НАУЧНОЕ ИЗДАНИЕ МГТУ ИМ. Н.Э. БАУМАНА
НАУКА и ОБРАЗОВАНИЕ
Эл № ФС 77 - 48211. Государственная регистрация №0421200025. ISSN 1994-0408
электронный научно-технический журнал

Гибридный метод глобальной оптимизации на основе искусственной иммунной системы

 # 08, август 2012
DOI: 10.7463/0812.0433381
Карпенко А. П., Шуров Д. Л.
УДК 519.6

Россия, МГТУ им. Н.Э. Баумана
apkarpenko@mail.ru
dmitry.shurov@mail.ru

Введение

Искусственные иммунные системы (ИИС) – это информационная технология, использующая понятия, аппарат и некоторые достижения теоретической иммунологии для решения прикладных задач. На основе ИИС разработано большое число различных методов для численного решения прикладных задач анализа данных, распознавания образов, компьютерной безопасности, диагностики неисправностей технических объектов, оптимизации и т.д.

Методы ИИС, ориентированные на решение задачи глобальной оптимизации, вдохновлены некоторыми аспектами поведения иммунной системы человека в процессе защиты ею организма от внешних факторов (патогенов и антигенов). Защитные клетки иммунной системы (антитела) при этом претерпевают множество изменений, целью которых является создание клеток, обеспечивающих наилучшую защиту от данного фактора. В результате создается большое число различных антител, и в борьбе с патогеном побеждают те из них, которые оказались наилучшим образом приспособлены для защиты. Иммунная система человека сохраняет в своей «памяти» победившие антитела, на основании чего именно такие антитела производятся при повторном проникновении в организм схожего патогена.

При разработке методов оптимизации на основе ИИС не ставится цель в точности воспроизвести детали функционирования живых иммунных систем. Авторы таких методов ограничиваются применением лишь некоторых принципов функционирования ИИС для конструирования механизмов оптимизации. Например, степень близости антител и антигенов формализуют в виде евклидовой нормы в пространстве варьируемых параметров, степень приспособленности антитела для борьбы с данным антигеном – в виде инвертированного значения целевой функции и т.д.

Основной целью публикации является представление оригинального гибридного метода глобальной оптимизации на основе ИИС. Метод получил наименование *SIA* (*Subplex Immune Algorithm*). В качестве базового метод использует известный метод *HIA* [1]. Модификация заключается, прежде всего, в отказе от процедуры мутации, как средства локального поиска. Вместо этого метод *SIA* использует для локального поиска алгоритм *SUBPLEX* [2], благодаря чему гарантируется, что до завершения полной проверки из популяции не будут исключены изначально кажущиеся плохими решения. Научная новизна работы заключается в новизне предложенного метода глобальной оптимизации.

Работа организована следующим образом.

В первом разделе дана постановка задачи глобальной оптимизации и общая схема иммунных методов ее решения.

Второй раздел содержит обзор значительного числа методов глобальной оптимизации, построенных на основе ИИС, в частности, в этом разделе дано описание базового метода *HIA*.

Третий раздел представляет разработанный метод *SIA*. Здесь же на основе сравнительного исследования эффективности алгоритмов локальной оптимизации, реализованных в известной программной библиотеке *NLopt* [3], обосновано использование алгоритма *SUBPLEX*.

Четвертый раздел посвящен разработке программного обеспечения, реализующего метод *SIA*. Реализация выполнена на языке программирования

общего назначения C++. Разработанное программное обеспечение *SIA Search* представляет собой многофункциональную расширяемую систему для тестирования и исследования эффективности методов оптимизации. В этом же разделе с использованием известных тестовых функций Розенброка, Растригина, и Химмельблау [4] выполнено тестирование *SIA Search*, показавшее корректность его функционирования.

В пятом разделе представлены результаты исследования эффективности алгоритма *SUBPLEX*, метода *SIA*, а также результаты сравнительного исследования эффективности методов *HIA*, *SIA* и методов глобальной оптимизации *CRS*, *MLSL*, *ISRES* из библиотеки *NLopt*.

В заключении сформулированы основные результаты работы и очерчены перспективы ее развития.

1. Постановка задачи и общая схема методов иммунной оптимизации

Рассматриваем задачу глобальной условной минимизации скалярной целевой функции $f(x)$ вещественных переменных x в области допустимых значений D:

$$\min_{x \in D \subseteq R^n} f(x) = f(x^\bullet) = f^\bullet. \qquad (1)$$

Здесь n - размерность вектора x. Общее число ограничений, определяющих область D, обозначаем N_D. Если далее не оговорено противное, то имеется в виду частный случай задачи (1) - задача безусловной оптимизации, когда $D = R^n$. Полагаем, что для этой задачи задана область поиска в виде параллелепипеда

$$\Pi = \{x \mid x_{\min} \le x_i \le x_{\max}, i \in [1:n]\},$$

где x_{min}, x_{max} - заданные константы.

В работе используем следующие основные понятия:

- *клетка*, *антитело* — решение $x \in R^n$;

- *аффинность* — величина, обозначающая полезность клетки и равная соответствующему значению целевой функции, взятому с противоположным знаком;
- *популяция* — множество клеток;
- *клон* — клетка, образованная из другой клетки путем ее полного копирования;
- *мутация* — случайное изменение компонентов вектора x.

Используем также следующие обозначения указанных и других сущностей:

- p — клетка;
- $f(p)$ — значение целевой функции, соответствующее данной клетке;
- $P=\{p_i, i \in [1:n_p]\}$ — популяция клеток;
- $p_{i,1}, p_{i,2},..., p_{i,n}$ — компоненты клетки $p_i \in P$;
- $\varphi_i = -f(p_i)$ — аффинность клетки p_i;
- M — популяция клеток памяти (не являющихся частью популяции P);
- C_i — множество клонов клетки p_i, созданных на текущей итерации метода;
- $C=\{C_i, i \in [1:n_p]\}$ — множество клонов всех клеток текущей популяции;
- $p_j^{C_i}$ — j-й клон клетки p_i;
- $p_{best}^{C_i}$ — лучший клон клетки p_i, то есть клон, которому соответствует минимальное значение целевой функции;
- n_c — число клонов, порождаемых каждой из клеток популяции P;
- f^* — минимальное значение целевой функции, найденное методом численной оптимизации;
- $\tilde{f}^*$ — среднее по мультистартам значение целевой функции, найденное этим методом;

- $\bar{n}_f$ — среднее по мультистартам число вычислений целевой функции (число испытаний).

Во введенных обозначениях основные особенности методов иммунной оптимизации можно сформулировать следующим образом.

1) Все методы иммунной оптимизации работают с популяцией клеток

$$P=\{p_i, i\in[1:n_p]\},$$

в которой клетка $p_i=(p_{i,1}, p_{i,2},..., p_{i,n})$ представляет собой вектор в пространстве варьируемых переменных.

2) Над клетками популяции осуществляют процедуры клонирования и мутации. При клонировании клетки p_i создают множество клеток C_i, идентичных исходной:

$$C_i=\left\{p_j^{C_i}:\ p_j^{C_i}=p_i,\ j\in[1:n_c]\right\}. \qquad (2)$$

При мутации компоненты вектора $p_i^{C_i}$ изменяют случайным образом по схеме

$$p_{j,k}^{Ci}=p_{j,k}^{C_i}+\Delta_{j,k},\ k\in[1:n], \qquad (3)$$

где $\Delta_{j,k}$ — шаг мутации, способ вычисления которого различен для различных методов иммунной оптимизации.

3) В результате процедуры клонального отбора лучшие из потомков замещают родителей по формуле

$$p_i=\begin{cases} p_i, & \text{если } f\left(p_{best}^{C_i}\right)> f\left(p_i\right),\\ p_{best}^{C_i}, & \text{если } f\left(p_{best}^{C_i}\right)\le f\left(p_i\right).\end{cases} \qquad (4)$$

4) Процедура сжатия популяции устраняет избыточность, исключая худшее из решений p_j, p_k, удовлетворяющих условию

$$\left\|p_j-p_k\right\|<b_r,\ \ j\neq k, \qquad (5)$$

где $\|\bullet\|$ - символ некоторой меры близости клеток, например, евклидова норма; b_r - порог сжатия.

5) Множество клеток памяти M хранит некоторое число наилучших решений, найденных за все время работы метода.

Процедуры клонирования и мутации обеспечивают локальную оптимизацию в окрестности каждой из клеток. Процедура сжатия популяции преследует цель ускорения сходимости метода и сокращения вычислительных затрат.

2. Обзор методов иммунной оптимизации

Последовательно рассматриваем методы *CLONALG*, *opt-AiNet*, *BCA*, *HIA*, *I-opt-AiNet* и, наконец, метод *T-Cell Model*.

2.1. Метод CLONALG

Метод *CLONALG* [5] был разработан для решения задач машинного обучения и распознавания образов. Позднее метод адаптировали для задач оптимизации. Основные принципы метода: наличие клеток памяти, отбор и клонирование наиболее полезных антител, исключение наименее полезных антител, выбор клонов пропорционально их полезности, обеспечение и сохранение разнообразия популяции антител.

Общая схема метода *CLONALG* имеет следующий вид [5].

1) Определяем полезность φ_i каждого из антител p_i множества P; $i \in [1 : n_p]$.

2) Во множестве P оставляем некоторое число антител с наибольшей полезностью, а остальные антитела исключаем.

3) Клонируем антитела из модифицированного множества P пропорционально их полезности и помещаем полученные клоны во множество C.

4) Множество клонов C подвергаем мутации, интенсивность α которой уменьшается с ростом полезности по формуле

$$\alpha = \exp(-\rho\, \varphi_i),$$

где ρ — положительный свободный параметр.

5) Определяем полезность всех мутировавших клонов множества C, и находим на этой основе клетку с наивысшей аффинностью. Если полезность данной клетки выше, чем у соответствующей ему клетки памяти, то заменяем последнюю указанным мутировавшим клоном.

6) Осуществляем замену d-ой части антител из множества P новыми клетками. Здесь d - свободный параметр метода.

2.2. Метод opt-AiNet

Главной особенностью метода *opt-AiNet* является сохранение всех найденных локальных и глобальных оптимумов целевой функции. Метод использует динамический размер популяции, управляемый механизмом сжатия [6].

Схему метода *opt-AiNet* передает следующая последовательность его основных шагов.

1) Инициализируем популяцию клеток.

2) Определяем полезность φ_i всех клеток популяции и нормализуем значениях этих величин так, чтобы нормализованная полезность $\hat{\varphi}_i$ принадлежала интервалу $[0;1]$; $i \in [1:n_p]$.

3) Порождаем n_c клонов для каждой из клеток.

4) Осуществляем мутацию каждого клона пропорционально полезности его родительской клетки по формуле

$$p_i' = p_i + \alpha_i N(0;1), \quad \alpha_i = \beta \exp(-\hat{\varphi}_i),$$

где p_i' — решение после мутации клетки p_i ; β — свободный положительный параметр метода.

5) Определяем полезность мутировавших клонов.

6) Для каждой родительской клетки и ее клонов выбираем наилучшее решение и исключаем из популяции остальные решения.

7) Вычисляем среднюю полезность популяции, и если она уменьшилась по сравнению с предыдущим шагом, то возвращаемся к шагу 2.

8) Определяем расстояние между всеми клетками популяции и

осуществляем сжатие тех клеток, расстояние между которыми ниже заданного порога b_r (см. формулу (5)). Оставшиеся клетки помещаем во множество клеток памяти.

9) Вводим в популяцию d -ю часть случайно сгенерированных клеток.

Здесь и далее $N(a;b)$ - нормальная случайная величина с математическим ожиданием a и среднеквадратическим отклонением b.

В работе [13] рекомендованы следующие значения свободных параметров метода: $b_r = 0{,}2$; $n_c = 10$; $d = 0{,}4$; $\beta = 100$. Начальный размер популяции n_p рекомендован равным 20. Гибридизация метода *opt-AiNet* с методом локального поиска рассмотрена в работе [7]. В этом случае к рассмотренной схеме метода добавляется этап, предполагающий локальную оптимизацию наилучшего решения.

2.3. Метод BCA (B-cell algorithm)

Метод *BCA*, как и метод *opt-AiNet*, основан на принципе клонального отбора [8]. Метод ориентирован на поиск глобального оптимума целевой функции, имеющий сложный ландшафт, и может обеспечить высокую эффективность поиска при небольших размерах начальной популяции $n_p = 3 \div 5$. Особенностью метода является использование оператора так называемой смежной гипермутации (*contiguous somatic hypermutation*), который предполагает мутацию не отдельных случайных компонентов клетки, а целой области из нескольких смежных (соседних по номерам) компонентов.

Схема метода *BCA* представлена ниже [8].

1) Инициализируем популяцию клеток.

2) Вычисляем полезность всех клеток популяции.

3) Для каждой клетки p_i порождаем набор клонов C_i, число клеток в котором равно начальному размеру популяции, то есть $n_c = n_p$; $i \in [1 : n_p]$.

4) Из каждого набора клонов C_i случайно выбираем одну клетку и осуществляем случайное изменение ее компонентов аналогично методу *CLONALG*.

5) Ко всем клонам применяем оператор гипермутации.

5) Вычисляем полезность всех мутировавших клонов. Для каждого набора клонов определяем клетку с наибольшей полезностью и, если ее полезность превышает полезность родительской клетки, заменяем последнюю первой.

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

2.4. Метод HIA

Метод *HIA* (*Hybrid Immune Algorithm*) [1] можно рассматривать как модификацию метода *opt-AiNet*. В отличие от последнего метода, размер популяции метода *HIA* остается постоянным в процессе всего итерационного процесса; каждая из клеток имеет дополнительный атрибут, называемый возрастом клетки; мутацию клонов осуществляют двумя способами - независимо для каждой компоненты клетки, либо только для одной ее компоненты. Метод *HIA* применяет стратегию, в соответствие с которой новые клетки создаются не на основе равномерного распределения по всей области поиска, как в методе *opt-AiNet*, а с использованием распределения Гаусса с динамически изменяющимися параметрами. Это приводит к постепенному сужению области поиска, но, благодаря второму типу мутации, не препятствует широкому исследованию пространства поиска.

В соответствии с работой [1] схему алгоритма *HIA* можно представить в следующем виде.

1) Инициализируем популяцию клеток.

2) Осуществляем клонирование, при котором каждая клетка популяции порождает одинаковое число n_c клонов.

3) Подвергаем клоны мутации.

4) Для каждой родительской клетки определяем лучший клон, и если полезность этого клона превосходит полезность родителя, заменяем родительскую клетку этим клоном. В противном случае возраст родителя увеличиваем на единицу.

5) Клетки, возраст которых достиг заданного значения перемещаем из основной популяции во множество клеток памяти (п.3).

6) Осуществляем сжатие клеток памяти и, если число этих клеток после сжатия превышает заданную величину, исключаем худшие из них.

7) Если не выполнен критерий окончания итераций, то переходим к шагу 2.

Первый способ мутации определяет формула

$$p'(t) = N\big(p(t), \alpha(t)\big),$$

где

$$\alpha(t) = \begin{cases} 2\left(p(t-1) - p'(t-1)\right), & f(p'(t-1)) < f(p(t-1)), \\ a(t-1), & \text{иначе.} \end{cases}$$

Второй способ мутации осуществляется в 20 % случаев и заключается в замене одной случайно выбранной компоненты $p_{i,s}$ родительской клетки p_i на случайное число, равномерно распределенное в заданном интервале $\left(x^s_{min}; x^s_{max}\right)$; $i \in [1:n_p]$, $s \in [1:n]$.

2.5. Метод I-opt-AiNet

Метод *I-opt-AiNet* представляет собой модификацию метода *opt-AiNet*, которая преследует цель уменьшить число испытаний в процессе поиска решения [9]. Изменения касаются определения числа выбираемых для клонирования решений и способа их выбора, числа удаляемых из популяции и добавляемых в нее решений, способа выбора решений для клонирования и числа порождаемых ими клонов, а также критерия остановки. Метод использует понятие лучшей средней полезности по всем предыдущим итерациям (*Best Fitness Average, BFA*), обозначаемой φ_{BFA}. Для того, чтобы предотвратить повторные ресурсоемкие вычисления полезности клеток, в процессе работы метода те клетки, значение полезности которых однажды уже было вычислено, помечают как «вычислены».

Схема метода *I-opt-AiNet* имеет следующий вид [9].

1) Инициализируем популяцию клеток.

2) Вычисляем полезность всех клеток популяции и эти клетки помечаем как «вычислены».

3) Из популяции исключаем клетки, полезность которых ниже величины φ_{BFA}.

4) Осуществляем клонирование клеток пропорционально их полезности, так что

$$n_{c_i} = \hat{\varphi}_i \, m_{\max}, \quad \hat{\varphi}_i = \frac{\varphi_{max} - \varphi_i}{\varphi_{max} - \varphi_{min}}; \quad i \in [1 : n_p].$$

Здесь $m_{\max}$ — максимальное число клонов, которое может быть порождено клеткой; $\varphi_{\max}$, $\varphi_{\min}$ — максимальное и минимальное текущие значения полезности среди всех клеток популяции соответственно; $\hat{\varphi}_i$ — нормированная полезность клонируемой клетки.

5) Осуществляем мутацию клонов по схеме метода *opt-AiNet*. Мутацию принимаем лишь в том случае, если результирующая клетка расположена в рассматриваемой части области поиска.

6) Вычисляем полезность всех клонированных клеток и помечаем их как «вычислены».

7) Из полученной популяции исключаем те клетки, полезность которых ниже φ_{BFA}.

8) В каждой паре клеток, расстояние между которыми меньше заданной величины b_r, исключаем клетку с меньшей полезностью.

9) В популяцию добавляем случайно сгенерированные клетки, число которых равно числу клеток, исключенных на шаге 3.

10) Если не выполнен критерий окончания итераций, то переходим к шагу 2.

В работе [9] рекомендованы следующие значения свободных параметров метода: максимальное число клонов каждой клетки $m_{\max} = 10$; порог сжатия $b_r = 0{,}2$; начальное число клеток $n_p = 10 \div 100$.

2.6. Метод T-Cell Model

В отличие от рассмотренных выше методов, метод *T-Cell Model* предназначен для решения задач глобальной условной оптимизации [10]. Метод использует три субпопуляции клеток - свободные клетки P_F, исполнительные клетки P_E и клетки памяти M. Свободные клетки обеспечивают разнообразие поиска и представляются двоичными строками в коде Грея. С помощью исполнительных клеток, также представляемых в коде Грея, исследуют части пространства поиска, лежащие вне области поиска, но недалеко от ее границы. Клетки памяти кодируют вещественными числами и используют для организации локального поиска в окрестностях лучших найденных решений.

Для того чтобы повысить вероятность локализации глобального экстремума целевой функции, метод *T-Cell Model* использует так называемый динамический допуск Δ_{DTF} (*Dynamic Tolerance Factor, DTF*). Величину Δ_{DTF} назначают для каждой из трех указанных выше субпопуляций по следующему правилу:

- вычисляем для каждой из субпопуляций величину σ_r, равную сумме значений ограничивающих функций, для которых клетками данной субпопуляции нарушено хотя бы одно из ограничений;

- полагаем $\sigma_r = \sigma_r / N_F$, $\sigma_r = \sigma_r / 3N_E$ для субпопуляции P_F, P_E соответственно, где N_F, N_E - числа клеток в этих популяциях;

- если для субпопуляции P_F имеет место неравенство $\sigma_r < 0{,}0001$, то принимаем для этой субпопуляции $\Delta_{DTF} = 0{,}1$;

- в той же ситуации для субпопуляции P_E принимаем $\Delta_{DTF} = 0{,}001$;

- для субпопуляции M во всех случаях принимаем $\Delta_{DTF} = const = 0{,}0001$.

Для субпопуляции P_F мутацию клеток не используют. Для клеток субпопуляций P_E, M, используют следующие законы мутации.

1.1) Вариант $p \in P_E$, $p \notin D$. Определяем величину наиболее нарушенного ограничения, и если она превышает σ_r / N_D , то каждый бит клетки *p* инвертируем с вероятностью $0{,}01 \div 0{,}2$. В противном случае по тому же правилу инвертируем биты только одной случайно выбранной компоненты клетки.

1.2) Вариант $p \in P_E$, $p \in D$. Каждый бит клетки *p* инвертируем с вероятностью $0{,}01 \div 0{,}2$.

2) Вариант $p \in M$. Мутацию клетки *p* производим по формуле

$$p' = p \pm \left(\frac{U(0;1)\left(x_{\max} - x_{min}\right)}{10^{\kappa}\ t\ N_D\ n} \right)^{U(0;1)} ,$$

где $U(0;1)$ — случайное число, равномерно распределенное в интервале $(0;1)$; κ — целое число (свободный параметр метода).

Важной в методе *T-Cell Model* является процедура отсеивания клеток, обеспечивающая сохранение лучших найденных решений. Для клеток, находящихся в области допустимых значений *D*, при отсеивании учитывают значения целевой функции, а для клеток вне этой области — величину σ_r . Любая из клеток, принадлежащих области *D*, считается лучше любой из клеток вне этой области.

Схема метода *T-Cell Model* состоит из следующих основных шагов.

1) Случайно генерируем заданное число свободных клеток.

2) Вычисляем значение динамического допуска Δ_{DTF} для популяции свободных клеток.

3) С учетом динамического допуска определяем, какие из свободных клеток принадлежат области *D*, а какие на принадлежат.

4) Заданный процент свободных клеток включаем в популяцию исполнительных клеток.

5) Вычисляем значение динамического допуска для популяции исполнительных клеток.

6) Повторяем 50 раз процедуру мутации исполнительных клеток и определяем, какие из исполнительных клеток лежат в области *D* с учетом их динамического допуска, а какие не лежат.

7) Заданный процент исполнительных клеток включаем в популяцию клеток памяти.

8) Повторяем 100 раз для клеток памяти действия, указанные в 6.

9) Описанные выше шаги выполняем заданное число раз.

2.7. Резюме

Во всех рассмотренных методах иммунной оптимизации задача интенсификации поиска (задача поиска локальных минимумов) решается с помощью операторов клонирования и мутации, которые на основе уже существующих в иммунной системе клеток продуцируют множество дочерних клеток. Задача диверсификации поиска (исследования пространства поиска) решается, преимущественно, путем введения в область поиска новых случайно сгенерированных решений. С целью сокращения вычислительных затрат, некоторые из рассмотренных методов имеют средства устранения близких решений. Наконец, все рассмотренные методы используют множество «клеток памяти», содержащее лучшие найденные решения.

Сравнительная характеристика рассмотренных методов представлена в таблице 1.

Вероятно, самой сильной стороной методов оптимизации на основе ИИС является наличие механизма памяти. Клетки памяти хранят информацию о найденных ранее наилучших решениях и позволяют использовать ее для повышения эффективности глобального поиска. Однако, на наш взгляд, рассмотренные методы не в полной мере используют потенциальные преимущества этого механизма, поскольку новые клетки в этих методах включаются в популяцию случайно по всей области поиска, независимо от состояния памяти. Единственным исключением в этом смысле является метод *HIA*, в котором с увеличением номера итерации информация о ландшафте целевой функции, хранящаяся в клетках памяти, все в большей степени влияет

на генерацию новых решений. Благодаря этому неперспективные области пространства параметров исключаются из рассмотрения, и поиск постепенно концентрируется в окрестности наилучших из найденных решений, что позволяет значительно повысить эффективность глобального поиска.

Таблица 1 — Сравнение методов иммунной оптимизации

Основные свойства	Метод оптимизации					
	CLONALG	*Opt-AiNet*	*BCA*	*HIA*	*I-opt-AiNet*	*T-Cell Model*
локальный поиск	мутация	мутация	мутация	два вида мутации	мутация	мутация
наличие клеток памяти	+	+	-	+	-	+
адаптивность	-	-	-	+	+	+
сжатие популяции	-	+	-	+	+	+

Другой важной отличительной особенностью рассмотренных методов является использование ими процедуры сжатия множества решений, которая устраняет клетки, расположенные в пространстве поиска слишком близко к другим клеткам. Такая процедура позволяет уменьшить избыточность популяции и, в результате, повысить скорость поиска. Поскольку размер популяции обычно невелик, вычислительные затраты на сжатие оказываются небольшими и оправдываются за счет уменьшения числа испытаний.

Главная слабость рассмотренных методов оптимизации состоит в низкой эффективности используемой ими процедуры локального поиска на основе мутации клеток. Эта процедура, по сути, представляет собой один из вариантов случайного поиска, эффективность которого, как хорошо известно, резко снижается с увеличением размерности задачи. Если даже начальное приближение сгенерировано в области притяжения глобального минимума, данная процедура может привести к потере этого решения, поскольку другие решения, находящиеся в областях притяжения локальных минимумов, могут иметь лучшие значения целевой функции.

3. Метод SIA

Разработка метода *SIA* имела целью преодолеть главный недостаток рассмотренных в предыдущем разделе методов иммунной оптимизации (низкую эффективность процедуры локального поиска) путем использования современного высокоэффективного алгоритма локальной оптимизации. Кроме того, метод *SIA* использует оригинальную адаптивную процедуру генерации начальных приближений, которая учитывает информацию о ранее найденных локальных решениях, хранящихся в клетках памяти.

Для выбора алгоритма локального поиска было проведено сравнительное тестирование алгоритмов, представленных в программной библиотеке *NLopt* [3]. В качестве тестовой функции использована овражная функция Розенброка (п.4), имеющая в точке минимума значение $f^{\bullet} = 0$. При тестировании был применен один из стандартных для библиотеки *NLopt* критерий окончания итераций: $|\Delta x_i \cdot x_i| < 10^{-4}$ для всех $i \in [1:n]$. Здесь Δx_i - текущий шаг по *i*-ой компоненте вектора варьируемых параметров; x_i - текущее значение этой компоненты. Использовано число мультистартов, равное 100. Рассмотрены три критерия эффективности алгоритма:

- минимальное достигнутое значение целевой функции f^*;
- среднее по мультистартам значение указанного критерия $\bar{f}^*$;
- среднее по мультистартам число испытаний $\bar{n}_f$.

Из результатов тестирования, представленых в таблицах 2 – 4, следует, что алгоритмы *NEWUOA*, *SUBPLEX* показывают примерно одинаково хорошие результаты. Однако поскольку алгоритм *NEWUOA* может иметь низкую эффективность в случае, когда вторая производная целевой функции не существует [3], для гибридизации нами был выбран алгоритм *SUBPLEX*.

Таблица 2 — Сравнение методов локального поиска по критерию f^*

n	f^*					
	COBYLA	BOBYQA	NEWUOA	PRAXIS	Nelder-Mead	SUBPLEX
5	0,0018550214	0,0000000027	0,0000000000	0,0000000000	0,0000000013	0,0000290945
10	-	0,0000003437	0,0000000000	0,0000000000	0,0000000033	0,0000213709
15	-	0,0000021825	0,0000000000	0,0000000001	0,0000000203	0,0000201872
20	-	0,0000012164	0,0000000000	0,0000000002	0,0000002865	0,0000196823
25	-	-	0,0000000000	0,0000000003	3,5218957616	0,0000202895
30	-	-	0,0000000000	0,0000000003	28,645998068	0,0000205485
40	-	-	0,0000000000	0,0000000006	178,22271606	0,0000201778
50	-	-	0,0000000000	0,0000000052	377,55999308	0,0000203529

Таблица 3 — Сравнение методов локального поиска по критерию $\bar{f}^*$

n	$\bar{f}^*$					
	COBYLA	BOBYQA	NEWUOA	PRAXIS	Nelder-Mead	SUBPLEX
5	3,8251309919	6,1604925340	0,7861683016	1122,8381968	42,639742522	1,2029188862
10	-	7,3035006557	0,9966453960	3950,7566576	594,36308318	1,0263534747
15	-	6,1797479486	0,9966561687	2218,9049581	1456,7683033	1,2378582278
20	-	1,4042984061	0,9169236137	9513,7499555	7684,8646762	1,1782738197
25	-	-	0,9567898536	12494,782606	12975,972222	0,7384074490
30	-	-	0,9966560931	9791,7841058	22780,663647	0,9086361899
40	-	-	1,1959871958	18631,417218	36132,595863	0,8660988796
50	-	-	0,9169236141	28088,890355	53856,724822	0,6491565486

Алгоритм *SUBPLEX (subspace-searching simplex)* предназначен для оптимизации сложных зашумленных недифференцируемых функций с большим числом варьируемых параметров. Метод разработан на основе симплекс-метода Нелдера-Мида [11] и использует его сильные стороны, устраняя недостатки. *SUBPLEX* осуществляет разбиение вектора варьируемых параметров x на подвекторы и применяет алгоритм Нелдера-Мида независимо к каждому из этих подвекторов. Таким образом, метод *SUBPLEX* осуществляет поиск в совокупности подпространств пространства варьируемых параметров R^n (см. ниже).

Таблица 4 — Сравнение методов локального поиска по критерию $\overline{n}_f$

n	$\overline{n}_f$					
	COBYLA	BOBYQA	NEWUOA	PRAXIS	Nelder-Mead	SUBPLEX
5	43416,7	603,4	419,2	587,7	687,1	158801,1
10	-	12986,9	1351,0	1889,4	4868,9	217677,0
15	-	11172,9	2619,6	13607,2	19739,1	566363,4
20	-	71910,0	3912,5	10596,2	39310,0	402161,0
25	-	-	5445,9	30484,8	58691,0	470694,0
30	-	-	7303,6	40116,9	90294,4	739809,0
40	-	-	11460,7	51475,8	168064,0	719550,1
50	-	-	17105,3	106220,4	272035,4	905081,6

Схема метода *SIA* имеет следующий вид.

1) Инициализируем популяцию с заданным числом клеток.

2) Методом *SUBPLEX* для каждой из клеток популяции выполняем локальный поиск и заменяем координаты этой клетки на координаты, полученные в результате локального поиска.

3) Перемещаем все клетки популяции во множество клеток памяти, исключаем их из популяции, и добавляем в популяцию новые клетки с координатами

$$p_{i,j} = N\left(p^M_{best,j};\ \sigma_i\right), \quad i \in [1:n_P], \quad j \in [1:n],$$

где

$$\sigma_i = \sigma_i(t) = \beta\sigma_i(t-1) + (1-\beta)\gamma_i;$$

$$\beta = \beta_0\left(1 - \left(1 - 1/t\right)^q\right);$$

$$\gamma_i = \max_{k,l}\left|p^M_{i,k} - p^M_{i,l}\right|, \quad k,l \in [1:n].$$

Здесь β - значение коэффициента обучения на текущей итерации t; p^M_j, p^M_k - клетки текущего множества памяти; β_0 и q — коэффициенты, задающие скорость обучения, рекомендованные значения которых равны $\beta_0 = 0{,}8;\ q = 5$ [1].

4) Осуществляем сжатие множества клеток памяти: для каждой пары этих клеток вычисляем расстояние между ними; если это расстояние меньше заданного порога b_r, то исключаем клетку с худшим значением целевой функции; если число клеток памяти после сжатия превышает величину m, то оставляем m лучших из них.

5) Если условия окончания итераций выполнены, то завершаем вычисления. В противном случае переходим к шагу 2.

В завершении раздела рассмотрим метод *SUBPLEX*. Метод использует следующие свободные параметры:

α - коэффициент отражения ($\alpha > 0$);

β - коэффициент расширения ($0 < \beta < 1$);

γ - коэффициент сужения ($\gamma > 1$);

δ - коэффициент сжатия ($0 < \delta < 1$);

ψ - коэффициент уменьшения симплекса ($0 < \psi < 1$);

ω - коэффициент уменьшения шага ($0 < \omega < 1$);

s_0 — начальное значение компонентов вектора шагов;

n_{min}, n_{max} - минимальная и максимальная размерности подпространств, подчиненные ограничениям

$$1 \le n_{min} \le n_{max} \le n, \quad n_{min} = ceil\left[n / n_{\max} \right] \le n,$$

где $cell\left[x\right]$ - ближайшее целое большее x.

В работе [2] рекомендованы следующие значения указанных параметров: $\alpha = 1$; $\beta = 0{,}5$; $\gamma = 2$; $\delta = 0{,}5$; $\psi = 0{,}25$; $\omega = 0{,}1$; $n_{min} = min\left(2, n\right)$; $n_{max} = min\left(5, n\right)$.

Последовательность основных шагов метода *SUBPLEX* имеет следующий вид.

1) Определяем величины компонентов вектора шагов.

2) Выделяем подпространства в пространстве поиска.

3) Для каждого из подпространств применяем классический метод

поиска Нелдера-Мида с так называемым внутренним критерием окончания итераций.

4) Проверяем выполнение внешнего условия окончания итераций. Если это условие не выполнено, то переходим к шагу 1. В противном случае завершаем вычисления.

Определение величины компонент вектора шагов. Компоненты этого вектора определяют величину и ориентацию начальных симплексов. На первой итерации все эти компоненты инициализируем значением s_0 . На последующих итерациях сначала масштабируем вектор шагов s по формуле

$$s(t) = \begin{cases} min\left(max\left(\|\Delta x\|_1 / \|s\|_1 , \omega \right), 1/\varpi \right) s(t-1), & \text{если } n_s > 1, \\ \psi\ s(t-1), & \text{если } n_s = 1, \end{cases}$$

где Δx — вектор приращения решения по сравнению с предыдущей итерацией; n_s — число подпространств, выделенных на предыдущей итерации; $\|x\|_1 = \sum_{i=1}^{n} |x_i|$. Затем корректируем компоненты вектора шагов по формуле

$$s_i = \begin{cases} sign\left(\Delta x_i\right) |s_i|, & \text{если } \Delta x_i \neq 0, \\ -s_i, & \text{если } \Delta x_i = 0. \end{cases}$$

Выделение подпространств в пространстве поиска. В результате выполнения этого этапа определяем число подпространств, размерность каждого из подпространств, а также координатные оси, задающие эти подпространства. Основная идея формирования подпространств состоит в том, чтобы направление вектора шагов находилось преимущественно в первом подпространстве. Число и размерности подпространств должны удовлетворять условиям

$$\sum_{i=1}^{n_s} n_i = n, \quad n_{min} \le n_i \le n_{max},$$

где n_i — размерность i -го подпространства.

Первым этапом определения подпространств является сортировка компонентов вектора шагов по убыванию их величин. Обозначаем исходный и отсортированный векторы шагов $\Delta x = (\Delta x_1, \ldots, \Delta x_n)$, $\Delta x' = (\Delta x_1', \ldots, \Delta x_n')$ соответственно, где $|\Delta x_i| \geq |\Delta x_{i+1}|,\ i \in [1:(n-1)]$.

На втором этапе определяем размерность n_1 первого подпространства как целое число $k \in [n_{min}; n_{max}]$, которое обеспечивает максимальное значение функции

$$r(k) = \begin{cases} \dfrac{\|(\Delta x_1' \ldots, \Delta x_k')\|_1}{k} - \dfrac{\|(\Delta x_{k+1}', \ldots, \Delta x_n')\|_1}{n-k}, & \text{если } k < n, \\ \dfrac{\|(\Delta x_1', \ldots, \Delta x_n')\|_1}{n}, & \text{если } k = n \end{cases}$$

при условии

$$n_{min} = ceil\left[{(n-k)} \big/ {n_{\max}} \right] \leq (n-k).$$

Выполнение этого условия гарантирует, что на основе оставшейся части вектора *x* также могут быть сформированы следующие подпространства.

По аналогичной схеме определяем n_2, n_3 и т.д.

Поиск в подпространстве выполняем с помощью классического метода Нелдера-Мида до тех пор, пока размер симплекса не уменьшится, по меньшей мере, в ψ раз по сравнению с его начальным размером (внутренний критерий останова). В качестве размера симплекса используем евклидово расстояние между наилучшей и наихудшей его вершинами.

Проверка внешнего условия окончания итераций. Локальный поиск по методу *SUBPLEX* останавливаем, когда выполнен внешний критерий остановки

$$\frac{max\left(\|\Delta x\|_\infty, \|s\|_\infty \psi\right)}{max\left(\|x\|_\infty, 1\right)} \leq \varepsilon,$$

где ε — заданная погрешность; $\|x\|_\infty = \max\limits_{i \in [1:n]} |x_i|$.

4. Программная реализация

При выборе программных средств для реализации метода *SIA* были рассмотрены следующие варианты:

- реализация на языке программирования общего назначения;

- реализация на специализированном языке одного из математических пакетов.

Последний вариант упрощает и ускоряет разработку вычислительных программ, но имеет несколько недостатков, главные из которых – это невозможность осуществления низкоуровневой оптимизации программы, зависимость от среды выполнения, возможное снижение скорости вычислений при интерпретации программы средой выполнения. Реализация на стандартном языке программирования общего назначения позволяет обеспечить максимальную производительность вычислений за счет оптимизации программы, не требует установки дорогостоящих программных пакетов и, что очень важно, обеспечивает возможность включения разработанной программы в состав другого программного обеспечения в виде библиотечной процедуры. На основе этих соображений было принято решение о реализации метода *SIA* на языке программирования C++, как одном из наиболее распространенных языков для решения подобных задач.

Разработанная программа, названная нами *SIA Search*, представляет собой расширяемую систему для тестирования и исследования эффективности методов оптимизации. Основными функциями *SIA Search* являются:

- задание множества значений свободных параметров метода, интересующих исследователя;

- автоматическое исследование эффективности метода в режиме мультистарта при всех заданных значениях свободных параметров;

- возможность введения в программу новой целевой функции без перекомпиляции программы;

- возможность расширения программы новыми методами оптимизации.

Поясним используемые в программе *SIA Search* правила, на основании которых задается указанное выше множество значений свободных параметров метода.

Пусть $B = (b_1, b_2, ..., b_\mu)$ - вектор интересующих исследователя свободных параметров рассматриваемого метода оптимизации, а

$$D_B = \{B \mid b_i \in [b_i^-; b_i^+], i \in [1:\mu]\} \text{ -}$$

- параллелепипед допустимых значений компонентов этого вектора.

Множество подлежащих исследованию значений свободных параметров должно быть задано в виде

$$\tilde{B} = (\tilde{B}_1 \otimes \tilde{B}_2 \otimes ... \otimes \tilde{B}_\mu),$$

где $\tilde{B}_i = \{b_{i,1}, b_{i,2}, ... b_{i,\nu_i}\}$ - набор допустимых значений свободного параметра b_i; ν_i - число значений этого параметра; $i \in [1:\mu]$.

Программа *SIA Search* для каждой из возможных комбинаций $(b_{1,j_1}, b_{2,j_2}, ..., b_{\mu,j_\mu}) \in \tilde{B}$ выполняет мультистарт с заданным числом запусков, для каждого запуска определяет значения указанных в п.3 критериев оптимальности метода, по завершении мультистарта находит средние значения этих критериев и представляет результаты исследования в виде таблицы.

Введение в программу *SIA Search* новой целевой функции осуществляется с помощью динамической библиотеки. Единственное условие для создания библиотеки - это наличие в операционной системе компилятора *GCC*.

4.1. Структура классов программы

Программа состоит из нескольких классов, структуру которых иллюстрирует рисунок 1. Сплошной линией на рисунке обозначено отношение наследования, а пунктиром - отношение «использует».

1) Класс **OptFunction** содержит информацию о целевой функции. Его конструктор принимает строковое имя целевой функции в виде экземпляра

вспомогательного класса **Name**, размерность целевой функции и указатель на функцию языка программирования Си, тип которой определен с помощью оператора `typedef` в виде

```
typedef double (*f)(unsigned int n, double* x).
```

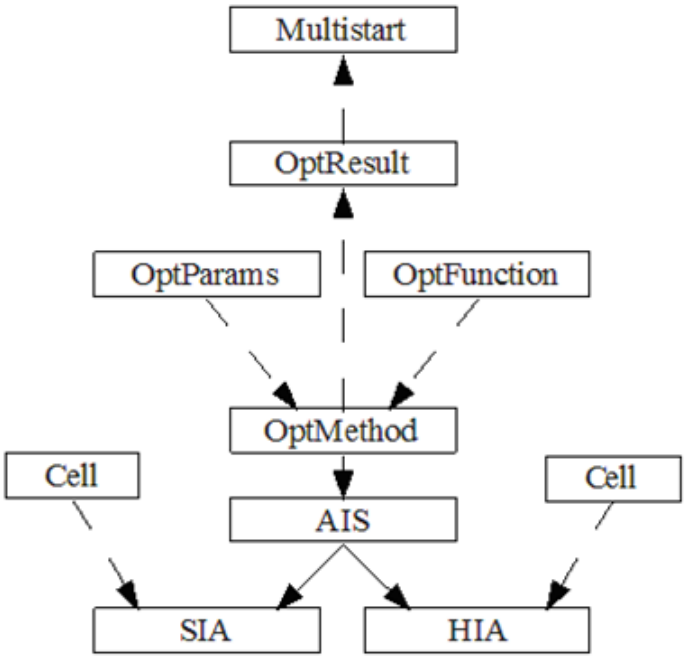


Рисунок 1 - Структура основных классов программы *SIA Search*

Класс **OptFunction** включает в себя следующие общедоступные функции:

- `double value(vector<double> x)` - вычисляет значение целевой функции в точке, заданной вектором `x`;
- `unsigned int dimension()` - возвращает размерность целевой функции;
- `Name name()` - возвращает имя целевой функции.

2) Класс **OptParams** осуществляет смену значений параметров, для которых осуществляется исследование эффективности метода оптимизации. Каждый параметр исследуемого метода представлен набором значений, заданных классом **OptParamVector**. Интерфейс класса **OptParamVector** составляют конструктор, осуществляющий функции добавления параметров,

и функция `step ()`, реализующая последовательный перебор заданных значений параметров. Задание множества параметров осуществляется с помощью функции

```
double get(Name name, vector<double> values).
```

Получение текущего значения параметра осуществляется обращением по его имени с помощью функции

```
double get(const char* name).
```

3) Класс **OptResult** предназначен для формирования результатов работы метода оптимизации и включает в себя следующие функции:

- `vector<double> coords()` - возвращает координаты наилучшего найденного решения;
- `double value()` - возвращает значение целевой функции, соответствующее наилучшему найденному решению;
- `unsigned int iterations()` - возвращает число итераций метода, затраченных на поиск полученного решения;
- `unsigned int evaluations()` - возвращает общее число вычислений целевой функции.

4) Виртуальный класс **OptMethod** является базовым классом для всех методов оптимизации. Его конструктор принимает класс целевой функции **OptFunction** и экземпляр класса **OptParams**, содержащий параметры используемого метода. Класс **OptMethod** имеет единственную общедоступную функцию `start ()`, которая запускает метод оптимизации и по его завершении возвращает экземпляр класса **OptResult** с результатами оптимизации.

Класс **OptMethod** имеет защищенные функции, обязательные для определения в производных классах:

- `void setParams(OptParams p);`
- `void init();`
- `void iterate();`

- `bool stop();`
- `OptResult bestResult()`.

Функция `setParams()` предназначена для получения параметров метода по их строковым идентификаторам и инициализации внутренних переменных класса, соответствующих этим параметрам.

Функция `init()` предназначена для задания начальных условий, требуемых для работы метода. Данная функция вызывается один раз для данного набора параметров.

Функция `iterate()` осуществляет выполнение основной итерации рассматриваемого метода оптимизации. Реализация данной функции полностью зависит от этого метода.

Функция `stop()` проверяет критерий окончания итераций (стагнация в течение заданного числа итераций либо достижение заданного числа вычислений целевой функции).

Функция `bestResult()` возвращает лучший найденный на текущей итерации результат.

5) Класс **Multistart** предназначен для сбора, обобщения и вывода результатов исследования. В его задачи входит подсчет числа проведенных запусков исследуемого метода оптимизации, хранение результатов оптимизации по каждому запуску, осреднение этих результатов по всем проведенным стартам, вывод в файл в виде таблицы осредненных результатов и лучшего результата по всему мультистарту.

6) Каждая клетка искусственной иммунной системы является экземпляром класса **Cell**, который имеет функции, осуществляющие клонирование и мутацию клеток, вычисление расстояния между клетками, а также получение вектора координат клетки.

7) Класс **AIS** включает в себя функции, реализующие процедуры, специфические для методов иммунной оптимизации:

- `Cell randomCell()` - создает клетку, координаты которой генерируются случайно в пределах области поиска;

- `vector<Cell> clone(Cell& A, unsigned int Nc)` – порождает множество клонов заданной клетки и возвращает массив, содержащий эти клоны;
- `bool stagnation()` - определяет наличие ситуации стагнации, когда отсутствует улучшение значения целевой функции для лучшей из клеток памяти в течение заданного числа итераций;
- `OptResult bestResult()` — возвращает экземпляр класса **OptResult**, содержащий информацию о наилучшей из клеток памяти;
- `void replace(unsigned int i, const Cell& cell)` — удаляет клетку p_i популяции и помещает на ее место клетку, заданную параметром `cell`;
- `void selectBest(vector<Cell>& P, unsigned int N)` – осуществляет сортировку популяции клеток по возрастанию значения целевой функции и сокращает популяцию до указанного размера;
- `void suppress(vector<Cell>& P, double D)` – осуществляет процедуру сжатия популяции с порогом сжатия D.

8) Класс **SIA** является основным классом, реализующим метод *SIA*. В данном классе определена функция `iterate()`, выполняющая все шаги метода; функция `generateNew()`, создающая новые клетки популяции на основе состояния клеток памяти; вспомогательная функция `maxDeltaCoords(unsigned int coord)`, определяющая максимальную разность координат клеток памяти.

9) Класс **HIA** был разработан для реализации сравнительного исследования эффективности метода *SIA*. Функции класса **HIA** близки к рассмотренным функциям класса **SIA**. Наиболее существенное отличие заключается в наличии функции `mutate(vector<Cell>& P, Cell& p)`, которая осуществляет локальный поиск с помощью указанных в п.2.4 видов мутации.

4.2. Тестирование программы

Тестирование метода и программы *SIA Search* выполнено на функциях Розенброка, Химмельблау и Растригина [4].

Функция Розенброка является унимодальной, но овражной, и имеет вид

$$f(x)=\sum_{i=1}^{n-1}\left[100\left(x_{i+1}-x_i^2\right)^2+(1-x_i)^2\right].$$

Область поиска ограничиваем гиперкубом Π, размеры которого равны $x_{\min}=-2{,}048,\ x_{\max}=2{,}048$. В этой области минимум $f^{\bullet}=0$ функция достигает в точке $x^{\bullet}=(1;1;...;1)$.

Функция Химмельблау

$$f(x_1,x_2)=\left(x_1^2+x_2-11\right)^2+\left(x_1+x_2^2-7\right)^2$$

двумерна и имеет четыре локальных минимума, являющихся одновременно глобальными минимумами:

$$f\left(x^{\bullet}\right)=\ f(3{,}0;\ 2{,}0)=0{,}0;$$

$$f\left(x^{\bullet}\right)=\ f(-2{,}8051;\ 3{,}1313)=0{,}0;$$

$$f\left(x^{\bullet}\right)=\ f(-3{,}7793;\ -3{,}2832)=0{,}0;$$

$$f\left(x^{\bullet}\right)=\ f(3{,}5844;\ -1{,}8481)=0{,}0.$$

Важно, что все точки минимума функции, кроме первой, имеют иррациональные значения координат.

Функция Растригина по каждому из измерений представляет собой сумму квадратичной и косинусоидальной составляющих, благодаря чему функция имеет большое число регулярно расположенных локальных минимумов. Функцию определяет формула

$$f(x)=10n+\sum_{i=1}^{n}\left[x_i^2-10\cos\left(2\pi x_i\right)\right].$$

Область поиска ограничивает гиперкуб, границы которого задают константы $x_{\min}=-5{,}12,\ x_{\max}=5{,}12$. Глобальный минимум $f^{\bullet}=0$ функция достигает в точке $x^{\bullet}=(0;\ 0;...;\ 0)$.

Далее нам понадобятся также тестовые функции Швефеля и Шекеля.

Функция Швефеля [4] является многоэкстремальной и обладает так называемым ложным глобальным минимумом. Многие методы поиска в этой ситуации склонны сходиться к последнему. Функцию Швефеля определяет формула

$$f(x) = \sum_{i=1}^{n} \left[-x_i \cdot \sin\left(\sqrt{|x_i|}\right) \right].$$

Поиск осуществляем в гиперкубе $П$, в котором $x_{\min} = -2{,}048$, $x_{\max} = 2{,}048$. Глобальный минимум функции в этом случае равен $f^{\bullet} = -418{,}9829\,n$ и достигается в точке $x^{\bullet} = (420{,}9687;\ 420{,}9687;,,,,;\ 420{,}9687)$.

Функция Шекеля [4] замечательна тем, что положение и глубину всех минимумов этой функции задает сам пользователь. Функция имеет вид

$$f(x) = -\sum_{i=1}^{k} \frac{1}{c_i + \sum_{j=1}^{n} \left(x_j - a_{i,j}\right)^2},$$

где k - число минимумов; $a_i = (a_{i,1}, a_{i,2}, ..., a_{i,n})$ - вектор, определяющий координаты i–го минимума; c_i - константа, задающая значение этого минимума.

Результаты тестирования программы *SIA Search* для двумерных вариантов функций Розенброка, Химмельблау и Растригина представлены в таблице 5. В верхних подстроках таблицы приведены значения, полученные с помощью метода *SIA*, а в нижних подстроках - аналитические значения. Таблица показывает корректность метода и его программной реализации.

Таблица 5 — Результаты тестирования метода *SIA*: $n = 2$

Функция	$\frac{f^*}{f^\bullet}$	$\frac{x^*}{x^\bullet}$
Розенброка	0,0000000214	(1,00004; 1,00006)
	0	(1; 1)
Химмельблау	0,0000000573	(3,58440; -1,84815)
	0	(3,584; -1,848)
	0,0000000777	(-2,80513; 3,13127)
	0	(-2,805; 3,131)
	0,0000000191	(2,99998; 2,00003)
	0	(3;2)
	0,0000000400	(-3,77931; -3,28315)
	0	(-3,779; -3,283)
Растригина	0,000000122	(-0,00002; 0,00001)
	0	(0; 0)

5. Исследование эффективности

В данном разделе представлены некоторые результаты исследования эффективности алгоритма *SUBPLEX*, метода *SIA*, а также результаты сравнительного исследования эффективности методов *HIA*, *SIA* и алгоритмов глобальной оптимизации *CRS*, *MLSL*, *ISRES*, представленных в программной библиотеке *NLopt*.

5.1. Алгоритм SUBPLEX.

Результаты исследования эффективности алгоритма *SUBPLEX* представлены в таблице 6. Исследование выполнено для функции Розенброка, имеющей размерность от 50 до 800. Во всех случаях использован мультистарт из 30 запусков. Таблица показывает, что для всех размерностей каждый из запусков обеспечил высокую точность локализации минимального значения функции. Очень важно, что при этом наблюдается почти линейная зависимость числа испытаний n_f от числа варьируемых параметров n.

Таблица 6 — Исследование эффективности алгоритма *SUBPLEX*: функция Розенброка

n	50	100	200	400	800
f^*	0,0000203529	0,0000206184	0,0000204257	0,0000207587	0,0000206517
$\tilde{f}^*$	0,6491565486	0,7223015334	0,8812430192	0,8390497163	0,7193811945
$\bar{n}_f$	905081	2047319	4364204	8683773	16748447

5.2. Метод SIA.

Прежде было выполнено исследование эффективности метода при поиске глобального минимума функции Растригина размерностью 12, 16, 20, 24, 28, а также смещенной функции Растригина тех же размерностей. Область поиска, как и ранее, была ограничена гиперкубом $\Pi = (x_i \mid x_{\min} \le x_i \le x_{\max}, i \in [1:n])$, где $x_{\min} = -5{,}12$, $x_{\max} = 5{,}12$. Глобальный минимум последней функции в этой области расположен в точке

$$x^{\bullet} = \left(\frac{5}{1}; \frac{5}{2}; \frac{5}{3}; \ldots; \frac{5}{n}\right).$$

Во всех случаях использован мультистарт из 100 запусков.

Результаты исследования представлены в таблице 7, которая показывает, что метод успешно находит точный глобальный минимум функции Растригина, имеющей размерность, не превышающую 20. При более высоких размерностях, когда число локальных минимумов становится чрезвычайно большим, метод находит локальные минимумы, близкие к точке глобального минимума.

Таблица 7 — Результаты тестирования метода *SIA*:
функция Растригина; $m_{\max} = 80$

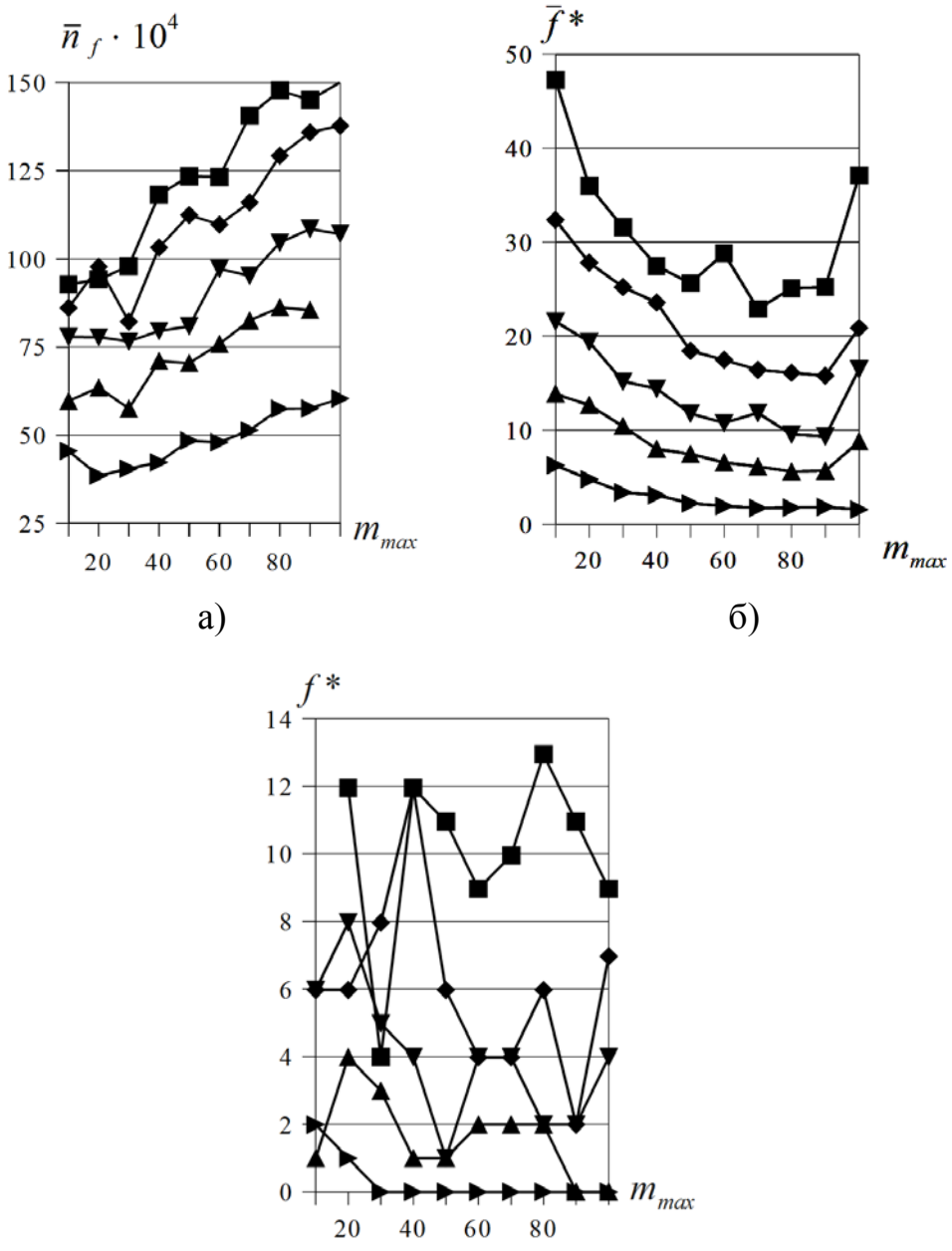
Функция	n	12	16	20	24	28
Растригина	$f^{\bullet}$	0,000000000	0,000000000	0,000000000	0,994959027	1,989918055
	$\tilde{f}^{*}$	0,527328284	1,890421901	3,472405898	6,626423396	9,173515079
Растригина (смещенная)	$f^{\bullet}$	0,000000001	0,000000004	0,000000007	0,000000009	1,989918075
	$\tilde{f}^{*}$	2,616741867	5,780709701	9,133717740	13,33243873	15,44174748

Параметрическое исследование метода *SIA* выполнено для функций Растригина и Шекеля, имеющих размерности 16, 24, 30, 36, 42. В качестве варьируемых параметров рассмотрены порог сжатия b_r, максимальное число клеток памяти $m_{max} \in \{10; 20; 40\}$, коэффициент обучения $\beta_0 \in \{0{,}7; 0{,}8; 0{,}9\}$, коэффициент $q \in \{4; 5; 6\}$. Использован фиксированный размер популяции $n_P = 5$. В каждом исследовании варьировался только один из рассматриваемых свободных параметров, а остальные параметры принимали свои «базовые» значения $b_r = 0{,}04$; $\beta_0 = 0{,}8$; $q = 5$; $m_{max} = 10$ для функции Шекеля и $m_{max} = 100$ для функции Растригина. Число мультистартов принято равным 100. В качестве критерия окончания итераций во всех случаях использована стагнация в течение 20 итераций.

Результаты исследования показали, что только параметр m_{max} оказывает существенное влияние на эффективность метода. На этом основании приводим зависимости рассматриваемых критериев оптимальности только от этого параметра (рисунок 2).

Результаты исследования показывают, что среднее по мультистарту значение целевой функции $\bar{f}^{*}$ уменьшается (улучшается) с ростом числа клеток памяти m_{max} до величины, примерно равной 90, которая слабо зависит от размерности этой функции (рисунок 2б). С другой стороны, с увеличением

m_{max} увеличивается среднее число испытаний $\bar{n}_f$. Таким образом, как и следовало ожидать, увеличение значений параметра m_{max} повышает качество поиска ценой увеличения числа испытаний.



► - $n = 16$, ▲ - $n = 24$, ▼ - $n = 30$, ♦ - $n = 36$, ■ - $n = 42$

Рисунок 2 - Эффективность метода *SIA*:
функция Растригина; варьируемый параметр m_{max}

Для функции Шекеля зависимость эффективности метода *SIA* от числа клеток памяти имела иной характер. Наилучшие результаты для всех указанных выше размерностей целевой функции были достигнуты в этом случае при $m_{max} = 10$, а для всех остальных m_{max} были получены примерно одинаковые результаты.

Разный характер зависимости критериев эффективности от параметра m_{max} для функции Растригина и Шекеля объясняется существенным различием ландшафтов этих функций.

5.4. Методы SIA, HIA, CRS, MLSL, ISRES

Сравнительное исследование эффективности указанных методов выполнено для функций Растригина и Шекеля, имеющих размерность 16, 24, 32, 40, 50. Для функции Растригина использована область поиска, указанная в п. 5.3. В случае функции Шекеля число минимумов k принято равным размерности целевой функции ($k = n$); положения минимумов определяют константы

$$a_{i,j} = \frac{i}{3} - \frac{j}{7},\ i \in [1:k],\ j \in [1:n];$$

глубину минимумов задают величины $c_i = \frac{i}{5},\ i \in [1:k]$. Область поиска для функции Шекеля ограничена гиперкубом со стороной, равной 10, и центром в начале координат:

$$\Pi = \left\{ x \mid x_i \in \left[x_{\min}; x_{\max} \right], i \in [1:n] \right\},\quad x_{\min} = -5,\quad x_{\max} = 5.$$

Для метода *SIA* использованы следующие значения свободных параметров:

$m_{max} = 10$ для функции Шекеля и $m_{max} = 100$ для функции Растригина;

$$\sigma_s = 0{,}04;\ n_P = 5;\ \beta_0 = 0{,}8;\ q = 5.$$

Аналогично для метода *HIA*:

$$m_{max} = 10;\ n_c = 1500;\ n_p = 5;\ b_r = 0{,}02;\ a_{max} = 5;\ \beta_0 = 0{,}8;\ q = 5.$$

В качестве значений свободных параметров методов *CRS*, *MLSL*, *ISRES* использованы их умолчательные значения, предлагаемые библиотекой *NLopt* [3].

Применен отличный от использованных ранее критерий окончания итераций - достижение заданного числа вычислений целевой функции, равного

$$n_f^{max} = 50000\, n\,.$$

Результаты исследования представлены в таблицах 8, 9.

Таблица 8 - Сравнительная эффективность методов *SIA*, *HIA*, *CRS*, *MLSL*, *ISRES*: функции Растригина

n	Метод									
	SIA		HIA		CRS		MLSL		ISRES	
	f^*	$\overline{f}^*$	f^*	$\overline{f}^*$	f^*	$\overline{f}^*$	f^*	$\overline{f}^*$	f^*	$\overline{f}^*$
16	0,000	1,333	0,002	0,032	0,994	11,332	4,974	16,371	2,984	12,616
24	0,000	5,571	0,003	0,104	3,979	16,755	14,924	39,052	11,939	21,958
32	1,989	12,297	0,033	0,201	4,974	21,182	36,813	67,428	17,909	33,490
40	2,984	19,093	0,070	0,356	2,984	19,859	66,662	99,057	23,879	43,101
50	8,954	29,599	0,131	0,569	4,974	26,137	68,651	136,975	32,833	57,021

Таблица 9 - Сравнительная эффективность методов *SIA*, *HIA*, *CRS*, *MLSL*, *ISRES*: функции Шекеля

n	Метод									
	SIA		HIA		CRS		MLSL		ISRES	
	f^*	$\overline{f}^*$	f^*	$\overline{f}^*$	f^*	$\overline{f}^*$	f^*	$\overline{f}^*$	f^*	$\overline{f}^*$
16	-3,729	-2,815	-3,724	-2,501	-5,774	-3,935	-3,729	-2,669	-3,729	-2,361
24	-3,370	-2,619	-2,582	-1,823	-3,370	-1,539	-2,586	-2,119	-1,723	-1,323
32	-2,383	-2,113	-1,970	-1,451	-1,410	-1,251	-1,973	-1,668	-1,226	-1,039
40	-2,254	-1,918	-2,232	-1,213	-1,119	-1,062	-1,594	-1,330	-1,119	-0,944
50	-1,303	-1,303	-1,191	-0,956	-0,963	-0,891	-1,195	-1,046	-1,195	-0,870

Из таблиц 8, 9 следует, что предложенный метод глобальной оптимизации *SIA* показал очень хорошие результаты на тестовой функции Шекеля, найдя наилучшее решение среди всех рассмотренных методов. Для функции Растригина по всем критериям метод показал третий результат.

Сильное влияние на эффективность метода оказывает свободный параметр m_{max} . Для функции Растригина оптимальное значение этого параметра равно 80, а для функции Шекеля - 10.

Заключение

В работе предложен оригинальный гибридный метод глобальной оптимизации *SIA*, основанный на технологии искусственных иммунных систем. Метод представляет собой глубокую модификацию известного гибридного алгоритма *HIA*, который для локального поиска использует процедуру мутации. В методе *SIA* для этой цели применен алгоритм *SUBPLEX*, реализованный в известной программной библиотеке *NLopt*.

Разработано программное обеспечение, реализующее метод *SIA*, которое представляет собой многофункциональную расширяемую систему для тестирования и исследования эффективности методов оптимизации.

На тестовых функциях Розенброка, Растригина, и Химмельблау, Швефеля и Шекеля выполнено исследование эффективности алгоритма *SUBPLEX*, метода *SIA*, а также сравнительное исследование эффективности методов *HIA*, *SIA* и алгоритмов глобальной оптимизации *CRS*, *MLSL*, *ISRES* библиотеки *NLopt*. Результаты исследования показывают высокую эффективность метода *SIA* для целевых функций, имеющих размерность, не превышающую 50, что является достаточным для широкого круга практически важных задач оптимизации.

Для повышения эффективности метода *SIA* при поиске минимума функции, характер ландшафта которой неизвестен (что является типичной для приложений ситуацией), предполагается модификация метода на основе самоадаптации его ключевых свободных параметров [12]. Предполагается

также разработка параллельных вариантов метода для кластерных вычислительных систем, а также для графических процессорных устройств.

Литература

1. Lucinska M., Wierzchon S.T. Hybrid Immune Algorithm for Multimodal Function Optimization // Recent Advances in Intelligent Information Systems, 2009, pp. 301-313. (http://iis.ipipan.waw.pl/2009/proceedings/iis09-30.pdf).

2. Rowan T.H. Functional Stability Analysis of Numerical Algorithms, Ph.D. Thesis, Department of Computer Sciences, University of Texas at Austin, 1990. (http://reference.kfupm.edu.sa/content/f/u/functional_stability_analysis_of_numeric_1308737.pdf).

3. The NLopt nonlinear-optimization package.
(http://ab-initio.mit.edu/wiki/index.php/NLopt).

4. Molga M., Smutnicki C. Test functions for optimization needs. 2005. (http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf).

5. De Castro L.N., Von Zuben F.J. Learning and Optimization Using the Clonal Selection Principle // IEEE Transactions on Evolutionary Computation, 2002, vol. 6, No. 3, pp. 231-259.
(http://www.idi.ntnu.no/emner/tdt4/2010%20articles/ clonalselection-13.10.pdf).

6. De Castro L.N., Timmis J. An Artificial Immune Network for Multimodal Function Optimization // Proceedings of IEEE Congress on Evolutionary Computation (CEC'02), 2002, May, Hawaii, vol. 1, pp. 699-674.
(http://neuro.bstu.by/ai/To-dom/My_research/Papers-0/Etc-done/wais02.pdf).

7. Yildiz A.R. A novel hybrid immune algorithm for global optimization in design and manufacturing // Robotics and Computer-Integrated Manufacturing, 2009, vol. 25, pp. 261-270.

8. Kelsey J., Timmis J. Immune Inspired Somatic Contiguous Hypermutation for Function Optimisation, 2003.
(http://www.cs.york.ac.uk/rts/docs/ GECCO_2003/papers/2723/ 27230207.pdf).

9. Agiza H.N., Hassan A.E., Salah A.M. An Improved Version of opt-AiNet Algorithm (I-opt-AiNet) for Function Optimization // IJCSNS International Journal of Computer Science and Network Security, March 2011, vol. 11, No. 3, pp. 80-85.

(http://paper.ijcsns.org/07_book/201103/20110313.pdf).

10. Aragon V.S., Escuivel S.C., Coello Coello C.A. Solving Constrained Optimization using a T-Cell Artificial Immune System, 2008.
(http://polar.lsi.uned.es/revista/index.php/ia/article/viewFile/579/563).

11. Nelder J. A., Mead R. A simplex method for function minimization // The Computer Journal, 1965, Vol. **7**, p. 308-313.

12. Eiben A. E., Michalewicz Z., Schoenauer M., Smith J. E. Parameter Control in Evolutionary Algorithms / Parameter Setting in Evolutionary Algorithms.- Springer Verlag, 2007, pp. 19-46.

SCIENTIFIC PERIODICAL OF THE BAUMAN MSTU
SCIENCE and EDUCATION
EL № FS 77 - 48211. №0421200025. ISSN 1994-0408
electronic scientific and technical journal

A hybrid global optimization method based on artificial immune system

08, August 2012
DOI: 10.7463/0812.0433381
Karpenko A.,P., Shurov D.L.

Russia, Bauman Moscow State Technical University
apkarpenko@mail.ru
dmitry.shurov@mail.ru

The paper presents an original hybrid method of global optimization – SIA – which based on the artificial immune systems technology. The method represents extreme modification of the hybrid optimization method HIA (Hybrid Immune Algorithm) which uses mutation procedure for local search. On the contrary, in SIA method for this purpose the known SUBPLEX algorithm is used. The authors give a description of this method and its program implementation. In addition, the authors investigate efficiency of the proposed method and present the results of this research.

Publications with keywords: global optimization, hybridization, immune algorithm, SUBPLEX
Publications with words: global optimization, hybridization, immune algorithm, SUBPLEX

References

1. Lucinska M., Wierzchon S.T. Hybrid Immune Algorithm for Multimodal Function Optimization. In: *Recent Advances in Intelligent Information Systems*, 2009, pp. 301-313. Available at: http://iis.ipipan.waw.pl/2009/proceedings/iis09-30.pdf .

2. Rowan T.H. *Functional Stability Analysis of Numerical Algorithms. Ph.D. Thesis*. Department of Computer Sciences, University of Texas at Austin, 1990. Available at: http://reference.kfupm.edu.sa/content/f/u/functional_stability_analysis_of_numeric_1308737.pdf .

3. *The NLopt nonlinear-optimization package*. Available at: http://ab-initio.mit.edu/wiki/index.php/NLopt .

4. Molga M., Smutnicki C. *Test functions for optimization needs*, 2005. Available at: http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf .

5. De Castro L.N., Von Zuben F.J. Learning and Optimization Using the Clonal Selection Principle. *IEEE Transactions on Evolutionary Computation*, 2002, vol. 6, no. 3, pp. 231-259. Available at: http://www.idi.ntnu.no/emner/tdt4/2010%20articles/clonalselection-13.10.pdf .

6. De Castro L.N., Timmis J. An Artificial Immune Network for Multimodal Function Optimization. *Proc. of IEEE Congress on Evolutionary Computation (CEC'02)*, May 2002, Hawaii, vol. 1, pp. 699-674. Available at: http://neuro.bstu.by/ai/To-dom/My_research/Papers-0/Etc-done/wais02.pdf .

7. Yildiz A.R. A novel hybrid immune algorithm for global optimization in design and manufacturing. *Robotics and Computer-Integrated Manufacturing*, 2009, vol. 25, pp. 261-270.

8. Kelsey J., Timmis J. *Immune Inspired Somatic Contiguous Hypermutation for Function Optimisation*, 2003.

Available at: http://www.cs.york.ac.uk/rts/docs/ GECCO_2003/papers/2723/ 27230207.pdf .

9. Agiza H.N., Hassan A.E., Salah A.M. An Improved Version of opt-AiNet Algorithm (I-opt-AiNet) for Function Optimization. *IJCSNS International Journal of Computer Science and Network Security*, March 2011, vol. 11, no. 3, pp. 80-85. Available at: http://paper.ijcsns.org/07_book/201103/20110313.pdf .

10. Aragon V.S., Escuivel S.C., Coello Coello C.A. *Solving Constrained Optimization using a T-Cell Artificial Immune System*, 2008.

Available at: http://polar.lsi.uned.es/revista/index.php/ia/article/viewFile/579/563.

11. Nelder J. A., Mead R. A simplex method for function minimization. *The Computer Journal*, 1965, vol. **7**, pp. 308-313.

12. Eiben A. E., Michalewicz Z., Schoenauer M., Smith J. E. Parameter Control in Evolutionary Algorithms. In: *Parameter Setting in Evolutionary Algorithms*. Springer Verlag, 2007, pp. 19-46.