заданное число раз возвращаемся к шагу 1.

Глобальные прыжки (*somersault process*) агенты выполняют из своего текущего положения в направлении текущего центра тяжести их координат по следующей схеме:

1) генерируем случайное вещественное число v, равномерно распределенное в интервале $[v_{\min}; v_{\max}]$ и имеющее смысл шага глобального прыжка:

$$v = U_1(v_{\min}; v_{\max}).$$

Здесь $v_{\min}$; $v_{\max}$ — нижняя и верхняя границы этой величины, которые, в общем случае, имеют разные знаки, так что величина v может быть как положительной, так и отрицательной. Для деверсификации поиска интервал $[v_{\min}; v_{\max}]$, очевидно, следует расширять, а для интенсификации — сужать;

2) находим новое возможное положение X_i' агента s_i по формуле

$$x'_{i,j} = x_{i,j} + v(x_j^c - x_{i,j}),\ i \in [1 : |S|],\ j \in [1 : |X|],$$

где $x_j^c = \frac{1}{|S|} \sum_{i=1}^{|S|} x_{i,j}$ — текущее положение центра тяжести агентов популяции по j-му координатному направлению;

3) если положение X_i' является допустимым, то полагаем $X_i = X_i'$ и завершаем для данного агента глобальные прыжки, иначе заданное число раз возвращаемся к шагу 1.

Отметим, что если величина v принимает положительное значение, то агент совершает прыжок в сторону центра тяжести X^c, а если отрицательное значение — в противоположную сторону. Поскольку алгоритм не гарантирует, что лучшее решение будет получено на последней итерации, каждый раз необходимо сохранять в памяти ЭВМ текущее лучшее решение.

2.6. Прочие алгоритмы

В данном пункте рассматриваем ряд менее разработанных алгоритмов — алгоритмы лягушек, летучих мышей, рыб и растущих деревьев. Известен также алгоритм оптимизации роем мух [25, 26], не вошедший в обзор по причине его недостаточной изученности.

2.6.1. Тасующий алгоритм прыгающих лягушек (*Shuffled Frog-Leaping, SFL*) предложил Юсуф (*M. Eusuff*) с соавторами в 2003 г. [27]. По сути, алгоритм представляет собой гибридизацию меметического алгоритма (п. 3.7) и алгоритма роя частиц. Алгоритм вдохновлен поведением группы лягушек в процессе поиска пищи.

Рассматриваем задачу глобальной условной максимизации. Алгоритм *SFL* включает в себя следующие основные шаги:

1) инициализируем популяцию $S = (s_i, i \in (1 : |S|))$;

2) оцениваем пригодность агентов популяции $\varphi(X_i) = \varphi_i$, $i \in (1 : |S|)$ и определяем на этой основе глобально лучшего агента s^{best};

3) разделяем агентов на $|S^p|$ наборов $S_j^p = (s_{j,k}, k \in [1 : n])$, $j \in [1 : |S^p|]$, названных авторами алгорит-

ма мемеплексами (*memeplexes*), каждый из которых содержит по $n = \frac{|S|}{|S^p|}$ агентов, так что $\bigcap_{i=1}^{|S^p|} S_j^p = S$. Разделение выполняем по следующему правилу соседства: агента s_1 относим к мемеплексу S_1^p, агента s_2 — к мемеплексу S_2^p и т. д. до мемеплекса S_n^p, к которому относим агента s_n; агента s_{n+1} относим к мемеплексу S_1^p, агента s_{n+1} — к мемеплексу S_2^p и так далее до последнего агента $s_{|S|}$, который должен быть отнесен к мемеплексу S_n^p. Полагаем, что величины $|S|, |S^p|$ кратны;

4) для каждого из мемеплексов выполняем *процедуру меметической эволюции* (*memetic evolution procedure*), которая состоит в следующем:

4.1) в каждом из мемеплексов S_j^p находим лучшего $s_{j,*}^{best}$ и худшего $s_{j,*}^{worst}$ агентов;

4.2) пытаемся улучшить положение худшего агента путем случайного перемещения его в направлении к или от лучшего агента по формуле

$$X_{j,*}^{worst}(t+1) = X_{j,*}^{worst} + U_1(-0{,}5;\ 0{,}5)v_{\max}\frac{X_{j,*}^{best} - X_{j,*}^{worst}}{\left\|X_{j,*}^{best} - X_{j,*}^{worst}\right\|_E},\ j \in [1 : |S^p|],$$

где $v_{\max}$ — максимально допустимое значение шага перемещения. Если $\varphi(X_{j,*}^{worst}(t+1)) > \varphi(X_{j,*}^{worst})$, то фиксируем удачное перемещение;

4.3) если предыдущая операция не привела к успеху, то по той же схеме пытаемся улучшить положение агента $s_{j,*}^{worst}$ путем перемещения его в направлении глобального лучшего агента s^{best};

4.4) если и последняя операция не привела к улучшению позиции агента $s_{j,*}$, то взамен этого агента случайным образом создаем в области поиска нового агента;

5) повторяем процедуру меметической эволюции заданное число раз, реализуя тем самым локальный поиск в пределах каждого из мемеплексов;

6) выполняем процедуру тасования (*shuffling procedure*) агентов, которая заключается в объединении агентов всех мемефлексов в одну группу и случайном изменении их номеров. В результате при повторном выполнении шага 3 будут получены новые мемефлексы $S_j^p, j \in [1 : |S^p|]$;

7) если условие завершения итераций не выполнено, то возвращаемся к шагу 2.

Таким образом, основой алгоритма *SFL* является комбинирование локального поиска в пределах каждого из мемефлексов и глобального поиска путем обмена информацией о положениях лучших агентов этих мемефлексов и определения на этой основе глобально лучшего агента.

Известны модификации алгоритма *SFL*, в которых, например, на этапе локального поиска агент движется не точно в направлении лучшего агента соответствующего мемефлекса, а в некотором случайным образом возмущенном направлении. Известны как последовательные, так и параллельные гибридизации алгоритма со многими популяционными алгоритмами, например, с алгоритмом искусственной иммунной системы [28, 29].

2.6.2. Алгоритм, инспирированный летучими мышами (*Bat-Inspired, BI*), предложил Янг (*X.-Sh. Yang*) в 2010 г. [30].

Большинство видов летучих мышей обладает удивительно совершенными средствами эхолокации, которая используется ими для обнаружения добычи и препятствий, а также для обеспечения возможности разместиться в темноте на насесте. Параметры локирующего звукового импульса мышей различных видов меняются в широких пределах, отражая их различные охотничьи стратегии. Большинство мышей используют короткие частотно-модулированные в пределах примерно одной октавы сигналы. В то же время некоторые виды не используют частотную модуляцию своих звуковых импульсов.

Алгоритм *BI* предполагает следующую модель поведения летучих мышей:

а) с помощью эхолокации все мыши могут измерять расстояние до добычи и препятствий, а также различать их;

б) мыши движутся случайным образом. Текущие положение и скорость мыши s_i, $i \in [1 : |S|]$, равны X_i, V_i соответственно. Для поиска добычи мыши генерируют сигналы, имеющие частоту ω_i и громкость a_i. В процессе поиска мыши могут менять частоту этих сигналов, а также частоту повторения излучаемых импульсов (*rate of pulse*) $r \in [0; 1]$;

в) частота сигналов может изменяться в диапазоне $[\omega^{\min}, \omega^{\max}]$, $\omega^{\max} > \omega^{\min} \geq 0$, а громкость сигналов — в пределах [0; 1].

Положим, что речь идет о задаче глобальной безусловной минимизации. Схема алгоритма *BI* включает следующие основные шаги:

1) инициализируем популяцию агентов s_i, $i \in [1 : |S|]$. Определяем глобально лучшего агента s^{best} и соответствующее ему решение X^{best};

2) выполняем перемещение всех агентов на один шаг в соответствии с используемой миграционной процедурой;

3) для каждого из агентов s_i, $i \in [1 : |S|]$, выполняем следующие действия:

3.1) генерируем случайное число $u = U_1(0; 1)$;

3.2) если $u > r_i$, то находим лучшее решение X_i^{best} данного агента;

3.3) в окрестности решения X_i^{best} реализуем процедуру локального поиска. Принимаем найденное решение в качестве текущего положения агента s_i;

4) в окрестности текущего решения случайным образом генерируем новое решение;

5) генерируем новое случайное число $u = U_1(0; 1)$;

6) если $u < a_i$ и $\varphi(X_i) < \varphi(X^{best})$, то принимаем решение X_i в качестве нового текущего положения агента s_i, увеличиваем значение параметра r_i и уменьшаем значение параметра a_i;

7) находим новое глобально лучшее решение X^{best};

8) если условие окончания итераций не выполнено, то возвращаемся к шагу 2.

На *этапе инициализации алгоритма* начальные значения частот ω_i^0, громкостей a_i^0 и частот повторения импульсов r_i^0, $i \in [1 : |S|]$, полагаем равномерно распределенными в соответствующих интервалах $[\omega^{\min}; \omega^{\max}]$, $[a^{\min}; a^{\max}]$, $[0; 1]$.

Миграцию агента s_i, $i \in [1 : |S|]$, осуществляем по формулам

$$X_i' = X_i + V_i',$$

$$V_i' = V_i + \omega_i'(X_i - X^{best}),$$

$$\omega_i' = \omega^{\min} + (\omega^{\max} - \omega^{\min})U_1(0; 1). \qquad (2.11)$$

Другими словами, на данной итерации агент перемещается в направлении, определяемом суммой вектора перемещения на предыдущей итерации (слагаемое V_i в формуле (2.11)) и случайным образом возмущенного вектора направления на лучшего агента $(X_i - X^{best})$. Заметим, что рассмотренная процедура миграции алгоритма *BI* имеет много общего с аналогичной процедурой алгоритма роя частиц [3].

Случайный *локальный поиск* выполняем по следующей схеме:

1) случайным образом варьируем текущее положение агента s_i в соответствии с формулой

$$X_i' = X_i + \overline{a}\, U_{|X|}(-1; 1),\ i \in [1 : |S|],$$

где $\overline{a}$ — текущее среднее значение громкостей всех агентов популяции:

$$\overline{a} = \frac{1}{|S|}\sum_{i=1}^{|S|} a_i;$$

2) вычисляем значение фитнесс-функции в новой точке $\varphi(X_i') = \varphi_i'$. Если $\varphi_i' > \varphi_i$, то завершаем процедуру локального поиска, в противном случае фиксированное число раз возвращаемся к шагу 1.

Эволюция параметров a_i, r_i осуществляется по правилам

$$a_i' = b_a a_i,\ r_i' = r_i^0(1 - \exp(-b_r t)),\ i \in [1 : |S|],$$

где $b_a \in (0; 1)$, $b_r > 0$ — заданные константы (свободные параметры алгоритма), рекомендуемые значения которых равны 0,9. Другими словами, с ростом числа итераций громкость импульсов, излучаемых каждой мышью, линейно уменьшаем (добыча уже найдена), а частоту повторения импульсов r_i в тех же условиях уменьшаем по экспоненциальному закону, так что имеют место предельные соотношения

$$a_i^t \underset{t\to\infty}{\to} 0,\ r_i^t \underset{t\to\infty}{\to} r_i^0,\ i \in [1 : |S|].$$

2.6.3. Поиск косяком рыб. *Алгоритм поиска косяком рыб* (*Fish School Search*, *FSS*) предложили в 2008 г. Фило (*B. Filho*) и Нето (*L. Neto*) [31].

Косяком рыб (*fish school*) называют агрегацию рыб, которые передвигаются приблизительно с одной и той же скоростью и ориентацией, поддерживая примерно постоянное расстояние между собой. Доказано, что всякого рода объединения рыб играют важную роль в повышении эффективности поиска ими пищи, защиты от хищников, а также в уменьшении энергетических затрат.

В алгоритме *FSS* рыбы плавают в *аквариуме* (области поиска) в поисках пищи (решения задачи оптимизации). Вес каждой рыбы формализует ее индивидуальный успех в поиске решения и играет роль ее памяти. Именно наличие веса у агентов популяции является главной особенностью парадигмы *FSS* в сравнении, например, с парадигмой оптимизации роем частиц. Эта особенность алгоритма *FSS* позволяет отказаться от необходимости отыскивать и фиксировать глобально лучшие решения, как это делается в алгоритме роя частиц.

Полагаем, что решается задача глобальной условной максимизации в области D, и что всюду в этой области фитнесс-функция принимает неотрицательные значения.

Операторы алгоритма *FSS* объединены в две группы:

а) оператор кормления, формализующий успешность исследования агентами тех или иных областей аквариума;

б) операторы плавания, реализующие алгоритмы миграции агентов.

Оператор кормления (*feeding operator*). Обозначим w_i, $i \in [1 : |S|]$, текущий вес агента s_i. В алгоритме *FSS* принято, что вес агента пропорционален нормализованной разности значений фитнесс-функции на следующей и текущей итерациях, т. е.

$$w_i' = w_i + \frac{\varphi(X_i') - \varphi(X_i)}{\max(\varphi(X_i'), \varphi(X_i))},\ i \in [1 : |S|].$$

Алгоритм ограничивает максимально допустимый вес агентов величиной $w_{\max} > 0$, так что во всех случаях вес агента s_i удовлетворяет условию

$$w_i \in [1 : w_{\max}],\ i \in [1 : |S|].$$

Здесь $w_{\max}$ — свободный параметр алгоритма.

При инициализации популяции всем агентам в качестве веса присваиваем значение $\frac{w_{\max}}{2}$.

Операторы плавания (*swimming operators*). В алгоритме *FFS* различают три вида плавания — индивидуальное, инстинктивно-коллективное и коллективно-волевое. Положим, что эти виды плаваний выполняются на интервалах $(t, \tau]$, $(\tau, \theta]$, $(\theta, t']$ основного итерационного интервала $(t, t']$ соответственно. Здесь $t < \tau < \theta < t'$, $t' = t + 1$.

Индивидуальное плавание (*individual swimming*). Направление перемещения агента в этом случае равновероятно случайно. Если это перемещение выводит агента за пределы области допустимых значений D, то перемещение не выполняем. Аналогично, перемещение агента s_i не проводится, если имеет место неравенство $\varphi(X_i^{\tau}) < \varphi(X_i^{t})$, т. е. в новой точке X_i^{τ} значение фитнесс-функции не выше ее значения в предыдущей точке X_i^{t}; $i \in [1 : |S|]$.

Компоненты шага перемещения V_i^{ind} полагаем случайными величинами, равномерно распределенными в интервале $[0;\ v_{\max}^{ind}]$:

$$V_i^{ind} = U_{|X|}(0;\ 1)\, v_{\max}^{ind},\ i \in [1 : |S|].$$

Для обеспечения постепенного перехода от диверсификации поиска на начальных итерациях к его интенсификации на завершающих итерациях линейно уменьшаем величину $v_{\max}^{ind}$ (свободный параметр алгоритма) с ростом числа итераций.

Процесс индивидуального плавания может включать в себя не одну итерацию, как в рассмотренной схеме, а некоторое их фиксированное число. Таким образом, индивидуальное плавание агента можно интерпретировать как локальный поиск в окрестности текущего положения агента.

Инстинктивно-коллективное плавание (*collective-instinct swimming*) реализуем после завершения всеми $|S|$ агентами индивидуальных плаваний по формуле

$$X_i^{\theta} = X_i^{\tau} + \frac{\sum_j V_j^{ind}(\tau)(\varphi(X_j^{\tau}) - \varphi(X_j^{t}))}{\sum_j (\varphi(X_j^{\tau}) - \varphi(X_j^{t}))},\ i \in [1 : |S|]. \quad (2.12)$$

Второе слагаемое в формуле (2.12) есть не что иное, как общий для всех агентов шаг миграции, представляющий собой взвешенную сумму индивидуальных перемещений агентов. Формула (2.12) означает, что в процессе инстинктивно-коллективного плавания на каждого из агентов оказывают влияние все остальные агенты популяции, и это влияние пропорционально индивидуальным успехам агентов.

Коллективно-волевое плавание (*collective volition swimming*) выполняем вслед за инстинктивно-коллективным плаванием. Коллективно-волевое плавание заключается в смещении всех агентов в направлении текущего центра тяжести популяции, если суммарный вес косяка в результате индивидуального и инстинктивно-коллективного плавания увеличился, и в противоположном направлении — если этот вес уменьшился. Другими словами, в случае успешных в среднем указанных плаваний популяция стягивается к своему центру тяжести, т. е. повышает интенсивность поиска. В противном случае популяция расширяется от того же центра, повышая свои диверсификационные свойства.

Координаты центра тяжести X_c косяка после завершения всеми его агентами инстинктивно-коллективных плаваний определяем по формуле

$$X_c^{\theta} = \frac{\sum_i w_i^{\theta} X_i^{\theta},\ i \in [1:|S|]}{w_{\Sigma}^{\theta}},$$

где $w_{\Sigma}^{\theta} = w_{\Sigma}(\theta) = \sum_{i=1}^{|S|} w_i^{\theta}$ — текущий суммарный вес популяции.

Коллективно-волевое плавание выполняем по правилу

$$X_i' = X_i^{\theta} \pm v^{vol}(X_i^{\theta} - X_c^{\theta}),\ i \in [1 : |S|], \quad (2.13)$$

где знак плюс используем в случае $w_{\Sigma}^{\theta} > w_{\Sigma}^{\theta-1}$, а знак минус — в противном случае. Здесь $w_{\Sigma}^{\theta-1}$ — суммарный вес популяции после завершения ее агентами инстинктивно-коллективных плаваний на предыдущей итерации.

Переменная v^{vol} в формуле (2.13) определяет размер шага перемещений агентов и является случайной величиной:

$$v^{vol} = v_{\max}^{vol}\, U_1(0;\ 1).$$

Здесь $v_{\max}^{vol} > 0$ — заданная максимально допустимая длина шага (свободный параметр алгоритма). Рекомендуется линейное уменьшение величины $v_{\max}^{vol}$ с ростом числа итераций.

Известен ряд модифицированных алгоритмов *FFS*. Приведем в качестве примера так называемый *плотный алгоритм FFS* (*Density FSS*, *DFSS*), использующий модификацию операторов классического алгоритма, а также новые операторы — так называемые операторы памяти и разбиения.

Оператор памяти (*memory operator*) строится на основе $(|S| \times 1)$-векторов памяти M_i, $i \in [1 : |S|]$, агентов популяции и формализует текущее и ряд предшествующих влияний на каждого данного агента других агентов популяции.

Оператор разбиения (*partitioning operator*) использует память агентов и предназначен для формирования на основе популяции $S = (s_i;\ i \in [1 : |S|])$ ряда подпопуляций.

Известны также высокоэффективные гибридные алгоритмы, построенные на основе алгоритма *FFS*. Примером такой гибридизации могут служить *волевой алгоритм роя частиц* (*Volitive PSO*, *VPSO*) [32].

2.6.4. Алгоритм растущих деревьев (*Saplings Sowing and Growing up*, *SSG*) предложил Карчи (*A. Karci*) с соавторами в 2002 г. [33]. Алгоритм вдохновлен эволюцией растущих деревьев и включает в себя два этапа — этап посадки (*sowing phase*) и этап роста (*growing up phase*).

Рассматриваем задачу глобальной уловной оптимизации в параллелепипеде Π. На этапе посадки саженцы (начальные решения) случайным образом равномерно располагаем в области поиска — создаем равномерный сад (*uniform garden*).

Этапы роста реализуем с помощью трех операторов — оператора скрещивания (*mating operator*), оператора ветвления (*branching operator*) и оператора прививки (*vaccinating operator*).

Оператор скрещивания. Скрещивание агентов s_i, s_j, $i, j \in [1 : |S|]$, $i \neq j$, заключается в обмене между векторами X_i, X_j некоторым числом своих компонентов. Вероятность ξ_m скрещивания агентов s_i, s_j определяем на основе нормированного евклидова расстояния между ними:

$$\xi_m(s_i, s_j) = 1 - \frac{\|X_i - X_j\|_E}{r}.$$

Здесь r — длина диагонали параллелепипеда Π, т. е.

$$r = \left(\sum_{k=1}^{|X|} (x_k^+ - x_k^-)^2 \right)^{1/2}.$$

Оператор ветвления. Рассмотрим агента s_i, $i \in [1 : |S|]$, и соответствующее текущее решение X_i. Положим, что компонента $x_{i,\,k}$, $k \in [1 : |X|]$, этого решения была изменена (в k-й точке имело место ветвление). Тогда вероятность ξ_b того, что будет изменена компонента $x_{i,\,l}$, $l \in [1 : |X|]$, $l \neq k$, того же вектора (в l-й точке будет проведено ветвление) может быть вычислена по "линейной" формуле

$$\xi_b(x_{i,\,l} | x_{i,\,k}) = 1 - \frac{1}{(|l-k|)^2}$$

или "нелинейной" формуле

$$\xi_b(x_{i,\,l} | x_{i,\,k}) = 1 - \frac{1}{\exp(|l-k|)^2}.$$

Оператор прививки применяется к агентам s_i, s_j, $i, j \in [1 : |S|]$, $i \neq j$, в том случае, когда мера их близости (*similarity*) $\sigma(s_i, s_j)$ превышает порог $\varepsilon|X|$, где ε — заданная положительная константа (свободный параметр алгоритма). В качестве меры близости в оригинальном алгоритме *SSG* использована мера

$$\sigma(s_i, s_j) = \sum_{k=1}^{|X|} \frac{|x_{i,\,k} - x_{j,\,k}|}{x_k^+ - x_k^-}.$$

Прививку выполняем по правилу

$$x'_{i,\,k} = \begin{cases} x_{i,\,k}, & \dfrac{|x_{i,\,k} - x_{j,\,k}|}{x_k^+ - x_k^-} \leqslant \varepsilon, \\ x_{j,\,k}, \text{ иначе;} \end{cases}$$

$$x'_{j,\,k} = \begin{cases} x_{j,\,k}, & \dfrac{|x_{i,\,k} - x_{j,\,k}|}{x_k^+ - x_k^-} \leqslant \varepsilon, \\ x_{i,\,k}, \text{ иначе;} \end{cases}$$

$$k \in [1 : |X|].$$

Другими словами, если модуль относительной разности между k-ми компонентами векторов X_i, X_j превосходит величину ε, то эти векторы обмениваются k-ми компонентами.

После применения к текущей популяции указанных операторов получаем промежуточную популяцию (S, S'), которой соответствует объединение исходных X_i и модифицированных векторов X'_i, $i \in [1 : |S|]$. После вычисления значений фитнесс-функции во всех указанных точках из промежуточной популяции отбираем $|S|$ лучших агентов в следующую популяцию S^{t+1}. Итерационный процесс повторяется до достижения заданного числа итераций.

3. Популяционные алгоритмы, инспирированные неживой природой и человеческим обществом

Материал данного раздела можно разделить на две части. Первая часть (пп. 3.1—3.3) посвящена популяционным алгоритмам поисковой оптимизации, которые порождены неживой природой. Вторая часть (пп. 3.3—3.8) включает в себя описание алгоритмов, инспирированных человеческим обществом.

Мы не рассматриваем широко известный *алгоритм интеллектуальных капель воды* (*Intelligent Water Drop*, *IWD*), вдохновленный способностью рек прокладывать близкие к оптимальным пути своего течения. Алгоритм предназначен для решения задач на графах, таких как задача о коммивояжере (*Travelling Salesman Problem*, *TSP*) или задача экономичного распределения нагрузок (*Economic Load Dispatch Problem*, *ELDP*). По той же причине в разделе не отражен алгоритм *RED* (*River Formation Dynamics*).

3.1. Гармонический поиск

Алгоритм поиска гармонии (*Harmony Search, HS*) предложил в 2008 г. Гим (*Z. W. Geem*) [34]. Алгоритм вдохновлен процессом поиска музыкантами гармонии в музыке. Ситуации идеальной гармонии звуков алгоритм *HS* ставит в соответствие глобальный экстремум в задаче многомерной оптимизации, а процессу импровизации музыканта — процедуру поиска этого экстремума.

3.1.1. Канонический алгоритм. Рассмотрим задачу глобальной условной минимизации в параллелепипеде Π.

Музыкантам ставим в соответствие индивидов s_i, а оркестру — популяцию $S = (s_i;\ i \in [1 : |S|])$. Аккорду, который берет музыкант s_i в данный момент времени, сопоставляем значение вектора варьируемых параметров X_i. Гармонию звуков формализует глобальный экстремум фитнесс-функции $\varphi(X)$. Совокупность текущих координат X_i, $i \in [1 : |S|]$, образует так называемую $(|S| \times |X|)$-*матрицу памяти гармоний* (*harmony memory*) H_M.

Схема канонического алгоритма *HS* включает в себя следующие основные шаги:

1) инициализируем алгоритм;

2) формируем вектор гармонии;

3) выполняем пошаговую настройку вектора гармонии;

4) обновляем матрицу памяти гармоний;

5) если условие окончания итераций не выполнено, то возвращаемся к шагу 2.

Последовательно рассмотрим эти шаги.

Инициализация. Начальные значения компонентов векторов X_i полагаем равномерно распределенными в гиперпараллелепипеде Π:

$$x^0_{i,j} = x^-_j + U_1(0;\ 1)(x^+_j - x^-_j),$$
$$i \in [1 : |S|],\ j \in [1 : |X|]. \qquad (3.1)$$

Совокупность векторов X^0_i, $i \in [1 : |S|]$, образует исходную матрицу памяти гармоний

$$H_M = \begin{pmatrix} x^0_{1,1} & x^0_{1,2} & \dots & x^0_{1,|X|} \\ x^0_{2,1} & x^0_{2,2} & \dots & x^0_{2,|X|} \\ \dots & \dots & \dots & \dots \\ x^0_{|S|,1} & x^0_{|S|,2} & \dots & x^0_{|S|,|X|} \end{pmatrix}.$$

Формирование вектора гармонии X' выполняем по следующим правилам.

С вероятностью ξ_h в качестве компоненты x'_j вектора X' используем соответствующую компоненту случайного вектора X_{i_1} текущей матрицы памяти гармоний H_M:

$$x'_j = x_{i_1,j},\ i_1 \in [1 : |S|],\ i_1 \in U_1(1 : |S|),\ j \in [1 : |X|].$$

Данная операция моделирует воспроизведение музыкантом какого-либо аккорда из его памяти гармоний и в оригинале называется операцией *random selection*.

С вероятностью $(1 - \xi_h)$ в качестве компоненты x'_j берем величину, сгенерированную по формуле вида (3.1). Операция адекватна ситуации формирования абсолютно случайного аккорда из доступного музыканту диапазона.

Свободный параметр алгоритма ξ_h имеет смысл относительной частоты операции *random selection*, т. е. *вероятности использования памяти гармоний* (*Harmony Memory Considering Rate, HMCR*).

Пошаговая настройка вектора гармонии. Если компонента x'_j, $j \in [1 : |X|]$, вектора X' выбрана из памяти гармоний, то выполняем следующие действия.

С вероятностью ξ_p изменяем эту компоненту по формуле

$$x'_j = x'_j + u^{\pm 1}_{sign}\, b_w N_1(0;\ 1) \qquad (3.2)$$

или с вероятностью $(1 - \xi_p)$ оставляем x'_j без изменений. Другими словами, компонента x'_j с равной вероятностью получает случайное положительное или отрицательное приращение величиной $N_1(0;\ 1)$. Здесь параметр ξ_p определяет относительную частоту пошаговой настройки, т. е. *вероятность изменения шага* (*Pitch Adjusting Rate, PAR*), а параметр b_w представляет собой значение шага, которому придается смысл *полосы пропускания инструмента* (*Bandwidth Rate, BW*).

Обновление матрицы памяти гармоний. Вычисляем значение фитнесс-функции $\varphi(X')$, соответствующее сформированному вектору X'. Если $\varphi(X') < \varphi(X^{worst})$, то худший вектор X^{worst} текущей памяти гармоний H_M заменяем вектором X'.

Рекомендуют следующие значения основных свободных параметров алгоритма *HS*: $|S| \leqslant 50$; $\xi_h = 0{,}7 \dots 0{,}95$; $\xi_p = 0{,}1 \dots 0{,}5$. Два последних параметра используют для регулирования скорости его сходимости. Чем ниже значения этих параметров, тем алгоритм сходится медленнее.

Наиболее существенное влияние на эффективность алгоритма *HS* оказывает свободный параметр b_w, который, вообще говоря, может принимать значения в диапазоне $(x^+_j - x^-_j)$, и априорные рекомендации по выбору его значений затруднительны. Известен, например, вариант вычисления динамических значений этой величины по формуле

$$b_w(t) = b^-_w + \frac{b^+_w - b^-_w}{\hat{t} - 1}(t - 1), \qquad (3.3)$$

где b^0_w — заданный шаг; b^-_w, b^+_w — его начальное и конечное значения; b^v_w — так называемое варьируе-

мое значение шага; $n_{ch}(t)$, $n_{ch}^{\max}$ — текущее и максимально допустимое числа обновлений памяти гармоний соответственно.

3.1.2. Некоторые модификации алгоритма. Для повышения эффективности канонического алгоритма *HS* разработано большое число его модификаций. Рассмотрим основные направления модификации алгоритма [35].

- Использование альтернативных правил инициализации популяции. Известна, например, модификация алгоритма *HS*, которая предполагает генерацию $2|S|$ векторов X_i^0 и выбор из них $|S|$ лучших.
- Различные динамические стратегии изменения значений свободных параметров алгоритма. К примеру, в так называемом *улучшенном алгоритме гармонического поиска* (*Improved Harmony Search*, *IHS*) используется динамическое обновление величин b_w, ξ_p по формулам

$$b_w(t) = b_w^+ \exp\left(\ln\left(\frac{b_w^-}{b_w^+}\right)\frac{t}{\hat{t}}\right),$$

$$\xi_p(t) = \xi_w^- + (\xi_w^+ - \xi_p^-)\frac{t}{\hat{t}},$$

где b_w^-, b_w^+, ξ_p^-, ξ_p^+ — начальные и конечные значения этих параметров соответственно [36].

- Новые или модифицированные операторы канонического гармонического поиска. Приведем несколько примеров.

В модификации алгоритма *HS* [37] при определении вектора X' вместо формулы (3.2) используют формулу вида

$$x_j' = x_j' + u_{sign}^{\pm 1} b_w E_1(a, b),$$

где $E_1(a, b)$ — вещественное случайное число, распределенное по экспоненциальному закону с параметрами a, b. Рекомендованные авторами алгоритма значения этих величин равны 0,3 и 0,1 соответственно.

Дифференциальный алгоритм гармонического поиска (*Differential Harmony Search*, *DHS*) [38] предполагает вычисление компонент вектора X' на основе оператора дифференциальной эволюции [1]:

$$x_j' = x_j' + b_h(x_{i_1,j} - x_{i_2,j}),$$
$$j \in [1 : |X|],\ i_1, i_2 \in [1 : |S|],\ i_1 \neq i_2,$$

где b_h — скалярный свободный параметр.

Глобально лучший алгоритм гармонического поиска (*Global Harmony Search*, *GHS*) [39] использует при определении вектора X' элементы алгоритма роя частиц [3]. Вектор X' выбирают в данном случае таким образом, чтобы он в некотором смысле был близок к текущему лучшему вектору памяти гармоний X^{best}. Отметим, что тем самым в алгоритм гармонического поиска добавляется социальный аспект.

- Различные варианты учета ограничений при формировании вектора X'. Возможно, например, использование известного алгоритма на основе скользящего допуска.
- Различные критерии включения нового вектора X' в память гармоний. Например, таким критерием может быть выполнение не условия $\varphi(X') < \varphi(X^{worst})$, как в каноническом алгоритме *HS*, а условия $\varphi(X') < \overline{\varphi}$, где $\overline{\varphi}$ — средняя пригодность гармоний в текущей матрице памяти гармоний. Известны критерии, построенные на основе близости в некоторой метрике вектора X' и векторов X_i, $i \in [1 : |S|]$.
- Изменение условий окончания итераций.
- Изменение структуры алгоритма. Эти изменения могут быть небольшими, как, например, многократные гармонии, предложенные Ли (*L. Li*) с соавторами. Изменения могут быть также кардинальными, основанными на низкоуровневой гибридизации вложением алгоритма *HS* с другими популяционными и не популяционными алгоритмами. В настоящее время известны такие гибридизации с одним, двумя и тремя алгоритмами.

3.2. Алгоритм гравитационного поиска

Алгоритм гравитационного поиска (*Gravitational Search*, *GS*) предложил Рашеди (*E. Rashedi*) с соавторами в 2009 г. [40]. Алгоритм использует аналогии движения тяжелых тел вследствие их гравитационного взаимодействия. Положим, что речь идет о задаче глобальной условной максимизации в области D.

Алгоритм *GS* является развитием детерминированного алгоритма глобального многомерного поиска, получившего название алгоритма оптимизации центральной силой (*Central Force Optimization*, *CFO*) [41]. Рассмотрим сначала данный алгоритм.

3.2.1. Алгоритм оптимизации центральной силой. Алгоритм *CFO* представляет собой детерминированную модель зондов, движущихся в области поиска под действием силы тяжести. Исходное распределение зондов в области поиска является детерминированным и равномерным.

Полагаем, что каждый из зондов s_i имеет переменную массу, равную значению функции приспособленности $\varphi(X)$ в точке текущего положения зонда. Координаты зондов изменяем в процессе итераций по формуле

$$x_{i,j}' = x_{i,j} + \frac{1}{2}a_{i,j}, \qquad (3.4)$$

где величину $a_{i,j}$ интерпретируем как текущее ускорение зонта s_i по j-координате, равное

$$a_{i,j} = \gamma \sum_k \chi\left(\operatorname{sign}(\varphi_k - \varphi_i)(\varphi_k - \varphi_i)^b \frac{x_{k,j} - x_{i,j}}{\|X_k - X_i\|^c}\right),$$

$$k, i = 1, 2, \ldots, |S|, k \neq i, j \in [1 : |X|], \varphi_l = \varphi(X_l). \quad (3.5)$$

Здесь $\chi(\bullet)$ — функция Хэвисайда, используемая в целях исключения возможности получения отрицательной массы; b, c, γ — заданные константы, из которых γ отождествляют с гравитационной постоянной.

Если некоторый зонд "вылетает" из области решений D, то его следует возвратить в середину между его предыдущим и текущим (недопустимым) положениями.

3.2.2. Схема алгоритма *GS*. Основные отличия алгоритма *GS* от алгоритма *CFO* состоят в следующем: алгоритм *GS* является стохастическим, а алгоритм *CFO* — детерминированным; массы и координаты зондов в указанных алгоритмах вычисляются по несколько отличным формулам; начальное распределение зондов в алгоритме *CFO* является детерминированным, а в алгоритме *GS* носит случайный характер; в алгоритме *CFO* гравитационная постоянная γ является постоянной, тогда как в алгоритме *GS* она по некоторому правилу изменяется в процессе поиска.

Гравитационную постоянную γ в алгоритме *GS* уменьшают с ростом числа итераций в соответствии с формулой

$$\gamma(t) = \gamma(0)\left(\frac{1}{t}\right)^d, \; d \in (0; 1),$$

где $\gamma(0)$ — ее начальное значение, а d — свободный параметр алгоритма.

В отличие от того, как это имеет место в природе, алгоритм *GS* в общем случае использует три вида гравитирующих масс — активную массу m^A, пассивную массу m^P и инертную массу m^I.

Полагаем, что гравитационная сила $F_{i,j}$, которая действуют на массу m_i со стороны массы m_j, равна

$$F_{i,j} = \gamma \frac{m_i^P m_j^A}{r_{i,j}}, \; i, j \in [1 : |S|], \; i \neq j,$$

т. е. обратно пропорциональна не квадрату расстояния между зондами, а лишь первой степени этого расстояния.

Ускорение, приобретаемое массой m_i в результате действия силы $F_{i,j}$, определяет формула

$$a_i = \frac{F_{i,j}}{m_i^I}.$$

Говоря более строго, алгоритм *GS* использует следующее правило вычисления силы, действующей на массу m_i со стороны массы m_j по k-му измерению:

$$F_{i,j,k} = \gamma \frac{m_i^P m_j^A}{r_{i,j} + \varepsilon}(x_{j,k} - x_{i,k}), \; k \in [1 : |X|].$$

Здесь ε — малая константа, $r_{i,j}$ — текущее евклидово расстояние между массами m_i, m_j.

Суммарная сила $F_{i,*,k}$, действующая на зонд s_i по k-му измерению, полагается случайной величиной, равной

$$F_{i,*,k} = \sum_{j=1, j\neq i}^{|S|} U_1(0; 1) F_{i,j,k}. \quad (3.6)$$

Ускорение указанного зонда по тому же измерению находим по формуле

$$a_{i,*,k} = \frac{F_{i,*,k}}{m_i^I}.$$

Если положение зонда s_i по k-му измерению на данной итерации есть $x_{i,k}$, то на следующей итерации его определяет выражение

$$x'_{i,k} = x_{i,k} + v'_{i,k}, \; i \in [1 : |S|], \; k \in [1 : |X|], \quad (3.7)$$

где скорость $v'_{i,k}$ равна

$$v'_{i,k} = U_1(0; 1) v_{i,k} + a_{i,k}. \quad (3.8)$$

Простейший алгоритм *GS* предполагает равенство активной, пассивной и инертной масс m^A, m^P, m^I, точнее говоря, полагается, что

$$m_i = \frac{\mu_i}{\sum_j \mu_j}, \; i \in [1 : |S|], \; j \in [1 : |S|],$$

где

$$\mu_i = \frac{\varphi(X_i) - \varphi^{worst}}{\varphi^{best} - \varphi^{worst}}.$$

Эффективность поиска с помощью алгоритма *GS* может быть повышена, если с ростом числа итераций сокращать по некоторому правилу в формуле (3.6) число зондов, оставляя лишь наиболее массивные из них. Обозначим S^{best} текущий набор лучших (самых тяжелых) зондов, I^{best} — номера этих зондов, а $|S^{best}|$ — их число. Тогда модифицированная формула (3.6) примет вид

$$F_{i,*,k} = \sum_j U_1(0; 1) F_{i,j,k}, \; j \in I^{best}, \; j \neq i, \; k \in [1 : |X|].$$

Число $|S^{best}|$ лучших зондов с ростом числа итераций уменьшают, например, по линейному закону так,

что, в конце концов, остается лишь один зонд, притягивающий остальные зонды.

3.2.3. Некоторые модификации алгоритма *GS*. Оригинальный алгоритм *GS* склонен к преждевременной сходимости в случае поиска экстремума сложных мультимодальных целевых функций. В связи с этим разработано значительное число модификаций алгоритма, имеющих целью преодоление данного его недостатка.

Вариантом решения данной проблемы является введение в оригинальный алгоритм *GS* операторов так называемых знаковой мутации (*sign mutation*) и переупорядочивающей мутации (*reordering mutation*) [42].

Знаковая мутация предполагает вычисления нового положения зонда s_i по k-му измерению по формулам, несколько отличным от формул (3.7), (3.8) и имеющим вид

$$x'_{i,k} = x_{i,k} + \tilde{v}'_{i,k},\ i \in [1 : |S|],\ k \in [1 : |X|], \qquad (3.9)$$

$$\tilde{v}'_{i,k} = \bar{u}_{\text{sign}}(\xi_s)\, v'_{i,k},$$

где $\bar{u}_{\text{sign}}(\xi_s)$ — случайная величина, с вероятностью ξ_s принимающая значение (-1) и с вероятностью $(1 - \xi_s)$ — значение $(+1)$:

$$\bar{u}_{\text{sign}}(\xi_s) = \begin{cases} -1, & U_1(0;\ 1) < \xi_s; \\ 1, & \text{иначе.} \end{cases}$$

Таким образом, знаковая мутация с вероятностью ξ_s изменяет соответствующую компоненту вектора скорости зонда, вычисленную по правилу (3.8).

Переупорядочивающую мутацию применяют к вектору скорости $v'_{i,k}$, вычисленному по формуле (3.8), с вероятностью ξ_r. Мутация состоит в случайном переупорядочении компонент этого вектора.

Комбинация знаковой и переупорядочивающей мутации обеспечивает в широких пределах варьирование ориентации вектора $\tilde{v}'_{i,k}$. Кроме того, для диверсификации поиска вектор $x'_{i,k}$, вычисленный по формуле (3.9), может быть подвергнут равномерной мутации [1].

3.3. Электромагнитный поиск

Рассматриваем два близких алгоритма, основанных на использовании электростатических законов Кулона и законов механики Ньютона — электромагнитный алгоритм и алгоритм поиска системой зарядов.

В данных алгоритмах каждого из агентов популяции интерпретируют как электрически заряженную частицу, заряд которой пропорционален значению фитнесс-функции в той точке области поиска, в которой на данной итерации находится агент. Текущий заряд частиц популяции определяет суммарную силу, действующую на данную частицу, а также направление и значение ее перемещения на текущей итерации. В соответствии с законами электростатики эта сила вычисляется путем векторного суммирования сил притяжения и отталкивания со стороны других частиц популяции. Рассматриваем задачу глобальной минимизации в параллелепипеде Π.

3.3.1. Электромагнитный алгоритм (*ElectroMagnetism-like algorithm*, *EM algorithm*) предложили Бирбил (*I. Birbil*) и Фанг (*S. C. Fang*) в 2003 г.

Общая схема канонического алгоритма *EM* включает в себя следующие шаги:

1) инициализируем популяцию;

2) выполняем локальный поиск;

3) вычисляем суммарные силы, действующие на каждую из частиц популяции;

4) реализуем перемещение частиц;

5) если условие окончания итераций не выполнено, то переходим к шагу 2.

Локальный поиск выполняем для каждого из текущих положений частиц s_i, $i \in [1 : |X|]$, в целях сбора локальной информации о ее окружении. Вообще говоря, этот поиск может быть реализован с помощью любого из детерминированных или стохастических алгоритмов локальной оптимизации. Авторы алгоритма *ES* предлагают использовать линейный стохастический поиск [43].

Вычисление суммарных сил. Если частица s_i находится в электростатическом поле частиц s_j, $j \in [1 : |X|]$, $j \neq i$, то справедлив принцип суперпозиции сил, в соответствии с которым результирующий вектор $\vec{F}_i$ сил, действующих на частицу s_i, представляет собой сумму векторов $\vec{F}_{i,j}$:

$$\vec{F}_i = \sum_{j=1,\, j\neq i}^{|S|} \frac{q_i q_j}{4\pi\varepsilon r_{i,j}}\, \vec{e}_{ij}.$$

Здесь $\vec{e}_{ij}$ — единичный вектор направления $\sum_{j=1,\, j\neq i}^{|S|} \vec{F}_{i,j}$.

На данной итерации заряду частицы s_i ставим в соответствие величину

$$q_i = \exp\left(-|X|\frac{\varphi_i - \varphi^{best}}{\sum_j (\varphi_j - \varphi^{best})}\right),$$

$$i \in [1 : |S|],\ j \in [1 : |S|],\ j \neq i. \qquad (3.10)$$

Множитель $|X|$ в формулу (3.10) добавлен в целях предотвращения слишком малых абсолютных значений величины под знаком экспоненциальной функции при высоких размерностях поискового пространства. Аргумент этой функции во всех случаях неположителен, так что заряд q_i всегда положителен и принадлежит интервалу (0; 1].

На той же итерации силе $\vec{F}_i$, $i \in [1 : |S|]$, ставим в соответствие $(|X| \times 1)$-вектор $F_i = (F_{i,j}, j \in [1 : |X|])$, компоненты которого определяет формула

$$F_i = \sum_{\substack{j=1 \\ j \neq i}}^{|S|} F_{i,j} = \sum_{\substack{j=1 \\ j \neq i}}^{|S|} \begin{cases} (X_j - X_i)\dfrac{q_i q_j}{\|X_j - X_i\|^2}, \varphi(X_j) < \varphi(X_i), \\ (X_i - X_j)\dfrac{q_i q_j}{\|X_j - X_i\|^2}, \varphi(X_i) \leqslant \varphi(X_j). \end{cases} \quad (3.11)$$

Из последней формулы следует, что частица с лучшим значением фитнесс-функции притягивает частицу с худшими значениями этой функции, и наоборот — вторая частица отталкивает первую. Из этой же формулы следует, что частица, имеющая на данной итерации лучшее значение фитнесс-функции, притягивает на этой итерации все остальные частицы популяции (подобно глобально лучшей частице в алгоритме роя частиц).

Перемещение частиц выполняем по правилу

$$X_i(t+1) = X_i + U_1(0;1)\frac{F_i}{\|F_i\|}V_i,$$
$$i \in [1 : |S|],\ X_i \neq X^{best}, \quad (3.12)$$
$$X^{best}(t+1) = X^{best}.$$

Здесь компоненты вектора V_i имеют следующие значения:

$$V_{i,j} = \begin{cases} (x_j^+ - x_{i,j}),\ F_{i,j} > 0, \\ (x_{i,j} - x_j^-),\ F_{i,j} \leqslant 0, \end{cases} \quad j \in [1 : |X|],\ j \neq i. \quad (3.13)$$

Формулы (3.12), (3.13) означают, что при перемещении частицы s_i из положения X_i в положение X_i' используется нормированная сила (3.11). По каждому из измерений вектора X перемещение выполняется с шагом случайного размера в направлении соответствующей верхней или нижней границ параллелепипеда Π. Частица s^{best} на данной итерации не перемещается.

Известно большое число модификаций алгоритма *EM*, часть из которых предлагают сами авторы алгоритма.

Локальный поиск. Прежде всего, процедура локального поиска вообще может быть исключена из схемы алгоритма. Исследования авторов алгоритма показывают, что такая модификация во многих случаях обеспечивает вполне приемлемую эффективность алгоритма по критерию вероятности локализации глобального экстремума многоэкстремальной функции. Процедура локального поиска может быть применена не ко всем частицам популяции, как в каноническом алгоритме *EM*, а только к лучшей на данной итерации частице. Такая модификация имеет целью сократить число испытаний, требуемых процедурой, с тем, чтобы обеспечить лучший баланс между вычислительной сложностью и точностью алгоритма.

Алгоритм линейного стохастического поиска, используемый в процедуре локального поиска канонического алгоритма *EM*, может быть заменен другими, более мощными популяционными и не популяционными алгоритмами. Например, известны гибридизации алгоритма *EM* с алгоритмом рассеянного поиска [44].

Правила вычисления сил F_i, $i \in [1 : |S|]$. Известно несколько вариантов правила вычисления зарядов q_i, фигурирующих в формуле (3.11) [45]. Так, может быть использовано масштабирование этих зарядов по формуле

$$q_i = \exp\left(-|X|\frac{\varphi_i - \varphi^{best}}{\varphi^{worst} - \varphi^{best}}\right),\ i \in [1 : |S|].$$

Возможна замена экспоненциальной функции другими монотонными функциями, например, функцией вида

$$q_i = \frac{1}{|X|\dfrac{\varphi_i - \varphi^{best}}{\varphi^{worst} - \varphi^{best}}},\ i \in [1 : |S|],$$

которая также гарантирует выполнение условия $q_i \in (0; 1]$, но обеспечивает по сравнению с предыдущей формулой с ростом размерности задачи $|X|$ более быстрое убывание зарядов частиц, отличных от лучшей частицы.

Известен вариант вычисления сил F_i с учетом их значений на предыдущей итерации:

$$F_i^t = F_i^t + \beta F_i^{t-1},\ i \in [1 : |S|],\ t > 1.$$

Здесь $\beta \in [0; 1]$ — так называемая константа памяти алгоритма.

Кардинальная модификация канонического алгоритма вычисления сил F_i, $i \in [1 : |S|]$, основана на использовании так называемой возмущающей частицы (*perturbed point*) $s^{per} = s_{i_p}$ в качестве которой может выступать частица, располагающаяся на данной итерации наиболее далеко от лучшей частицы s^{best}:

$$X^{per} = \max_{i \in [1:|S|]} \|X^{best} - X_i\|_E.$$

Для всех $i \in [1 : |S|]$, $i \neq i_p$, сила F_i вычисляется по формуле (3.11). Для частицы s^{per} эта формула несколько трансформируется и приобретает вид

$$F_{i_p,j} = \begin{cases} (X_j - X^{per})\dfrac{U_1(0;1)q^{per}q_j}{\|X_j - X^{per}\|^2},\ \varphi(X_j) < \varphi(X^{per}); \\ (X^{per} - X_j)\dfrac{U_1(0;1)q^{per}q_j}{\|X_j - X^{per}\|^2},\ \text{иначе}; \end{cases}$$

$$F_{i_p,j} := \begin{cases} F_{i_p,j},\ U_1(0;1) < \xi_v; \\ -F_{i_p,j},\ \text{иначе}. \end{cases}$$

Здесь ξ_v — вероятность инверсии компоненты $F_{i_p, j}$ вектора F_{i_p} (свободный параметр алгоритма).

Закон перемещения частиц. Очевидной является модификация формулы (3.12) с использованием адаптивной длины шага b^t перемещения частицы:

$$X_i' = X_i + b^t U_1(0; 1)\frac{F_i}{\|F_i\|} V_i,\ i \in [1 : |S|],\ X_i \neq X^{best},$$

$$X^{best}(t+1) = X^{best}.$$

Можно предложить различные законы изменения величины b^t, например, авторы алгоритмы *EM* предлагают закон

$$b^{t+1} = \frac{b^t}{1 + \alpha b^t},\ b^0 = 1,$$

где $\alpha \in (0; 1)$ — заданная константа (свободный параметр алгоритма).

Учет ограничений. Канонический алгоритм *EM* предполагает решение задачи оптимизации в параллелепипеде *П*. Для решения задачи условной оптимизации с ограничениями общего вида могут быть использованы различные варианты барьерных и штрафных функций.

3.3.2. Алгоритм поиска системой зарядов. Алгоритм поиска системой зарядов (*Charged System Search*, *CSS*) представили Кавех (*A. Kaveh*) и Талатахари (*S. Talatahari*) в 2010 г. [46].

В обозначениях п. 3.3.1 в алгоритме *CSS* текущий заряд q_i частицы $s_i \in S$ определяет выражение

$$q_i = \frac{\varphi_i - \varphi^{best}}{\varphi^{best} - \varphi^{worst}},\ i \in [1 : |S|],$$

а текущее расстояние $r_{i,j}$ между частицами s_i, s_j — выражение

$$r_{i,j} = \frac{\|X_i - X_j\|_E}{\left\|\frac{X_i - X_j}{2} - X^{best}\right\|_E + \varepsilon},\ i, j \in [1 : |S|],$$

где положительная малая константа ε введена для того, чтобы избежать сингулярности.

Так же, как в алгоритме *EM*, силы взаимодействия между частицами популяции могут быть притягивающими и отталкивающими. Характер этой силы между частицами s_i, $s_j \in S$ определяет константа $c_{i,j}$, имеющая вид

$$c_{i,j} = \begin{cases} -1, & \varphi_i < \varphi_j; \\ 1, & \varphi_i \geqslant \varphi_j. \end{cases}$$

В отличие от алгоритма *EM* каждая из заряженных частиц $s_i \in S$ представляет собой сферу, радиус которой равен величине $a > 0$ (свободный параметр алгоритма). В результате этого соглашения суммарная сила, действующая на частицу $s_j \in S$ со стороны всех остальных частиц популяции, приобретает вид

$$F_i = q_i \sum_{\substack{j=1 \\ j \neq i}}^{|S|} \left(\frac{q_j r_{i,j} \beta_1}{a^3} + \frac{q_j \beta_2}{r_{i,j}^2} \right) c_{i,j}(X_j - X_i),\ i \in [1 : |S|],$$

где значения величин β_1, β_2 определяют условия

$$\beta_1 = 0,\ \beta_2 = 1, \text{ если } r_{i,j} \geqslant a,$$

$$\beta_1 = 1,\ \beta_2 = 0, \text{ если } r_{i,j} < a.$$

Миграцию частиц в области поиска определяет выражение

$$X_i' = U_1(0; 1) b_a \frac{F_i}{q_i} + U_1(0; 1) b_v V_i + X_i,\ i \in [1 : |S|].$$

Здесь первое слагаемое моделирует ускорение частицы, в второе — ее скорость; b_a, b_v — весовые коэффициенты, определяющие веса ускорения и скорости соответственно (свободные параметры алгоритма);

$$V_i' = X_i' - X_i,\ i \in [1 : |S|].$$

Относительные значения величин b_a, b_v определяют баланс между диверсификацией и интенсификацией поиска. По одной из рекомендаций законы изменения этих параметров имеют вид

$$b_a^t = 3\left(1 - \frac{t}{\hat{t}}\right),\ b_v^t = 1 + \frac{t}{\hat{t}}.$$

Схема алгоритма *CSS* близка приведенной выше схеме алгоритма *EM*.

3.4. Алгоритм эволюции разума

Концепцию *алгоритма эволюции разума* (*mind evolutionary computation*, *MEC*) предложил в 1998 г. Ченгай (*S. Chengyi*) с соавторами. Алгоритм *MEC* моделирует некоторые аспекты поведения человека в обществе, а не работу человеческого мозга, как можно было бы предположить. В алгоритме *MEC* каждый индивид рассматривается как разумный агент, функционирующий в некоторой группе людей. При принятии решений он ощущает влияние как со стороны членов своей группы, так и со стороны членов других групп. Точнее говоря, чтобы достичь высокого положения в обществе, индивиду приходится учиться у наиболее успешных индивидов в своей группе. В то же время, для того чтобы группа, которой принадлежит данный индивид, становилась более успешной по сравнению с другими группам, этот индивид, как и все индивиды его группы, должны руководствоваться тем же самым принципом в межгрупповой конкуренции [47].

Алгоритм *MEC* удобно интерпретировать как многопопуляционный алгоритм. Мультипопуляция алгоритма *MEC* состоит из *лидирующих групп* (*superior groups*)

$\mathbf{S}^b = (S_1^b, S_2^b, ..., S_{|\mathbf{S}^b|}^b)$ и *отстающих групп* (*temporary groups*) $\mathbf{S}^w = (S_1^w, S_2^w, ..., S_{|\mathbf{S}^w|}^w)$, числом $|\mathbf{S}^b|$ и $|\mathbf{S}^w|$ соответственно. В оригинальном алгоритма *MEC* числа агентов в каждой из групп полагаются одинаковыми и равными $|S|$.

Каждая из групп S_i^b, S_j^w имеет свою коммуникационную среду, которая названа авторами алгоритма *локальной доской объявлений* (*local billboard*). Обозначим эти доски C_i^b, C_j^w соответственно. Кроме того, вся мультипопуляция ($\mathbf{S}^b$, $\mathbf{S}^w$) имеет общую *глобальную доску объявлений* (*global billboard*) C^g.

Если далее в обозначениях групп и соответствующих локальных досок объявлений индексы b, w опущены, то имеется в виду произвольная из этих групп и их досок объявлений. Полагаем, что речь идет о задаче условной максимизации в параллелепипеде $П$.

3.4.1. Простой алгоритм эволюции разума. Исходная версия алгоритма *MEC*, названная авторами *простым алгоритмом эволюции разума* (*Simple MEC*, *SMEC*), построена на основе операций инициализации групп, *локальных состязаний* (*similar-taxis*) и *диссимиляции* (*dissimilation*) [47].

Операция инициализации групп создает группы $\mathbf{S}^b$, $\mathbf{S}^w$ из $|S|$ агентов каждая и размещает их в области поиска. Рассмотрим схему операции на примере группы S_i, $i \in [1 : |\mathbf{S}|]$:

1) генерируем случайный вектор $X_{i,\,1}$, компоненты которого равномерно распределены в области поиска $П$. Отождествляем этот вектор с индивидом $s_{i,\,1}$ группы S_i;

2) определяем начальные координаты остальных агентов данной группы $s_{i,\,j}$, $j \in [2 : |S|]$, по формуле

$$X_{i,\,j} = X_{i,\,1} + N_{|X|}(0, \sigma), \qquad (3.14)$$

т. е. размещаем случайным образом вокруг агента $s_{i,\,1}$ в соответствии с нормальным законом распределения.

Операция локальных состязаний реализует локальный поиск максимума фитнесс-функции каждой из групп S_i^b, S_j^w, $i \in [1 : |\mathbf{S}^b|]$, $j \in [1 : |\mathbf{S}^w|]$. Рассмотрим в качестве примера схему этой операции для группы S_i:

1) с доски объявлений C_i берем информацию о текущем агенте-победителе группы S_i. Пусть это будет агент $s_{i,\,k}$, $k \in [1 : |S|]$;

2) определяем новые координаты остальных агентов $s_{i,\,l}$, $l \in [1 : |S|]$, $j \neq k$, данной группы по правилу вида (3.14), т. е. размещаем случайным образом вокруг победителя в соответствии с нормальным законом распределения;

3) вычисляем новые *счета* (*scores*) агентов группы $\varphi_{i,\,l} = \varphi(X_{i,\,l})$, $l \in [1 : |S|]$;

4) определяем нового победителя группы $s_{i,\,j}$, $m \in [1 : |S|]$, т. е. агента данной группы, который имеет максимальный текущий счет:

$$\max_j \varphi_{i,\,j} = \varphi(X_{i,\,m}) = \varphi_{i,\,m},\ j \in [1 : |S|];$$

5) заносим информацию о новом победителе группы $s_{i,\,m}$ на доски объявлений C_i, C^g.

Операция диссимиляции управляет глобальным поиском. Схема операции имеет следующий вид:

1) с глобальной доски объявлений C^g считываем счета φ_i^b, φ_j^w, $i \in [1 : |\mathbf{S}^b|]$, $j \in [1 : |\mathbf{S}^w|]$, всех групп (т. е. текущие счета победителей этих групп);

2) выполняем сравнение указанных счетов между собой. Если счет некоторой лидирующей группы S_i^b меньше счета одной из отстающих групп S_j^w, то последняя группа занимает место группы S_i^b во множестве лидирующих групп, а группа S_i^b — место группы S_j^w среди отстающих групп. Если счет группы S_j^w ниже счетов всех лидирующих групп, то удаляем группу S_j^w из популяции;

3) с помощью операции инициализации вместо каждой из удаленных групп инициализируем новую группу.

В алгоритме *SMEC* операции локальных состязаний и диссимиляции итерационно повторяются до тех пор, пока имеет место увеличение максимального счета лидирующих групп. При прекращении роста этого счета решение задачи, соответствующее победителю лучшей из лидирующих групп, объявляем точкой глобального экстремума.

3.4.2. Расширенный алгоритм эволюции разума (*Extended Mind Evolutionary Computation*, *EMEC*) использует модифицированные операции локальных состязаний и диссимиляции, а также новую *операцию кооперации* (*cooperation*) [48]. В алгоритме *EMEC* каждая из лидирующих и отстающих групп в процессе своего развития проходит три стадии развития: *стадию инициализации* (*initialization stage*), *стадию эволюции* (*evolution stage*) и *стадию зрелости* (*maturity stage*). Группа приобретает статус зрелой, если ее счет не изменяется в течение заданного числа итераций. Если в ходе эволюции данная группа сближается на некоторое фиксированное расстояние с другой группой или перекрывает ее, то первая из групп получает статус *ассимилирующей группы* (*assimilative group*). Информация о статусе группы заносится на глобальную доску объявлений.

Пусть $s_{i,\,k}$, $k \in [1 : |S|]$, — текущий победитель группы S_i. Тогда в *модифицированном операторе локальных состязаний* координаты остальных агентов этой группы на следующей итерации определяет формула

$$\begin{cases} X_{i,\,j} = X_{i,\,k} + N_1(0; 1)V_j, \\ V_j = \rho V_j, \end{cases} \quad j \in [1 : |S|],\ j \neq k, \qquad (3.15)$$

где V_j — $(|X| \times 1)$-вектор текущего *шага обучения* (*learning step*) индивида $s_{i,\,j}$, $\rho \in (0; 1)$ — заданный *когнитивный параметр* (*cognitive parameter*), обеспечивающий уменьшение шага V_j с ростом числа итераций.

Направление шага V_j в формуле (3.15) совпадает с направлением $(X_{i,\,k} - X_{i,\,j})$, т. е. с направлением от текущего положения индивида $s_{i,\,j}$ к текущему поло-

жению победителя данной группы. Размер шага V_j определяет значение обучающего воздействия победителя на индивида $s_{i,j}$ и является некоторой возрастающей функцией разности счетов победителя и данного индивида $(\varphi_{i,k} - \varphi_{i,j})$. В простейшем случае в качестве этой функции используют линейную функцию вида

$$V_j = \alpha(\varphi_{i,k} - \varphi_{i,j}),$$

где α — положительный коэффициент пропорциональности (свободный параметр).

В процессе выполнения *модифицированной операции диссимиляции*, в первую очередь, проверяем текущий статус группы. Если она имеет статус "зрелая" или "ассимилирующая", то удаляем ее из популяции, а вместо нее создаем новую группу. Первого агента новой популяции, к примеру, агента $s_{j,1}$, в отличие от алгоритма *MEC* размещаем в области поиска не с помощью генератора случайных равномерно распределенных чисел, а на основе алгоритма имитации отжига [49]. Заметим, что вследствие этого алгоритм *EMEC* следует классифицировать как гибридный популяционный алгоритм.

Опять же, в отличие от алгоритма *MEC*, полученные координаты $X_{j,1}$ агента $s_{j,1}$ сравниваем с текущими координатами победителей всех групп (информацию берем с глобальной доски объявлений C^g). Новую группу в окрестности точки $X_{j,1}$ инициализируем только в том случае, если расстояние между агентом $s_{j,1}$ и этими победителями превышает заданное минимально допустимое расстояние.

На содержательном уровне *операция кооперации* алгоритма *EMEC* состоит в том, что в процессе эволюции каждая из групп на основе информации с глобальной доски объявлений C^g находит соседнюю лучшую группу (обладающую более высоким счетом) и изменяет свое положение в пространстве поиска, перемещаясь в направлении этой группы.

Пусть $s_{i,j}$ — текущий победитель группы S_i. Положим, что $s_{k,l}$ — текущий победитель соседней лучшей группы S_k, так что справедливо неравенство $\varphi_k > \varphi_i$. Тогда операцию кооперации определяет формула

$$\begin{cases} X_{i,j} = X_{i,j} + N_1(0;1)V_i, \\ V_i = \lambda(X_{k,l} - X_{i,j}), \end{cases}$$
$$i, k \in [1 : |\mathbf{S}|], j \neq k, j, l \in [1 : |S|],$$

где $\lambda > 0$ — величина, определяющая длину шага.

Как и в алгоритме роя частиц [3], в общем случае при определении соседней лучшей группы S_k могут быть использованы различные топологии соседства.

3.4.3. Улучшенный алгоритм. Дальнейшее развитие *MEC*-алгоритма представляет собой так называемый *улучшенный MEC-алгоритм* (*Improved Mind Evolutionary Computation*, *IMEC*) [50]. Основные особенности улучшенного алгоритма эволюции разума заключаются в следующем: при инициализации групп алгоритм использует стратегию с разделением области поиска на подобласти; используется самоадаптирующийся алгоритм, реализующий операцию локальных состязаний; операция диссимиляции расширена средствами нишевания [1].

Стратегия инициализации с разделением пространства поиска на подобласти (*region-partition initialization strategy*) имеет целью предотвратить преждевременную сходимость алгоритма. Идея состоит в том, что на этапе начальной инициализации мультипопуляции $(\mathbf{S}^b, \mathbf{S}^w)$ область поиска Π тем или иным образом разделяют на параллелепипеды Π_1, Π_2, ... равного объема, и начальные точки каждой из указанных групп генерируют внутри этих подобластей случайным образом на основе равномерного закона распределения.

Самоадаптирующийся механизм (*self-adaptive mechanism*) операции локальных состязаний использует расширенный вектор варьируемых параметров задачи, включающий в себя помимо вектора X также вектор шага миграции индивида V. Для группы S_i основная формула самоадаптирующейся операции локальных состязаний имеет вид

$$\begin{cases} x'_{i,j,k} = x_{i,l,k} + N_1(0,1)v'_{i,j,k}; \\ v'_{i,j,k} = \mu v_{i,l,k} + N_1(0,1)a_{i,j,k}, \end{cases} \qquad (3.16)$$
$$i \in [1 : |\mathbf{S}|], j, l \in [1 : |S|], j \neq l, k \in [1 : |X|],$$

где $x_{i,j,k}$, $v_{i,j,k}$ — компоненты расширенного вектора $(X_{i,k}, V_{i,j})$ варьируемых параметров агента $s_{i,j} \in S_i$; $x_{i,l,k}$, $v_{i,l,k}$ — компоненты аналогичного вектора $(X_{i,l}, V_{i,l})$ победителя данной группы; $\mu \in (0; 1)$ — свободный вещественный параметр, определяющий значение шага $v'_{i,j,k}$; параметр $a_{i,j,k}$ определяет вклад случайной составляющей в значение этого шага и равен

$$a_{i,j,k} = \frac{x_{i,l,k} - x_{i,l,k}(t^-_{i,l,k})}{t - t^-_{i,l,k}}.$$

Здесь $t^-_{i,l,k}$ — номер итерации, на котором был выявлен агент $s_{i,l} \in S_i$ в качестве победителя группы S_i.

В качестве критерия окончания итераций (3.16) модифицированная операция локальных состязаний использует условие

$$\frac{1}{|X|}\sum_{k=1}^{|X|} v_{i,l,k} \leqslant \varepsilon_v. \qquad (3.17)$$

Отметим, что условие (3.17) одновременно является условием зрелости группы S_i.

Нишевание в модифицированной операции диссимиляции реализовано путем задания минимально допустимого расстояния между двумя разными группами.

В качестве расстояния между группами S_i, S_j, $i \neq j$, операция использует величину

$$\rho(S_i, S_j) = \sqrt{\frac{1}{|X|}\sum_{k=1}^{|X|}\left(\frac{x_{i,m,k} - x_{j,n,k}}{x_k^+ - x_k^-}\right)^2},$$

где $s_{i,m}$, $s_{j,n}$ — победители групп S_i, S_j соответственно.

3.4.4. Хаотический алгоритм эволюции разума (*Chaotic Mind Evolutionary Computation*, *CMEC*), можно сказать, представляет собой гибридизацию рассмотренного выше алгоритма *MEC* и хаотического алгоритма локального поиска [51].

Алгоритм *CMEC* использует при начальной инициализации мультипопуляции *палаточное отображение*, а для инициализации временных групп, которые в процессе функционирования алгоритма создаются вместо удаленных групп, — *логистическое отображение*. Использование не классических генераторов случайных чисел, а хаотических последовательностей, обеспечивает более равномерное распределение агентов в области решения. Операция локализации алгоритма *CMEC* построена на основе хаотического алгоритма локального поиска.

3.5. Стохастический диффузионный поиск

Алгоритм стохастического диффузионного поиска (*Stochastic Diffusion Search*, *SDS*) представил в 1989 г. Бишоп (*J. Bishop*). До настоящего времени алгоритм, преимущественно, развивается его автором, а также Насуто (*S. Nasuto*) [52]. По сравнению с другими популяционными алгоритмами данный алгоритм отличает достаточно глубокое математическое обоснование. В исходном виде алгоритм *SDS* представляет собой алгоритм дискретной оптимизации. Модификация алгоритма для задачи глобальной непрерывной оптимизации была предложена только в 2011 г.

Алгоритм инспирирован игрой "Ресторан". Суть игры состоит в следующем.

Группа делегатов участвует в длительной конференции в незнакомом городе, в котором имеется большое число ресторанов. Делегаты ставят перед собой задачу найти лучший из ресторанов, т. е. такой, который понравится наибольшему числу делегатов. Проверка каждого ресторана и его ассортимента каждым делегатом займет слишком много времени. Поэтому делегаты используют следующую стратегию поиска. Каждый делегат формирует свою гипотезу относительно того, какой ресторан является лучшим в городе. Вечером он проверяет свою гипотезу, ужиная в этом ресторане и заказывая одно случайное блюдо из меню. На следующее утро те делегаты, которые не были довольны своим ужином, просят поделиться своими впечатлениями случайного коллегу. Если его впечатления положительны, то недовольный делегат принимает выбор этого коллеги. В противном случае, он выбирает новый случайный ресторан.

В терминах алгоритма *SDS* схема рассмотренной стратегии поиска лучшего ресторана имеет следующий вид:

1) на этапе *инициализации* алгоритма каждый из агентов высказывает начальную *гипотезу* (*hypothesis*) о лучшем ресторане;

2) каждый агент *тестирует* (*test*) свою гипотезу, т. е., обедая в выбранном ресторане, подтверждает или опровергает свое первоначальное мнение;

3) путем прямых контактов между агентами происходит *диффузия* (*diffusion*) или обмен гипотезами и результатами их проверки;

4) на этой основе агенты формируют новые гипотезы, тестируют их и вновь обмениваются гипотезами. И так далее до выполнения условий *сходимости* (*convergence*, *halt*) итераций.

Рассмотрим задачу глобальной минимизации в параллелепипеде Π.

Тестирование. Приспособленность каждого из агентов s_j, $i \in [1 : |S|]$, сравниваем с приспособленностью агента s_j, $j \in [1 : |S|]$, $j \neq i$. Если $\varphi_i \leqslant \varphi_j$, то агента s_i объявляем *активным агентом* (*active agent*), а в противном случае — *неактивным агентом* (*inactive agent*).

Диффузия. Для каждого из агентов s_i, $i \in [1 : |S|]$, выполняем следующие действия:

1) агент s_i неактивен. Выбираем случайного агента s_j, $j \in [1 : |S|]$, $j \neq i$. Если этот агент является активным, то агент s_i присваивает гипотезу из некоторой окрестности агента s_j, т. е. полагаем $X_i = d(X_j)$, $\varphi_i = \varphi(X_i)$, где окрестность $d(X_j)$ равна

$$d(X_j) = \{X | x_j^l \leqslant x_k \leqslant x_j^u,\ k \in [1 : |X|]\};$$

$$x_j^l = \max((x_j - 0{,}5b(x_j^+ - x_j^-),\ x_j^-),$$

$$x_j^u = \max((x_j + 0{,}5b(x_j^+ - x_j^-),\ x_j^+).$$

Значение свободного параметра b лежит в интервале (0; 1) и обычно равно 0,1;

2) агент s_i активен. Выбираем для этого агента новую гипотезу. Выбор может проводиться случайно или детерминировано. Рекомендуют выбор гипотезы в центре наибольшей из текущих неисследованных областей множества Π по следующей схеме:

- сортируем текущие координаты всех агентов популяции по каждому из измерений x_k;
- находим наибольший интервал между двумя соседними точками по каждому из измерений x_k; пусть координаты этих точек равны $x_k^{\min} < x_k^{\max}$, $k \in [1 : |X|]$;
- помещаем агента s_i в точку X_i с координатами

$$x_{i,k} = x_k^{\min} + 0{,}5(x_k^{\max} - x_k^{\min}),\ k \in [1 : |X|].$$

Сходимость. Алгоритм *SDS* может использовать различные условия окончания итераций и, в частности, так называемые, сильный и слабый критерии.

Сильный критерий (*strong halting criterion*) использует тот факт, что в процессе итераций алгоритма *SDS* имеет место кластеризация, т. е. формирование подмножеств (кластеров) агентов, разделяющих одну и ту же гипотезу. При этом самый многочисленный кластер включает в себя оптимальное решение. В соответствии с сильным критерием итерации завершают при выполнении следующих условий: мощность самого большого кластера превышает минимально допустимое значение; эта мощность не меняется в течение заданного числа итераций (имеет место стагнация размера кластера).

Слабый критерий (*weak halting criterion*) предполагает завершение итераций в следующих условиях: число активных агентов превышает минимально допустимое значение; это число не меняется в течение заданного числа итераций (имеет место стагнация числа активных агентов).

Особенностью алгоритма *SDS* является достаточно высокая рекомендуемая численность популяции $|S|$ — от 100 до 1000 агентов.

Канонический алгоритм *SDS* является неадаптивным (все свободные параметры определены априори). Адаптивной модификацией канонического алгоритма *SDS* является *фокусированный алгоритм* (*focused SDS*). Канонический алгоритм *SDS* является однопопуляционным. Вместе с тем, известны многопопуляционные модификации этого алгоритма, например, так называемый *контекстно-зависимый SDS* (*context-sensitive SDS*).

3.6. Культурный алгоритм

Культурный алгоритм (*Cultural Algorithms, CA*) представляет собой, по сути, концепцию совершенствования популяционных алгоритмов поисковой оптимизации путем учета опыта, приобретаемого в ходе решения задачи. Принято считать, что алгоритм предложил Рейнольдс (*R. Reynolds*) в 1994 г. [53, 54].

В процессе развития общества люди накапливают информацию об окружающем мире. Эту информацию можно считать базой знаний общества, которую его члены могут эксплуатировать в целях оптимизации своего поведения. В культурном алгоритме эту базу знаний формализуют в виде *пространства убеждений* (*belief space*) и используют в качестве еще одного элемента эволюционного воздействия на популяцию. Таким образом, если говорить, например, о генетическом алгоритме, то его "культурная" модификация предполагает учет как генетической, так и культурной информации.

3.6.1. Общая схема алгоритма. Рассмотрим задачу глобальной условной минимизации с ограничениями типа неравенств

$$\min_{X \in D \cap \Pi} f(X) = f(X^*) = f^*,$$

$$D = \{X \mid G(X) \geqslant 0\},\ \Pi = \{X \mid x_i^- \leqslant x_i \leqslant x_i^+,\ i \in [1 : |X|]\}.$$

Назовем ограничения, определяемые ограничивающей вектор-функцией $G(X)$ и параллелепипедом Π, *ограничениями задачи* и *ограничениями области допустимых значений* соответственно. Положим, что множество $D \cap \Pi$ является выпуклым, и всюду в этом множестве фитнесс-функция $\varphi(X)$ принимает неотрицательные значения.

Основными компонентами культурного алгоритма являются популяция и пространство убеждений.

Популяция и пространство убеждений взаимодействуют посредством *функции принятия* (*acceptance function*) и *функции влияния* (*influence function*). С помощью первой из указанных функций определяют множество наилучших агентов, которые имеют возможность корректировать пространство убеждений. Вторая функция задает правила, по которым убеждения способны влиять на эволюцию агентов популяции.

В простейшем случае функция принятия использует агентов с приспособленностью, которая выше средней приспособленности по популяции. При построении функции влияния могут быть использованы два подхода — изменение приспособленности агентов и изменение принципов их модификации.

Псевдокод культурного алгоритма представлен ниже.

Begin
  $t = 0$;
  Initialize Population $S(t)$;
  Initialize Beliefspace $B(t)$;
  Evaluate Population $S(t)$;
  repeat
    Communicate ($S(t)$, $B(t)$);
    Adjust Beliefspace $B(t)$;
    Communicate ($B(t)$, $S(t)$);
    $t = t + 1$;
    Evolve Population $S(t)$;
    Evaluate Population $S(t)$;
  until (*termination condition*)
End

Здесь функции *Initialize Population* $S(t)$ и *Initialize Belief space* $B(t)$ выполняют инициализацию популяции и пространства убеждений соответственно. Функция *Evaluate Population* $S(t)$ оценивает популяцию с помощью фитнесс-функции. Функции *Communicate* ($S(t)$, $B(t)$) и *Communicate* ($B(t)$, $S(t)$) реализуют взаимодействие между популяцией и пространством убеждения посредством функций принятия и влияния. Функция *Adjust Belief space* B (t) выполняет коррекцию пространства убеждений, а функция *Evolve Population* $S(t)$ отвечает за эволюцию популяции на текущей итерации.

3.6.2. Построение и коррекция пространства убеждений. Как отмечалось выше, пространство убеждений содержит информацию об ограничениях задачи. Эта информация выражается в форме *сети ограничений* (*interval constraint-network*), которая состоит из узлов (*nodes*), соединенных между собой посредством *ветвей* (*branches*). Узел представляет собой текущий

интервал или несвязанное множество интервалов, определяемых ограничениями области допустимых значений и ограничениями задачи. Ветвь показывает зависимость узлов, которые она связывает.

Коррекцию пространства убеждений осуществляют путем коррекции сети ограничений с помощью рассматриваемых ниже операций. Каждую из этих операций применяют к текущим интервалам, связанным с соответствующими узлами сети ограничений.

Операцию конкретизации (*specialization*) используют в том случае, когда число отобранных агентов в пределах некоторых частей интервала меньше заданного минимального числа элементов.

Операцию обобщения (*generalization*) применяют в случае, когда хотя бы некоторые из отобранных агентов находятся за пределами текущего интервала. Результатом операции является расширенный интервал, включающий в себя всех этих агентов.

Операция разделения (*fission*) представляет собой особый случай операции конкретизации, при котором внутреннюю пустую часть текущего интервала отбрасывают в целях получения двух интервалов.

Операция слияния (*fusion*) предназначена для объединения двух несвязных интервалов при условии существования между ними отобранных агентов.

Операция распространения (*propagation*) позволяет скорректировать ограничения задачи на основе измененных ограничений области допустимых значений. Операция использует результаты интервальной арифметики.

Операция обратного распространения (*back propagation*) обратна операции распространения и предназначена для коррекции ограничений области допустимых значений на основе скорректированных ограничений задачи.

С использованием рассмотренных операций обновление сети ограничений проводим по следующей схеме:

- текущие значения агентов, отобранных функцией принятия, отображаем в соответствующие ограничения области допустимых значений и ограничения задачи;
- корректируем сеть последовательным применением операторов распространения и обратного распространения.

3.6.3. Культурная модификация генетических операторов. Как отмечалось выше, возможна культурная модификация любого из популяционных алгоритмов поисковой оптимизации, т. е. гибридизация этих алгоритмов с культурным алгоритмом. Гибридизация сводится к культурной модификации миграционных операторов соответствующего популяционного алгоритма.

Рассмотрим в качестве примера культурную модификацию некоторых таких операторов, используемых в известной программной системе *GENOCOP* (*GEnetic algorithm for Numerical Optimization for COnstrained Problems*) [55].

Оператор граничной мутации [1]. Если $X = (x_1, ..., x_k, ..., x_{|X|})$ — хромосома до мутации, то граничную мутацию этой хромосомы выполняют путем случайного выбора гена x_k в и присваивания этому гену значения

$$x_k' = \begin{cases} l_k(X), \ U_1(0:1) = 0, \\ u_k(X), \ U_1(0:1) = 1. \end{cases} \qquad (3.18)$$

Здесь $l_k(X)$, $u_k(X)$ — нижняя и верхняя границы области $D \cap \Pi$ по k-му измерению в точке X соответственно.

Культурную модификацию оператора граничной мутации определяет правило

$$x_k' = \begin{cases} l_k(X), (l_k(X) \in [\bullet]_k) \wedge (u_k(X) \notin [\bullet]_k), \\ u_k(X), (u_k(X) \in [\bullet]_k) \wedge (l_k(X) \notin [\bullet]_k), \\ x_k', \text{ иначе}, \end{cases} \qquad (3.19)$$

где $[\bullet]_k$ — текущий интервал пространства убеждений по k-измерению; величина x_k' в правой части равенства вычислена по формуле (3.18). Для случая $|X| = 2$ правило (3.19) иллюстрирует рисунок (показана вторая ситуация).

Оператор неравномерной мутации Михалевича [1] определяет формула

$$x_k' = \begin{cases} x_k + v(u_k(X) - x_k), \ U_1(0:1) = 0, \\ x_k - v(x_k - l_k(X)), \ U_1(0:1) = 1, \end{cases}$$

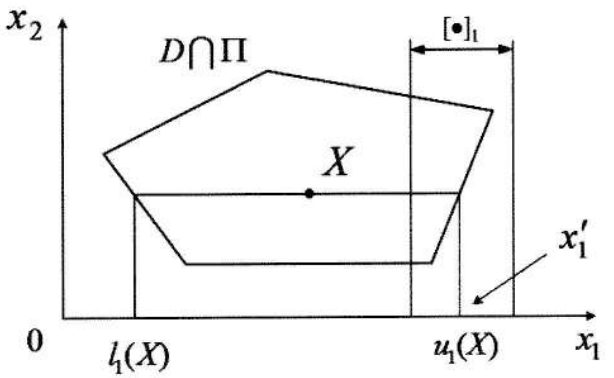
где

$$v(y) = yU_1(0:1)(1 - t/T)^b;$$

b — свободный параметр оператора.

Культурный аналог данного оператора имеет вид

$$x_k' = \begin{cases} x_k + v(u_k(X) - x_k), (d[\bullet]_k > 0) \wedge (x_k < c[\bullet]_k), \\ x_k - v(x_k - l_k(X)), (d[\bullet]_k < 0) \wedge (x_k < c[\bullet]_k), \\ x_k \pm v(x_k - l_k), \text{ иначе}, \end{cases} \qquad (3.20)$$



К культурному аналогу граничной мутации: $k = 1$

где

$$d[\bullet]_k = \begin{cases} 1, \ (c[\bullet]'_k - c[\bullet]_k) > 0, \\ -1, \ (c[\bullet]'_k - c[\bullet]_k) < 0, \\ 0, \text{ иначе,} \end{cases}$$

$c[\bullet]_k$ — координата центра текущего интервала пространства убеждений по k-измерению.

Гауссов оператор мутации [1] задает выражение

$$x'_k = x_k + N_1(m, \sigma).$$

Культурный вариант гауссова оператора мутации определяет выражение вида (3.20) с тем отличием, что в качестве шага v используется величина

$$v = \begin{cases} N_1(0, 0{,}1) \text{ при } ([\bullet]^+_k - [\bullet]^-_k) > 0{,}1; \\ N_1(0, ([\bullet]^+_k - [\bullet]^-_k)), \text{ иначе,} \end{cases}$$

где величина $([\bullet]^+_k - [\bullet]^-_k)$ представляет собой длину текущего интервала пространства убеждений по k-измерению.

3.7. Меметические алгоритмы

Википедия определяет *мемéтику* (*memetics*) как подход к эволюционным моделям передачи информации, который основывается на концепции *мемов* (*meme*), а мемы, являющиеся аналогами генов, — как единицы культурной информации, распространяемые между людьми посредством имитации, научения и др. Понятие мема и основы меметики разработаны Докинзом (*C. R. Dawkins*) в 1976 г.

Меметические алгоритмы (*Memetic Algorithms*, *MAs*) определяют как гибридные популяционные метаэвристические алгоритмы поисковой оптимизации, основанные на концепции мема и неодарвиновском принципе эволюции [56, 57]. В контексте меметических алгоритмов мем является реализацией какого-либо алгоритма локальной оптимизации, уточняющего текущие решения на каждой итерации, либо через некоторое число итераций. В широком смысле, меметические алгоритмы можно рассматривать как гибридизацию одного из популяционных алгоритмов глобального поиска и одного или нескольких классических или популяционных алгоритмов локальной оптимизации.

Первоначально меметические алгоритмы были предложены как один из вариантов повышения эффективности генетических алгоритмов (*GA-MA hybrids*). Однако схема комбинации глобального и локального поиска, использованная в *GA-MA* гибридизациях, может быть применена для построения поисковых алгоритмов также на основе других популяционных алгоритмов.

3.7.1. Общая схема алгоритмов. Рассматриваем задачу глобальной условной минимизации. Общую схему меметических алгоритмов передает следующая последовательность шагов:

1) инициализируем популяцию $S^0 = (s_i^0, i \in [1 : |S^0|])$; $|S^0| = |S|$;

2) для текущего положения X_i каждого из агентов s_i вычисляем соответствующее значение фитнесс-функции φ_i, $i \in [1 : |S|]$;

3) исходя из значений φ_i, $i \in [1 : |S|]$, формируем промежуточную популяцию S', включающую в себя $|S'|$ лучших индивидов текущей популяции S;

4) применяем ко всем агентам популяции S' операторы рассматриваемого популяционного алгоритма;

5) исходя из новых текущих положений каждого из агентов s'_i, $i \in [1 : |S'|]$, выполняем локальный поиск;

6) для найденных новых положений X'_i агентов вычисляем соответствующие значения фитнесс-функции φ'_i; $i \in [1 : |S'|]$;

7) на основе популяций S^t, S' формируем новую популяцию S^{t+1};

8) если условия окончания итерации не выполнены, то переходим к шагу 2.

Очевидно, что общая схема меметических алгоритмов оставляет широкие возможности для разработки многих его вариантов. Укажем следующие основные направления конструирования этих вариантов:

- локальный поиск можно использовать не на каждой итерации, а через некоторое фиксированное или изменяющееся по определенному закону число итераций;
- наряду с приведенным выше правилом возможны другие правила отбора агентов в промежуточную популяцию S'. Эти правила могут быть как статическими, не зависящими от номера итерации, так и динамическими, изменяющимися тем или иным образом с ростом номера итерации;
- могут быть применены различные условия окончания локального поиска (опять же как статические, так и динамические);
- для всех агентов популяции S' на всех итерациях может быть использован один и то же алгоритм локальной оптимизации или разные алгоритмы для разных групп агентов и/или для различных итераций.

Из числа перечисленных направлений конструирования меметических алгоритмов чаще всего используют вариант, порождающий *мультимемеевые* (*multi-memes*) адаптивные алгоритмы. Основной задачей конструирования таких алгоритмов является задача выбора целесообразной стратегии использования того или иного мема из роя доступных мемов $M = (m_j, j \in [1 : |M|])$. Задача относится к классу задач метаоптимизации. Указанные стратегии иногда называют *гиперэвристиками* (*hyperheuristic*).

Выбор мемов из роя мемов M может происходить в зависимости от значений их некоторых характеристик

и/или рассматриваемого в данный момент фрагмента области решения. Обычно в качестве характеристики мемов, на основе которой происходит их выбор, используют некоторую оценку эффективности мемов.

3.7.2. Гиперэвристические мультимемевые адаптивные алгоритмы. Наиболее известны следующие три категории гиперэвристик, используемых в *гиперэвристических адаптивных мультимемеевых алгоритмах*: случайные гиперэвристики (*random hyperheuristics*); жадные гиперэвристики (*greedy hyperheuristics*); гиперэвристики с функцией выбора (*choice-function hyperheuristics*).

Случайные гиперэвристики. В этой категории гиперэвристик самой простой является стратегия *простого случайного выбора* (*simple random choice*), когда в каждой точке принятия решения мем выбирают случайным образом из роя мемов M, и вероятность выбора каждого из мемов не меняется в процессе итераций.

Другой известной стратегией данной категории является стратегия *случайного спуска* (*random descent choice*). Здесь на первом шаге мем выбирают из роя M случайным образом. Выбранный мем используют до тех пор, пока он не перестанет обеспечивать локальные уточнения решений. Затем случайным образом выбирается другой мем и т. д.

Вариантом последней стратегии является стратегия *спуска со случайной перестановкой* (*random permutation descent choice*). В этом варианте вначале фиксируется случайная перестановка $\tilde{M}$ мемов $m_j, j \in [1 : |M|]$. После того, как используемый мем m_j исчерпает себя, используют следующий мем m_{j+1} в перестановке $\tilde{M}$.

Жадная гиперэвристика предполагает, что сначала все мемы $m_j, j \in [1 : |M|]$, применяются к каждому из агентов популяции S', а затем выбирается мем, показавший наилучший результат. Недостатком данной эвристики являются высокие вычислительные затраты.

Гиперэвристики с функцией выбора основаны на использовании функции выбора $\phi(m_j)$, которая может быть построена на основе одной или нескольких метрик, отражающих различные аспекты качества мемов. Известным вариантом функции $\phi(m_j)$ является трехкомпонентная функция

$$\phi(m_j, m_k) = \lambda_1\phi_1(m_j) + \lambda_2\phi_2(m_j, m_k) + \lambda_3\phi_3(m_j), \quad j, k \in [1 : |M|], j \neq k,$$

представляющая собой аддитивную свертку функций $\phi_1(m_j)$, $\phi_2(m_j, m_k)$, $\phi_3(m_j)$ с весами $\lambda_1, \lambda_2, \lambda_3$. Полагаем, что лучшему мему соответствует большее значение функции $\phi(m_j, m_k)$; $m_k \in M$ — предыдущий использованный мем.

Компонента $\phi_1(m_j)$ формализует последние улучшения фитнесс-функции, сделанные мемом m_j, и выражает ту идею, что если данный мем показал высокую эффективность на предшествующих итерациях, то, вероятнее всего, он будет эффективным и в течение некоторого числа последующих итераций.

Вторая компонента $\phi_2(m_j, m_k)$ учитывает возможный синергетический эффект от последовательного применения мемов m_j, m_k. Точнее говоря, эта компонента формализует последние улучшения фитнесс-функции, сделанные комбинацией (последовательным использованием) этих мемов.

Третья компонента $\phi_3(m_j)$ призвана повысить разнообразие используемых мемов путем увеличения вероятности применения тех из них, которые давно не выполняли локальную оптимизацию. В качестве значений функции $\phi_3(m_j)$ может быть использовано число итераций меметического алгоритма, в течение которых мем m_j не использовался ни разу.

Известно несколько гиперэвристик на основе функции выбора. Простейшей является стратегия *прямого выбора* (*straight choice*), предполагающая, что на каждом шаге принятия решения выбирается мем m', который обеспечивает наилучшее значение функции выбора $\phi(m_j, m_k)$:

$$\phi(m', m_k) = \max_j \phi(m_j, m_k), j \in [1 : |M|], j \neq k.$$

В стратегии *ранжированного выбора* (*ranked choice*) вначале мемы $m_j, j \in [1 : |M|]$, ранжируют согласно их текущим значениям функции выбора $\phi(m_j, m)$. Затем проводят эксперименты по локальному уточнению текущих решений с помощью нескольких лучших мемов. В качестве мема m' используют мем, обеспечивший наибольшее относительное уменьшение значения фитнесс-функции.

Стратегия *рулеточного выбора* (*roulette choice*) основана на выборе мема $m' = m_j$ с вероятностью

$$\xi_j = \frac{\phi(m_j, m_k)}{\sum_l \phi(m_l, m_k)}, \; j, \; l \in [1 : |M|], \; l \neq k.$$

В стратегии *декомпозиционного выбора* (*decomposition choice*), в первую очередь, формируют набор мемов $(m_{j_1}, m_{j_2}, m_{j_3}, m_j) = M'$, которые обеспечивают максимальные значения функций $\phi_1(m_j, m_k)$, $\phi_2(m_j, m_k)$, $\phi_3(m_j, m_k)$, $\phi(m_j, m_k)$ соответственно; $j_1, j_2, j_3, j \in [1 : |M|]$, $j_1, j_2, j_3, j \neq k$. Затем проводят эксперименты по локальному уточнению текущих решений с помощью каждого из мемов роя M'. Аналогично стратегии ранжированного выбора в качестве мема m' используют лучший из испытанных мемов. Заметим, что в рое M' могут быть совпадающие мемы, так что число различных мемов в этом рое может принимать значения от единицы до четырех.

Известна также стратегия *выбора с запретами* (*tabu-search choice*), в которой на основе функции выбора $\phi(m_j, m_k)$ сначала рой мемов M сужают до роя M'' путем формирования в каждой точке принятия решения списка запрещенных мемов (*tabu-list*). Затем из

роя M'' выбирают мем m' по схеме стратегии декомпозиционного выбора.

Гиперэвристики могут строиться также на основе идеи *декомпозиции на подзадачи* (*sub-problem decomposition*). Эта идея предполагает: а) динамическое разбиение области решения на подобласти, а самой задачи оптимизации — на соответствующие подзадачи; б) выбор наиболее подходящего мема для каждой из этих подзадач. При этом выборе используется информация, накопленная в процессе решения задачи, об эффективности локализации каждым из мемов роя M решения на конкретном участке области поиска. В результате для каждого из этих участков применяется специализированный мем, что, как правило, значительно повышает эффективность поиска.

3.7.3. Самоадаптирующиеся мультимемевые алгоритмы. *Самоадаптирующиеся мультимемеевые алгоритмы* представляют собой меметическую модификацию генетического алгоритма. Каждого агента s_i, $i \in [1 : |S|]$, популяции S в этом случае кодируют мультихромосомой $\bar{X}_i = (X_i, m_{i_k})$, $i_k \in [1 : |M|]$, первая часть которой представляет собой соответствующее решение задачи, а вторая часть — мем, с помощью которого получено данное решение. По общему правилу эволюции мультихромосом генетические операторы применяют к каждой из указанных частей хромосомы в отдельности, не скрещивая эти части между собой. Таким образом, данные алгоритмы реализуют метод самоадаптивной метаоптимизации [53].

Мутация мемеевой части мультихромосомы $\bar{X}_i$ состоит в случайной замене по тому или иному правилу мема m_{i_k} новым мемом из роя M.

Скрещивание мемеевых частей хромосом $\bar{X}_i$, $\bar{X}_j$, $i, j \in [1 : |\bar{S}|]$, $i \neq j$ проводим по следующей схеме. Потомок $\bar{X}_i'$ приобретает мем m_{i_k} в том случае, если оценка улучшения решения, достигнутая данным мемом, лучше такой же оценки мема m_{j_k}. В противном случае потомку передаем мем m_{j_k}. Если оценки эффективности мемов родителей m_{i_k}, m_{j_k} одинаковы, то потомок получает одни из этих мемов, выбранный случайным образом.

3.8. Самоорганизующийся миграционный алгоритм

Самоорганизующийся миграционный алгоритм (*Self Organizing Migrating*, *SOM*) был представлен Зелинка (*I. Zelinka*) в 1999 г. [58]. Алгоритм вдохновлен коллективным поведением интеллектуальных индивидуумов.

Положим, что речь идет о задаче безусловной максимизации. В алгоритме *SOM* популяция агентов эволюционирует в циклах, которые называют *миграционными циклами* (*migration loops*). В каноническом алгоритме число миграционных циклов (число поколений) фиксировано и равно $\hat{t}$.

Общую схему канонического алгоритма *SOM* передает следующая последовательность его основных шагов:

1) инициализируем популяцию S;

2) применяем к текущей популяции миграционные операторы базового популяционного алгоритма;

3) объявляем агента s^{best}, которому соответствует лучшее значение фитнесс-функции, текущим лидером популяции;

4) для каждого из агентов s_i, $i \in [1 : |S|]$, исключая лидера s^{best}, выполняем следующие действия:

4.1) принимаем в качестве лучшего положения агента s_i его текущее положение: $X_i^{best} = X_i$, $\varphi_i^{best} = \varphi(X_i)$;

4.2) перемещаем агента s_i в направлении лидера s^{best} на фиксированное число шагов n. На каждом шаге этого перемещения вычисляем соответствующее значение фитнесс-функции агента и, если оно лучше текущего значения φ_i^{best}, обновляем это значение;

4.3) перемещаем агента s_i в новое лучшее положение;

5) если условия окончания поиска не выполнены, то возвращаемся к шагу 2.

Миграционный цикл составляют операторы 2—5 рассмотренной схемы алгоритма *SOM*.

Шаг перемещения агента s_i в направлении лидера s^{best} выполняем в соответствии с формулой

$$X_i' = X_i + \lambda V_i \otimes (X^{best} - X_i),$$

где $\lambda \in (0; 1]$ — длина текущего шага; V_i — $(|X| \times 1)$-мерный рандомизирующий случайный бинарный вектор, компоненты которого определяет выражение

$$v_{i,j} = \begin{cases} 0, & U_1(0; 1) \leqslant \xi_m, \\ 1, & \text{иначе}; \end{cases} \quad j \in [1 : |X|].$$

Здесь $\xi_m \in (0; 1]$ — вероятность мутации (свободный параметр алгоритма).

Рандомизация в алгоритме *SOM* преследует те же цели, что мутация в генетическом алгоритме, т. е., прежде всего, обеспечение разнообразия популяции. Подчеркнем, что в текущем миграционном цикле для каждого из агентов s_i рандомизация проводится только один раз, так что направление движения в сторону лидера в процессе этого движения не меняется. Нулевое значение компоненты $v_{i,j}$ означает, что перемещение агента s_i по j-му измерению в данном миграционном цикле запрещено (это измерение "заморожено"). Таким образом, единичный вектор V_i означает отсутствие рандомизации.

Обычно в алгоритме *SOM* разрешено замораживание не всех $|X|$ компонентов вектора X_i, а лишь их некоторого числа $k \leq |X|$. Таким образом, в каждом миграционном цикле поиск проводится только по $(|X| - k)$

измерениям. Номера замораживаемых компонентов полагают случайными величинами.

Рекомендуются следующие значения свободных параметров алгоритма: $\lambda \in [1{,}1; 3]$; $\xi_m = 0{,}1$.

Заключение

В обзор не вошел непрерывный вариант *алгоритма рассеянного поиска* (*Scatter Search*), который был представлен Гловером (*F. W. Glover*) в 1977 г. В обзоре также не нашел места *алгоритм прокладки путей* (*Path Relinking*), который предложили тот же Гровер и Лагуна (*M. Laguna*) в 1997 г. [59, 60]. Оба указанных алгоритма предназначены для решения задачи глобальной условной оптимизации. Последний алгоритм был предложен как подход, призванный интегрировать интенсификационные и диверсификационные стратегии поиска в контексте алгоритма поиска с запретами [61, 62].

В настоящее время основным средством конструирования новых высокоэффективных популяционных алгоритмов является гибридизация (*hybridization*) известного популяционного алгоритма с двумя и более известными популяционными и/или не популяционными алгоритмами [63]. Объем обзора позволил включить в него лишь очень небольшую часть примеров гибридных популяционных алгоритмов.

Как отмечалось выше, одной из особенностей популяционных алгоритмов является наличие в них значительного, а часто, большого числа свободных параметров. От значений этих параметров может сильно зависеть эффективность алгоритма, однако формальные рекомендации по выбору значений этих параметров исходя из особенностей решаемой задачи оптимизации, как правило, отсутствуют. В связи с этим в настоящее время интенсивно развиваются методы адаптации и самоадаптации значений этих параметров, общим для которых является наименование *метаоптимизация* (*meta-optimization*) [64]. Ограниченный объем работы не позволил рассмотреть данную проблематику.

В связи с высокой вычислительной сложностью практически значимых задач глобальной оптимизации и доступностью различных высоко- и сверхвысокопроизводительных вычислительных систем все более актуальной становится проблема разработки параллельных популяционных алгоритмов, ориентированных на различные классы параллельных ЭВМ. По указанной причине в работе не рассмотрены также параллельные популяционные алгоритмы.

При решении практических, прежде всего, технических задач оптимизации часто возникает задача многокритериальной (*multi criteria*) оптимизации, когда желательно экстремизировать не одну, а несколько целевых функций. Хорошо известно, что, поскольку нельзя добиться экстремального значения всех целевых функций одновременно, решение такой задачи представляет собой некоторый компромисс, принадлежащий так называемому множеству Парето (*Pareto set*) задачи, т. е. множеству решений, не улучшаемых одновременно по всем критериям. Часто решением задачи многокритериальной оптимизации называют множество Парето этой задачи, предполагая, что после построения некоторой аппроксимации данного множества, лицо, принимающее решение, неформальными или формализованными методами выберет из него единственное решение. Известно большое число весьма не тривиальных популяционных алгоритмов решения (в указанном смысле) задачи многокритериальной оптимизации. Мы не имели возможность включить в обзор данный материал.

Предполагается, что указанные вопросы составят предмет самостоятельного обзора или обзоров.

Список литературы

1. **Курейчик В. М.** Генетические алгоритмы и их применение. Таганрог: Изд-во Таганрогского РТУ, 2002. С. 244.
2. **Рутковская Д., Пилиньский М., Рутковский Л.** Нейронные сети, генетические алгоритмы и нечеткие системы: Пер. с польск. М.: Горячая линия — Телеком, 2004. С. 452.
3. **Карпенко А. П., Селиверстов Е. Ю.** Глобальная оптимизация методом роя частиц. Обзор // Информационные технологии. 2010. № 2. С. 25—34.
4. **МакКоннелл Дж.** Основы современных алгоритмов. М.: Техносфера, 2004. С. 368.
5. **Субботин С. А., Олейник Ал. А.** Мультиагентная оптимизация на основе метода пчелиной колонии // Кибернетика и системный анализ. 2009. № 3. С. 15—25.
6. **Дасгупта Д.** Искусственные иммунные системы и их применение. М.: Физматлит, 2006. С. 344.
7. **Hart E., Timmis J.** Application Areas of AIS: Past, Present and Future // Journal of Applied Soft Computing. 2008. V. 8 (1). P. 191—201.
8. **Herrera-Lozada J. C., Calvo H., Taud H.** A Micro Artificial Immune System // Polibits. 2011. V. 43. P.107—111.
9. **Engelbrecht A.P.** Computational Intelligence. An Introduction. Second Edition. John Wiley & Sons, 2007. P. 597.
10. **Das S., Biswas A., Dasgupta S., Abraham A.** Bacterial Foraging Optimization Algorithm: Theoretical Foundations, Analysis, and Applications // Foundations of Computational Intelligence. Publisher: Springer, 2009. V. 203. P. 23—55.
11. **Chen H., Zhu Yu., Hu K.** Cooperative Bacterial Foraging Optimization // Discrete Dynamics in Nature and Society. 2009. V. 2009. P. 1—17.
12. **Eiben A. E., Michalewicz Z., Schoenauer M., Smith J. E.** Parameter Control in Evolutionary Algorithms // Parameter Setting in Evolutionary Algorithms. Springer Verlag, 2007. P. 19—46.
13. **Kim D. H., Abraham A., Cho J. H.** A hybrid genetic algorithm and bacterial foraging approach for global optimization // Information Sciences. 2007. V. 177. P. 3918—3937.
14. **El-Abd, Kamel M.** A taxonomy of cooperative search algorithms // Hybrid Metaheuristics: Second International Workshop. Springer, 2005. V. 3636. P. 32—41.
15. **Raidl G. R.** A Unified View on Hybrid Metaheuristics // Lecture Notes in Computer Science. Springer-Verlag, 2006. V. 4030. P. 1—12.

16. **Datta T., Misra I. S., Mangaraj B.B., Imtiaj S.** Improved adaptive bacteria foraging algorithm in optimization of antenna array for faster convergence // Progress In Electromagnetics Research. 2008. V. 1. P. 143—157.

17. **Korani W. M., Dorrah H. T., Emara H. M.** Bacterial Foraging Oriented by Particle Swarm Optimization Strategy for PID Tuning // Computational Intelligence in Robotics and Automation (CIRA). IEEE International Symposium on 15—18 Dec. 2009. P. 445—450.

18. **Yang Xin-She**. Firefly Algorithm, Stochastic Test Functions and Design optimization // International Journal of Bio-Inspired Computation. 2010. V. 2. N 2. P. 78—84.

19. **Krishnanand K. N., Ghose D.** Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions // Swarm Intelligence. 2009. V. 3. N 2. P. 87—124.

20. **Mehrabiana A.R., Lucasc C.** A novel numerical optimization algorithm inspired from weed colonization // Ecological informatics. 2006. № 1. P. 355—366.

21. **Yang X.-S., Deb S.** Cuckoo search via L'evy flights // Proc. of the world Congress on Nature & Biologically Inspired Computing (NaBIC 2009), December 2009, India. IEEE Publications, USA, pp. 210—214.

22. **Tuba M., Subotic M., Stanarevic N.** Modified cuckoo search algorithm for unconstrained optimization problems // Proc. of the 5th European Computing Conference (ECC'11), Paris, France, April 28—30, 2011, pp. 263—268.

23. **Mucherino A., Seref O.** Monkey Search: A Novel Meta-Heuristic Search for Global Optimization // Data Mining, System Analysis and Optimization in Biomedicine, AIP Conference Proceedings. 2007. P. 162—173.

24. **Zhao R., Tang W.** Monkey Algorithm for Global Numerical Optimization // Journal of Uncertain Systems. 2008. V. 2. N 3. P. 165—176.

25. **Abidin Z. Z., Ngah U. K., Arshad M. R., Ong B. P.** A novel Fly Optimization Algorithm for swarming application // Robotics Automation and Mechatronics (RAM). 2010 IEEE Conference on: 28—30 June 2010, Singapore P. 425—428.

26. **Abidin Z. Z., Arshad M. R., Ngah U. K.** A simulation based fly optimization algorithm for swarms of mini autonomous surface vehicles application // Indian Journal of Geo-Marine Sciences. 2011. V. 40 (2). P. 250—266.

27. **Eusuff M. M., Lansey K., Pasha F.** Shuffled Frog Leaping Algorithm: A Memetic Meta-Heuristic for Discrete Optimization // Engineering Optimization. 2006. V. 38. N 2. P. 129—154.

28. **Aghababa M. P., Akbari M. E., Shotorbani A. M.** An Efficient Modified Shuffled Frog Leaping Optimization Algorithm // International Journal of Computer Applications. 2011. V. 32. N 1. P. 26—30.

29. **Elbeltagi E., Hegazy T., Grierson D.** A Modified Shuffled Frog-Leaping Optimization Algorithm // Applications to Project Management. Structure and Infrastructure Engineering. 2007. V. 3. N 1. P. 53—60.

30. **Yang X.-S.** A New Metaheuristic Bat-Inspired Algorithm, in: Nature Inspired Cooperative Strategies for Optimization (NISCO 2010). Studies in Computational Intelligence. Berlin: Springer, 2010. Vol. 284. P.65-74.

31. **Bastos-Filho C.J.A., Lima-Neto F. B., Lins A., Nascimento A., Lima M.** Fish school search // Nature-Inspired Algorithms for Optimisation. SCI. Springer, Heidelberg, 2009. Vol. 193. P. 261—277.

32. **Cavalcanti-J'unior G. M., Bastos-Filho C. J. A., Lima-Neto F. B., Castro R. M. C. S.** A Hybrid Algorithm Based on Fish School Search and Particle Swarm Optimization for Dynamic Problems // International Conference on Swarm Intelligence 2011 (ICSI). V. 2. P. 543—552.

33. **Karci A.** Theory of Saplings Growing Up Algorithm // Proc. of the 8th international conference on Adaptive and Natural Computing Algorithms(ICANNGA '07). Springer-Verlag Berlin, Heidelberg, 2007. V. 1. P. 450—460.

34. **Lee KS., Geem ZW.** A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice // Computational Methods Applied Mechanics Engineering, 2005. V. 194. P. 3902—3933.

35. **Ingram G., Zhang T.** Overview of Applications and Developments in the Harmony Search Algorithm / Geem ZW. (Ed.): Music-Inspired Harmony Search Algorithm. Berlin: Springer-Verlag, 2009. P. 15—37.

36. **Mahdavi M., Fesanghary M., Damangir E.** An improved harmony search algorithm for solving optimization problems // Applied Mathematics and Computation. 2007. V. 188. P. 1567—1579.

37. **Coelho L. S., Bernert D. L. A.** A Harmony Search Approach Using Exponential Probability Distribution Applied to Fuzzy Logic Control Optimization // Recent Advances in Harmony Search Algorithm. Berlin: Springer-Verlag, 2010. P. 77—88.

38. **Chakraborty P., Roy G.G., Das S., Jain D., Abraham A.** An Improved Harmony Search Algorithm with Differential Mutation Operator // Fundamenta Informaticae, 2005. V. 95. P. 1—26.

39. **Omran M. G. H., Mahdavi M.** Global-best harmony search // Applied Mathematics and Computation. 2008. V. 198. N 2. P. 643—656.

40. **Rashedi E., Nezamabadi-pour H., Saryazdi S.** GSA: A Gravitational Search Algorithm // Information Sciences. 2009. V. 179. P. 2232—2248.

41. **Formato R. A.** Central Force Optimization: A new metaheuristic with applications in applied electromagnetics // Progress In Electromagnetics Research. 2007. V. 77. P. 425—491.

42. **Nobahari H., Nikusokhan M., Siarry P.** Non-dominated Sorting Gravitational Search Algorithm // ICSI 2011: International conference on swarm intelligence, Cergy, France. 2011. June 14—15. P. 1—10.

43. **Birbil I., Fang S. C.** An electro-magnetism-like mechanism for global optimization // Journal of Global Optimization. V. 25. P. 263—282.

44. **Debels D., DeReyck B., Leus R., Vanhoucke M.** A hybrid scatter search/electromagnetism meta-heuristic for project scheduling // European Journal of Operational Research. 2006. V. 169. P. 638—653.

45. **Rocha A. M. A. C., Fernandes E. M. G. P.** Modified movement force vector in a electromagnetism-like mechanism for global optimization // Optimization Methods and Software. 2008. P. 1-17.

46. **Kaveh A., Talatahari S.** A novel heuristic optimization method: charged system search // Acta Mechanica. 2010. V. 213. N 3—4. P. 267—289.

47. **Li N.** The Application of Sparse Antenna Array Synthesis Based on Improved Mind Evolutionary Algorithm // Intelligent Systems and Applications, 2011. N 3. P. 40—46.

48. **Jie J., Han Ch., Zeng J.** An Extended Mind Evolutionary Computation Model for Optimizations // Applied Mathematics and Computation, 2007. N 185 (2). P. 1038—1049.

49. **Лопатин А.С.** Метод отжига // Стохастическая оптимизация в информатике. 2005. Вып. 1. С. 133—149.

50. **Jie J., Zeng J.** Improved mind evolutionary computation for optimizations // Proc. of the 5th World Congress on Intelligent Control and Automation, WCICA 2004. Hangzhou, China. V. 3. P. 2200—2204.

51. **Lui J., Li N., Xie K.** Application of Chaos Mind Evolutionary Algorithm in Antenna Arrays Synthesis // Journal of computers. 2010. V. 5. N 5. P. 717—724.

52. **Nasuto S.J., Bishop J.M.** Convergence Analysis of Stochastic Diffusion Search // Parallel Algorithms and Applications. 1999. V. 1. N 2. P. 89—107.

53. **Reynolds R.G., Chung Ch.-J.** Function optimization using evolutionary programming with self-adaptive cultural algorithms // Lecture Notes on Artificial Intelligence. Springer-Verlag Press, 1997. P. 184—198.

54. **Reynolds R. G., Chung Ch.-J.** Knowledge-Based Self-Adaptation in Evolutionary Search // International Journal of Pattern Recognition and Artificial Intelligence. 2000. V. 14. N 1. P. 19—33.

55. **Kato K., Sakawa M., Katagiri H.** Revision of a Floating-Point Genetic Algorithm GENOCOP V for Nonlinear Programming Problems // The Open Cybernetics and Systemics Journal. 2008. V. 2. P. 24—29.

56. **Krasnogor N.** Studies on the Theory and Design Space of Memetic Algorithms, Ph.D. Thesis, Faculty of Computing, Mathematics and Engineering, University of the West of England, Bristol, U.K, 2002. 289 p.

57. **Ong Y. S., Lim M. H., Zhu N., Wong K.W.** Classification of adaptive memetic algorithms: A comparative study // IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 2006. P. 141—152.

58. **Nollea L., Zelinka I., Hopgooda A.A. Goodyear A.** Comparison of an self-organizing migration algorithm with simulated annealing and differential evolution for automated waveform tuning // Advances in Engineering Software. 2005. V. 36. P. 645—653.

59. **Glover F., Marti R.** Fundamentals of Scatter Search and Path Relinking // Control and Cybernetics. 2000. V. 29. N 3. P. 653—684.

60. **Glover F., Laguna M., Marti R.** Scatter Search and Path Relinking: Advances and Applications // International Series in Operations Research & Management Science. 2003. V. 57. P. 1—35.

61. **Glover F.** Tabu search — Part I // ORSA Journal on Computing. 1990. Vol. 1 (3). P. 190—206.

62. **Glover F.** Tabu search - Part II // ORSA Journal on Computing. 1990. V. 2 (1). P. 4—32.

63. **Blum C., Roli A.** Metaheuristics in combinatorial optimization: overview and conceptual comparison // ACM Computing Surveys. 2003. N 35 (3). P. 268—308.

64. **Eiben A. E., Michalewicz Z., Schoenauer M., Smith J. E.** Parameter Control in Evolutionary Algorithms // Parameter Setting in Evolutionary Algorithms. Springer Verlag, 2007. P. 19—46.

СВЕДЕНИЯ ОБ АВТОРЕ

КАРПЕНКО Анатолий Павлович — доктор физико-математических наук, профессор кафедры САПР МГТУ.

Имеет более 240 научных работ в области теории автоматического управления, математического моделирования, методов оптимизации, методов параллельных вычислений и автоматизации проектирования.

Основными областями научных интересов А. П. Карпенко являются методы решения однокритериальных и многокритериальные задач оптимизации, методы балансировки загрузки многопроцессорных вычислительных систем, методы управления многосекционными роботами-манипуляторами, методы обеспечения траекторной безопасности летальных аппаратов, нейросетевые алгоритмы, интеллектуальные обучающие системы.

Учредитель ООО "Издательство "Новые технологии"
107076, Москва, Стромынский пер., 4

Телефон редакции журнала **(499) 269-5510**

Художественный редактор *Т. Н. Погорелова.* Технический редактор *Е. В. Конова.*
Корректор *Т. В. Пчелкина.*

Сдано в набор 28.04.2012. Подписано в печать 22.06.2012. Формат 60×88 1/8. Бумага офсетная.
Усл. печ. л. 3,92. Уч.-изд. л. 4,35. Заказ IP712.

Журнал зарегистрирован в Государственном комитете Российской Федерации по печати.
Свидетельство о регистрации № 018866 от 27.05.99.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
105120, г. Москва, ул. Нижняя Сыромятническая, д. 5/7, стр. 2, офис 2.