Детализация онтологической модели по роевым алгоритмам, основанным на поведении насекомых и животных

В.В. Баранюк, О.С. Смирнова

Аннотация **– В статье представлено описание веток (листьев) дерева онтологической модели бионических технологий на примере роевых алгоритмов, основанных на поведении насекомых и животных, включающее описание биологических предпосылок и самих алгоритмов.**

Ключевые слова **– роевые алгоритмы; бионические технологии; онтологическая модель; алгоритм роя частиц; муравьиный алгоритм; пчелиный алгоритм; алгоритмы, вдохновленные роем светлячков; алгоритм кукушкиного поиска; алгоритм летучих мышей.**

Введение

Для разработки базы знаний интеллектуальной системы информационной поддержки процессов создания и развития перспективных бионических технологий [1, 2] предполагается формирование и использование онтологической модели. Сведения для одного узла дерева онтологической модели – роевого интеллекта в качестве примера представлены в [3].

Накопление сведений для детализации узла «роевой интеллект» по веткам, определяющим различные алгоритмы, позволяет уточнять онтологическую модель бионических технологий.

Поскольку в настоящее время существует большое количество различных роевых алгоритмов, для осуществления быстрого поиска необходимых информационных ресурсов следует все алгоритмы распределить на группы.

Одним из подходов такого разделения может быть группировка на роевые алгоритмы, основанные на поведении насекомых и животных, и роевые алгоритмы, инспирированные неживой природой и бактериями.

Для того, чтобы дальше продолжать детализацию веток онтологической модели необходимо собрать из разных источников сведения по алгоритмам. Эти сведения могут отражать информацию о биологических предпосылках создания алгоритма, его авторах, подробного описания самого алгоритма, области его возможного применения и др.

Далее представлены примеры сведений для описания веток (листьев) дерева онтологической модели по алгоритмам, основанным на поведении насекомых и животных:

– алгоритм роя частиц;
– муравьиный алгоритм;
– пчелиный алгоритм;
– алгоритмы, вдохновленные роем светлячков;
– алгоритм кукушкиного поиска;
– алгоритм летучих мышей.

Примеры описания веток (листьев) дерева онтологической модели по алгоритмам, инспирированным неживой природой и бактериями, представлены в [4].

1 Алгоритм роя частиц

Алгоритм роя частиц (метод роя частиц, англ. Particle Swarm Optimization, PSO) был предложен в 1995 году Джеймсом Кеннеди и Расселом Эберхартом как метод для оптимизации непрерывных нелинейных функций [5].

Вдохновением послужила имитационная модель Рейнольдса, а также работа Хеппнера и Гренадера на схожую тему [6]. Кеннеди и Эберхарт отметили, что обе модели основаны на управлении дистанциями между птицами – а, следовательно, синхронность стаи является в них функцией от усилий, которые птицы прилагают для сохранения оптимальной дистанции.

Летая большими группами, птицы почти никогда не сталкиваются в воздухе, при этом стая двигается плавно и скоординировано. Следует отметить, что вожака в стаях большинства видов птиц нет, при этом все равно обеспечивается скоординированное поведение и стая представляет собой роевый интеллект. Птицы в ней действуют согласно определенным, довольно простым правилам: в небе каждая из них следит за своими сородичами и координирует свое движение согласно их положению [7].

Разработанный алгоритм довольно прост и может быть реализован буквально в нескольких десятках строчек кода на любом высокоуровневом языке программирования. Он моделирует многоагентную систему, где агенты-частицы двигаются к оптимальным решениям, обмениваясь при этом информацией с соседями (рисунок 1).

Статья получена 03.12.2015 г.

Исследование выполнено федеральным государственным бюджетным образовательным учреждением высшего образования «Московский государственный университет информационных технологий, радиотехники и электроники» (МИРЭА, МГУПИ) за счет гранта Российского научного фонда (проект №14-11-00854).

К.т.н., с.н.с., В.В. Баранюк, МГТУ МИРЭА (e-mail: valentina_bar@mail.ru).

О.С. Смирнова, МГТУ МИРЭА (e-mail: mail.olga.smirnova@yandex.ru).

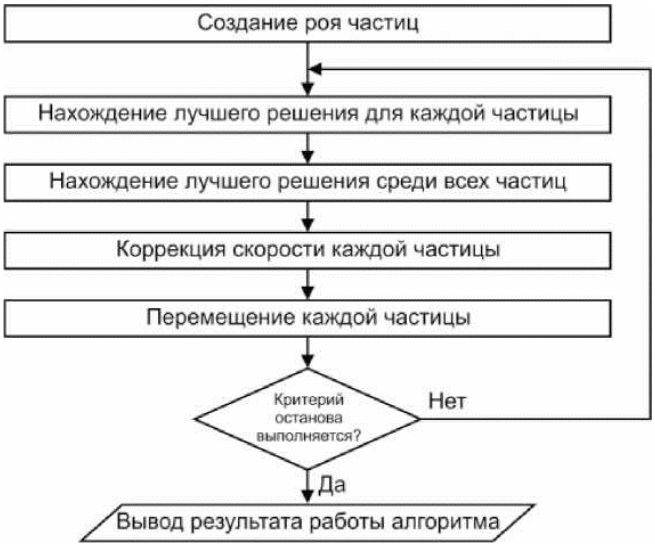

Рисунок 1 – Схема алгоритма роя частиц

Текущее состояние частицы характеризуется координатами в пространстве решений (то есть, собственно, связанным с ними решением), а также вектором скорости перемещения. Оба этих параметра выбираются случайным образом на этапе инициализации. Кроме того, каждая частица хранит координаты лучшего из найденных ею решений, а также лучшее из пройденных всеми частицами решений – этим имитируется мгновенный обмен информацией между птицами. На каждой итерации алгоритма направление и длина вектора скорости каждой из частиц изменяются в соответствие со сведениями о найденных оптимумах:

$$v_i=v_i+a_1 \cdot rnd(\,) \cdot (pbest_i - x_i)+a_2 \cdot rnd(\,) \cdot (gbest_i - x_i\,), \quad (1)$$

где v – вектор скорости частицы (v_i – его i-ый компонент);

a_1, a_2 – постоянные ускорения;

pbest – лучшая найденная частицей точка;

gbest – лучшая точка из пройденных всеми частицами системы;

x – текущее положение частицы.

Функция *rnd( )* возвращает случайное число от 0 до 1 включительно.

После вычисления направления вектора v, частица перемещается в точку $x=x+v$. В случае необходимости, обновляются значения лучших точек для каждой частицы и для всех частиц в целом. После этого цикл повторяется.

Ниже рассмотрены наиболее примечательные из модификаций данного метода.

При обновлении направления и скорости движения частицы в *lbest* используют информацию о решениях соседних частиц:

$$v_i=v_i+a_1 \cdot rnd(\,) \cdot (pbest_i - x_i)+a_2 \cdot rnd(\,) \cdot (lbest_i - x_i), \quad (2)$$

где *lbest* – лучший результат среди частицы и её соседей.

Соседними считаются либо частицы, отличающихся от данной индексом не более чем на некоторое заданное значение, либо частицы, расстояние до которых не превышает заданного порога.

Данный алгоритм более тщательно исследует пространство поиска, однако является более медленным, чем оригинальный. При этом, чем меньшее число соседей учитывается при формировании вектора скорости, тем ниже скорость сходимости алгоритма, но тем эффективней он избегает субоптимальных решений.

В 1998 году Юхи Ши и Рассел Эберхарт предложили модификацию, на первый взгляд совсем незначительно отличающуюся от классического алгоритма [8]. С учётом того, что одной из главных проблем при решении задач оптимизации является баланс между тщательностью исследования пространства поиска и скоростью сходимости алгоритма, а также того, что в зависимости от задачи и характеристик поискового пространства в ней, этот баланс должен быть различным, учёные предложили изменить правило обновления векторов скоростей частиц:

$$v_i=\omega \cdot v_i+a_1 \cdot rnd(\,) \cdot (pbest_i - x_i)+a_2 \cdot rnd(\,) \cdot (gbest_i - x_i\,), \quad (3)$$

где ω – коэффициент инерции.

Коэффициент инерции определяет упомянутый баланс между широтой исследования и вниманием к найденным субоптимальным решениям. В случае, когда ω>1, скорости частиц увеличиваются, они разлетаются в стороны и исследуют пространство более тщательно. В противном случае, скорости частиц со временем уменьшаются и скорость сходимости зависит от выбора параметров a_1 и a_2. Значение коэффициента инерции может как убывать, так и расти. При его убывании частицы сначала исследуют область поиска экстенсивно, находя множество субоптимальных решений, и со временем все более концентрируются на исследовании их окрестностей. Возрастание инерции способствует сходимости алгоритма на поздних стадиях работы.

В 2002 году Марис Клер и Джеймс Кеннеди предложили свою модификацию алгоритма роя частиц, которая стала настолько популярной, что теперь ее принято называть каноническим алгоритмом роя частиц [9]. Он позволяет избавиться от необходимости «угадывать» подходящие значения регулируемых параметров алгоритма, контролируя сходимость частиц.

Был изменен способ вычисления векторов скоростей частиц, введением в него дополнительного множителя:

$$v_i=X|v_i+a_1\,(pbest_i - x_i\,)+a_2\,(gbest_i - x_i\,)|, \quad (4)$$

где $a_1+a_2>4$, а коэффициент сжатия X равен:

$$X = \frac{2k}{\left|2 - a - \sqrt{a^2 - 4a}\right|}$$

Такой подход гарантирует сходимость алгоритма без необходимости явно контролировать скорость частиц.

В своей работе 2004 года Руи Мендес, Джеймс Кеннеди и Жозе Невес заметили, что принятое в каноническом алгоритме роя частиц допущение о том, что на каждую из частиц влияет только наиболее успешная, не соответствует лежащим в его основе природным механизмам и, возможно, ведет к снижению эффективности алгоритма [10]. Ученые предположили, что из-за чрезмерного внимания алгоритма к единственному решению может быть потеряна важная информация о структуре пространства поиска. Исходя из этого, решили сделать все частицы «полностью информированными», то есть получающими информацию от всех соседних частиц. Для этого был

изменен в каноническом алгоритме закон изменения скорости:

$$v_i = X \cdot \left[v_i + \sum_{keN} a_k \cdot W(k) \cdot rnd() \cdot (pbest_{k,i} - x_i) \right], \quad (5)$$

где N – множество соседей частицы,

$pbest_k$ – лучшая из пройденных k-ым соседом точек;

$W(k)$ – весовая функция, которая может отражать любую характеристику k-ой частицы, которая считается важной: значение целевой функции в точке, в которой она находится, дистанцию от нее до данной частицы и так далее [11].

Муравьиный алгоритм

Муравьиный алгоритм (алгоритм оптимизации муравьиной колонии, англ. ant colony optimization, ACO) – один из эффективных полиномиальных алгоритмов для нахождения приближенных решений задач поиска маршрутов на графах.

Следует отметить, что муравьи относятся к социальным насекомым, живущим внутри некоторого коллектива – колонии. Их поведение при транспортировании пищи, преодолении препятствий, строительстве муравейника и других действиях зачастую приближается к теоретически оптимальному. В качестве примера приведена структура взаимосвязанных гнёзд суперколонии муравьев Formica lugubris в Швейцарии. Сеть муравейников близка к минимальному основному дереву, соединяющему все гнёзда колонии – вершины графа на рисунке 2 [12].

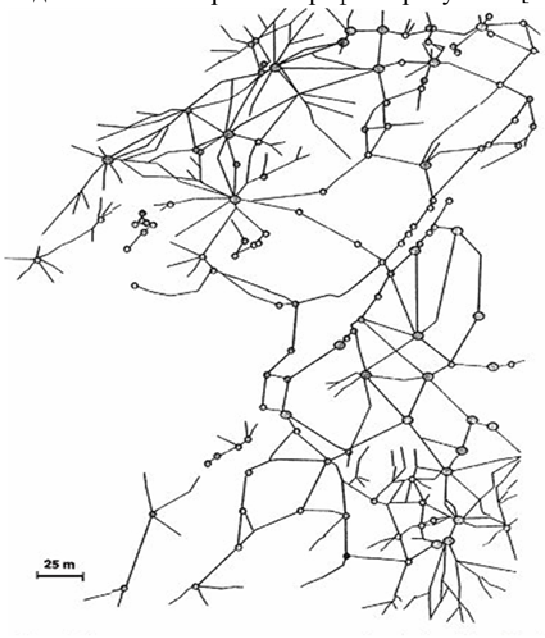

Рисунок 2 – Сеть гнёзд суперколонии муравьев Formica lugubris в Швейцарии

Движение муравья основывается на одном и очень простом вероятностном уравнении. Если муравей еще не закончил путь, то есть не посетил все узлы сети, для определения следующего ребра пути используется уравнение:

$$P = \frac{\tau(r,u)^{\alpha} \times \eta(r,u)^{\beta}}{\sum_k \tau(r,u)^{\alpha} \times \eta(r,u)^{\beta}}, \quad (6)$$

где $\tau(r,u)$ – интенсивность фермента на ребре между узлами r и u;

$\eta(r,u)$ – функция, которая представляет измерение обратного расстояния для грани;

α – вес фермента;

β – коэффициент эвристики.

Параметры α и β определяют относительную значимость двух параметров, а также их влияние на уравнение. Так как муравей путешествует только по узлам, которые ещё не были посещены (как указано списком табу), вероятность рассчитывается только для ребер, которые ведут к ещё не посещённым узлам. Переменная k представляет эти ребра [12].

Пройденный муравьем путь отображается, когда муравей посетит все узлы графа. Циклы запрещены, поскольку в алгоритм включен список табу. После завершения длина пути может быть подсчитана – она равна сумме всех рёбер, по которым путешествовал муравей. Уравнение (7) показывает количество феромона, который был оставлен на каждом ребре пути для муравья k. Переменная Q является константой.

$$\Delta\tau_{ij}^{k}(t) = \frac{Q}{L^{k}(t)}. \quad (7)$$

Результат уравнения является средством измерения пути. Короткий путь характеризуется высокой концентрацией феромонов, а более длинный путь – более низкой. Затем полученный результат используется в уравнении (8), чтобы увеличить количество феромона вдоль каждого ребра пройденного муравьём пути.

$$\tau_{ij}(t) = \Delta\tau_{ij}(t) + (\tau_{ij}^{k}(t) \times \rho. \quad (8)$$

Важно, что данное уравнение применяется ко всему пути, при этом каждое ребро помечается феромоном пропорционально длине пути. Поэтому следует дождаться, пока муравей закончит путешествие и только потом обновить уровни феромона, в противном случае истинная длина пути останется неизвестной. Константа ρ – значение между 0 и 1.

В начале пути у каждого ребра есть шанс быть выбранным. Чтобы постепенно удалить рёбра, которые входят в худшие пути графа, ко всем рёбрам применяется процедура испарения феромона. Используя константу ρ из уравнения (8), мы получаем уравнение (9):

$$\tau_{ij}(t) = \tau_{ij}(t) \times (1-\rho). \quad (9)$$

Поэтому для испарения феромона используется обратный коэффициент обновления пути.

После того как путь муравья завершён, рёбра обновлены в соответствии с длиной пути и произошло испарение феромона на всех рёбрах, алгоритм запускается повторно. Список табу очищается и длина пути обнуляется. Муравьям разрешается перемещаться по графу, основывая выбор ребра на уравнении (6). Этот процесс может выполняться для постоянного количества путей или до момента, когда на протяжении нескольких запусков не было отмечено повторных изменений. Затем определяется лучший путь, который и является решением.

3 Пчелиный алгоритм

Пчелиный алгоритм (в англоязычных статьях также встречаются названия Artificial Bee Colony (ABC) Algorithm и Bees Algorithm) – довольно новый алгоритм для нахождения глобальных экстремумов (максимумов или минимумов) сложных многомерных функций.

Для описания поведения пчёл в природе используются три следующих основных понятия [13].

Источник нектара характеризуется своей полезностью, которая определяется такими факторами, как удалённость от улья, концентрация нектара, удобство его добычи.

Занятые фуражиры – пчёлы, которые «связаны» с одним из источников нектара, т.е. добывают на нём нектар. Занятые фуражиры владеют следующей информацией о «своём» источнике нектара: направление от улья на источник и полезность источника.

Незанятые фуражиры – пчёлы-разведчики, которые осуществляют поиск источников нектара для их использования, а также пчёлы-наблюдатели, которые в данное время выполняют некоторые работы в улье.

Каждый незанятый фуражир может полететь к источнику нектара, следуя за пчелой-разведчиком, которая нашла путь к такому источнику. Пчела-разведчик выполняет «вербовку» незанятых пчел с помощью танца на специальной площадке улья – области танцев. Завербованная пчела следует за соответствующей пчелой-разведчиком к области с нектаром и становится, таким образом, занятым фуражиром.

Занятый фуражир после добычи нектара возвращается в улей и оставляет его там. После этого данный фуражир может выполнить одно из следующих действий: оставить «свой» источник нектара и стать незанятым фуражиром; продолжить заготовку нектара из прежнего источника, не вербуя других пчел; выполнить вербовку. Пчела выбирает одно из указанных действий по некоторому вероятностному закону.

Одновременно в пределах области танцев разные пчёлы могут «рекламировать» различные источники нектара. Механизмы принятия решений, в соответствии с которыми пчела решает следовать за той или иной пчелой-вербовщиком, исследованы недостаточно хорошо. Логично предположить, что вероятность вербовки тем или иным образом определяется полезностью соответствующего источника нектара [14].

Таким образом, самоорганизация пчелиного роя основывается на четырёх следующих основных механизмах:

1) положительная обратная связь – на основе информации, полученной от других пчёл, пчела летит к одному из источников нектара;

2) отрицательная обратная связь – основываясь на информации, полученной от других пчёл, данная пчела может решить, что «её» источник нектара значительно хуже других найденных источников и оставить этот источник;

3) случайность – вероятностный поиск пчёлами-разведчиками новых источников нектара;

4) множественность взаимодействия – информация об источнике нектара, найденном одной пчелой, передается многим другим пчёлам улья.

В информатике и исследовании операций, пчелиный алгоритм на основе алгоритма поиска впервые разработан в 2005 году. В базовой версии алгоритм выполняет своего рода соседний поиск в сочетании со случайным поиском и может использоваться для функциональной и комбинаторной оптимизации. На рисунке 3 представлена схема работы пчелиного алгоритма.

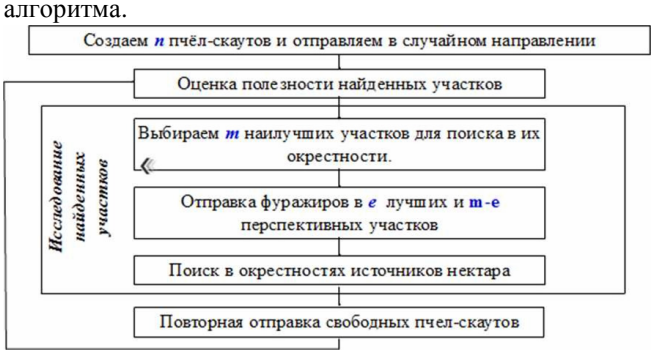

Рисунок 3 – Схема работы алгоритма пчёл

Применительно к задаче оптимизации в пчелином алгоритме каждое решение представляется в виде пчелы, которая знает (хранит) расположение (координаты или параметры многомерной функции) какого-то участка [15]. Выделим два варианта поведения.

В первом варианте две пчелы нашли два разных пересекающихся участка и оба этих участка следует отметить как лучшие или выбранные. Во втором варианте будем считать, что это один участок, центр которого находится в точке, соответствующему большому значению целевой функции.

Второй вариант поведения менее подвержен попаданию в локальные оптимумы за счёт просмотра перспективных мест и их окрестностей. Причём, на каждой итерации алгоритма область просмотра уменьшается.

Основная идея пчелиного алгоритма заключается в том, что все пчёлы на каждом шаге будут выбирать как элитные участки для исследования, так и участки в окрестности элитных, что позволит, во-первых, разнообразить популяцию решений на последующих итерациях, во-вторых, увеличить вероятность обнаружения, близких к оптимальным решений. После чего в окрестности остальных участков (m-e), в зависимости от значения их целевой функции (ЦФ), отправляются рабочие пчёлы (l = N-n).

Таким образом, работа алгоритма зависит от следующих основных параметров: общего числа пчёл-разведчиков (N); общего числа участков (m); числа элитных участков (e); числа пчёл-разведчиков на элитных участках (n); числа пчёл (l) на остальных (m-e) участках; начального размера участков, т.е. размера участков вместе с их окрестностями (S); максимального числа итераций (I).

Описание алгоритма сводится к следующему [16]. Сначала создается популяция пчёл и производится оценка их ЦФ, затем выбор участков для поиска в их

окрестностях и отправка пчёл-разведчиков, далее производится выбор пчёл с лучшими значениями ЦФ с каждого участка и отправка рабочих пчёл, осуществляющих случайный поиск с оценкой их ЦФ. Затем формируется новая популяция и производится проверка условия окончания работы алгоритма.

Таким образом, ключевой операцией алгоритма пчёл является совместное исследование перспективных областей и их окрестностей. В конце работы алгоритма популяция решений будет состоять из двух частей: пчёлы с лучшими значениями ЦФ элитных участков, а также группы рабочих пчёл со случайными значениями ЦФ. Отличительной особенностью алгоритма является способность динамически разбивать поисковое пространство на области, что уменьшает время работы алгоритма. Главным преимуществом является тот факт, что резко снижается вероятность по¬падания в локальный оптимум, а за счёт распараллеливания уменьшается время работы алгоритма.

4 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 (англ. firefly algorithm) и алгоритм оптимизации роем светлячков (англ. Glowworm Swarm Optimization, GSO). Основное различие между firefly- и glowworm-светлячками состоит в том, что вторые являются бескрылыми [17].

Алгоритм светлячков

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

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:

$$B_{i,j} = \beta_0 \exp(-\gamma r_{i,j}^2), \quad i,j \in [1:|S|], i \neq j, \qquad (10)$$

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 (10) можно заменить функцией $\frac{1}{1+r_{i,j}^2}$. При этом указанное выражение приобретает вид:

$$B_{i,j} = \frac{\beta_0}{1 + \gamma r_{i,j}^2}, \quad i,j \in [1:|S|], i \neq j. \qquad (11)$$

Формулы (10), (11) определяют характерное расстояние $r_c = 1/\sqrt{\gamma}$, на котором привлекательность изменяется от β_0 до $\beta_0 e^{-1}$ для случая (10) и от β_0 до $\beta_0/2$ для случая (11).

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

$$\beta(r) = \beta_0 e^{-\gamma r^n}, \quad n \geq 1. \qquad (12)$$

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

$$X_j = X_i + \beta(r_{i,j})(X_j - X_i) + \alpha U_{|X|}(-1;1), \; i,j \in [1:|S|], \; i \neq j, \qquad (13)$$

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:

$$\alpha = \alpha_\infty + (\alpha 0 - \alpha_\infty)e^{-t}, \qquad (14)$$

где α_0 – начальное значение параметра рандомизации;

α_x – его окончательное значение.

Для улучшения сходимости алгоритма в формуле (13) может быть использован еще один член вида:

$$\lambda U(-1;1) \cdot (X_j - X^{best}), \qquad (15)$$

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 (15) перемешаем 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: $\beta_0 = 1$; $\alpha \in [0; 1]$; $\gamma = 1$.

Алгоритм оптимизации роем светлячков

Алгоритм GSO предложили в 2005 г. Кришнананд и Гхоус [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:

$l_i'=(1-\rho)l_i + \gamma\varphi(X_i)$, $i \in [1:|S|]$, (16)

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:

$$X_i' = Xi + \lambda \frac{X_j - X_i}{\|X_j - X_i\|E}, \quad i,j \in [1:|S|], i \neq j, \qquad (17)$$

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

Новый радиус окрестности светлячка s_i определяем в соответствии с выражением:

$r_i'=min(r_{min},max(0,(r_i+\varepsilon(n-|N_i|))))$, $i \in [1:|S|]$, (18)

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

$X_{i,j}' = X_{i,j} + \lambda(0{,}5 - U_i(0;1))$, $j \in [1:|X|]$. (19)

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 (18), а по формуле вида:

$$r_i' = \frac{r_{max}}{1 + \varepsilon_r d_i}, \qquad (20)$$

где r_{max} – максимально допустимое значение этой величины;

$\varepsilon_r > 0$ – заданная константа;

d_i – средняя текущая плотность агентов в окрестности N_i.

$$d_i = \frac{|N_i|}{\pi(r_i)^2} \qquad (21)$$

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 её 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 $[0{,}33\lambda; 0{,}1r_s]$, где r_s – свободный параметр алгоритма.

5 Алгоритм кукушкиного поиска

Алгоритм кукушкиного поиска (англ. Cuckoo Search, CS) был предложен в 2009 г. [20]. Алгоритм вдохновлен поведением кукушек в процессе вынужденного гнездового паразитизма.

Такие виды кукушек, как Ani и Guird, 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 [17].

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 гнёздах. 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 гнёзда 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 полётов Леви [20] и находим новое положение кукушки X_c;

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

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

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

Полёты Леви кукушки реализуем по формуле:

$$X_c' = X_c + V \text{ □ } L_{|X|}(\lambda), \qquad (22)$$

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

Известно несколько модификаций алгоритма *CS*. В каноническом алгоритме *CS* кукушка в своих полётах 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:

$$V=U_1(0;1)(X^{best} - X_k),\ k \text{ □ } [1:|S|], \qquad (23)$$

где $X_к \neq X^{best}$ – случайно выбранное гнездо [21]. Так определенный вектор V обеспечивает большую вероятность полёта кукушки к гнёздам, имеющим высокую приспособленность.

Вместе с тем, в каноническом алгоритме *CS* вероятность ξ_a и параметры полёта 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:

$$\xi_a(t) = \xi_a^{max} - \frac{1}{t}\left(\xi_a^{max} - \xi_a^{min}\right); \qquad (24)$$

$$v(t) = v^{max}\exp(d^t), \qquad d\frac{1}{t}\ln\left(\frac{v^{min}}{v^{max}}\right). \qquad (25)$$

Здесь ξ_a^{min}, ξ_a^{max}, v^{min}, v^{max} – заданные константы.

6 Алгоритм летучих мышей

Алгоритм летучих мышей (англ. Bat Algorithm, BA) был предложен в 2010 г. [22].

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 лоцирующего 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 тоже время некоторые виды не используют частотную модуляцию своих звуковых импульсов [17].

Алгоритм BA предполагает следующую модель поведения летучих мышей:

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

б) мыши движутся случайным образом. Текущие положение и скорость мыши S_i, i□[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□[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 BA включает следующие основные шаги:

1) инициализируем популяцию агентов S_i, i□[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□[1:|S|], выполняем следующие действия:

– генерируем случайное число $u=U_1(0;1)$; если $u>r_i$, то находим лучшее решение X_j данного агента;

– 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 $<a_i$ и $\varphi(X_i)<\varphi(X^{best})$, то принимаем решение X_j в качестве нового текущего положения агента S_j, 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 (рисунок 4).

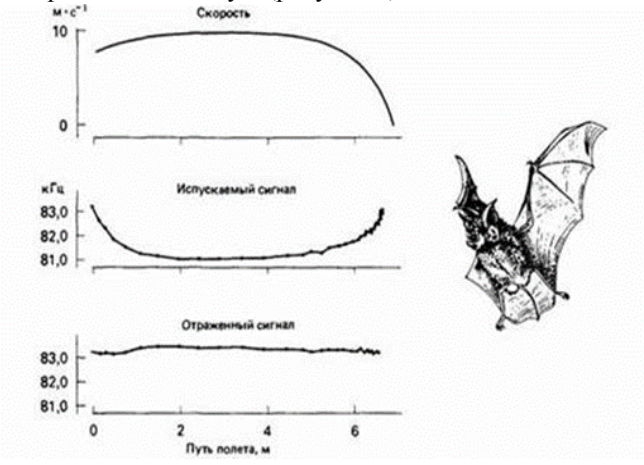

Рисунок 4 – путь полета летучей мыши

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:

$$X_i^{'}=X_i+V_i^{'},$$
$$V_i^{'}=V_i+\omega_i^{'}(X_i - X^{best}),$$
$$\omega_i^{'}=\omega^{min}+(\omega^{max}-\omega^{min})U_1\ (0;1). \quad (26)$$

Другими словами, на данной итерации агент перемещается в направлении, определяемом суммой вектора перемещения на предыдущей итерации (слагаемое V_i в формуле (26)) и случайным образом возмущённого 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 [23].

Случайный локальный поиск выполняем по следующей схеме:

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

$$X_i^{'}=X_i+\bar{a}U_{|X|}(-1;1),\ i \in [1:|S|], \quad (27)$$

где a – текущее среднее значение громкостей всех агентов популяции:

$$\bar{a} = \frac{1}{|S|}\sum_{i=1}^{|S|} a_i; \quad (28)$$

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:

$$a_i^{'}=b_a\, a_i, r_i^{'}=r_i^0\ (1-\exp(-b_r\, t)),\ i \in [1:|S|], \quad (29)$$

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 в тех же условиях уменьшаем по экспоненциальному закону, так что имеют место предельные соотношения

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

Заключение

Представленные сведения по роевым алгоритмам позволяют детализировать узлы, ветки и листья онтологической модели бионических технологий от узла «роевой интеллект» и создавать расширяемое представление рассматриваемой предметной области.

При формировании базы знаний интеллектуальной системы информационной поддержки процессов создания и развития перспективных бионических технологий [24] может использоваться меньший или больший объём данных по сравнению с предложенным в данном примере.

Сведения по роевым алгоритмам, собранные в статье из разных источников, могут быть полезны специалистам при проведении работ в области бионики или при решении различных задач оптимизации.

БИБЛИОГРАФИЯ

[1] Сигов А.С., Нечаев В.В., Кошкарев М.И. Архитектура предметно-ориентированной базы знаний интеллектуальной системы. International Journal of Open Information Technologies ISSN: 2307-8162 vol. 2, no.12, 2014.

[2] Sigov A.S., Nechaev V.V., Baranyuk V.V., Koshkarev M.I., Smirnova O.S., Melikhov A.A., Bogoradnikova A.V. Architecture of domain-specific data warehouse for bionic information resources. Ecology, environment and conservation, №4, 2015.

[3] Баранюк В.В., Смирнова О.С. Роевой интеллект как одна из частей онтологической модели бионических технологий. International Journal of Open Information Technologies. Vol.3, no. 12, 2015.

[4] Смирнова О.С., Богорадникова А.В., Блинов М.Ю. Описание роевых алгоритмов, инспирированных неживой природой и бактериями, для использования в онтологической модели. International Journal of Open Information Technologies. Vol.3, no. 12, 2015.

[5] J. Kennedy, R. C. Eberhart, «Particle swarm optimization» // In Proceedings of IEEE International Conference on Neural Networks, стр. 1942–1948, 1995 г.

[6] F. Heppner, U. Grenander, «A stochastic nonlinear model for coordinated bird flocks» // The Ubiquity of Chaos, стр. 233–238, 1990 г.

[7] Объектно-событийное программирование. Роевые алгоритмы. Эл. ресурс: http://bourabai.kz/alg/swarm.htm

[8] Y. Shi, R. Eberhart, «A modified particle swarm optimizer» // The 1998 IEEE International Conference on Evolutionary Computation Proceedings, стр. 69–73, 1998 г.

[9] M. Clerc, J. Kennedy, «The particle swarm – explosion, stability, and convergence in a multidimensional complex space» // IEEE Transactions on Evolutionary Computation, №6 (1), стр. 58–73, 2002 г.

[10] R. Mendes, J. Kennedy, J. Neves, «The fully informed particle swarm: Simpler, maybe better» // IEEE Transactions on Evolutionary Computation, №8 (3), стр. 204–210, 2004 г.

[11] Алгоритм роя частиц. Эл. ресурс: http://habrahabr.ru/post/105639/

[12] Алгоритмы муравьиной колонии. Эл. ресурс: http://www.wikiznanie.ru/wikipedia/index.php/Алгоритмы_муравьиной_колонии

[13] Гришин А.А., Карпенко А.П. Исследование эффективности метода пчелиного роя в задаче глобальной оптимизации. Электронное научно-техническое издание «Наука и образование», №8 август 2010 г.

[14] Sumpter D.J.T., Broomhead D.S. Formalising the Link between Worker and Science: Proceedings of the First International Workshop on Multi-Agent Systems and Agent-Based Simulation. – MABS'98 LNAI, 1998. – pp.95 – 110

[15] Курейчик В.В., Курейчик Вл.Вл. Биоспирированный поиск при проектировании и управлении. Известия ЮФУ. Технические науки. 178–193 с.

[16] Курейчик В.В., Запорожец Д.Ю. Роевой алгоритм в задачах оптимизации //Известия ЮФУ. Технические науки. – 2010. – №7 (108). – С.28–32.

[17] Карпенко А.П. Популяционные алгоритмы глобальной поисковой оптимизации. Обзор новых и малоизвестных алгоритмов. Журнал «Информационные технологии», Приложение, №7/2012 г. Изд.: «Новые технологии», 32 с.

[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] 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.

[21] 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.

[22] 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.

[23] Карпенко А. П., Селиверстов Е. Ю. Глобальная оптимизация методом роя частиц. Обзор // Информационные технологии. 2010. № 2. С. 25–34.

[24] Баранюк В.В., Смирнова О.С., Богорадникова А.В. Интеллектуальная система информационной поддержки развития перспективных бионических технологий: основные направления работ по созданию. International Journal of Open Information Technologies ISSN: 2307-8162 vol. 2, no. 12, 2014.

Detailed swarm intelligence algorithms description for expanding the bionics ontology

V.V. Baranjuk, O.S. Smirnova

Abstract – **The paper regards the detailed description of swarm algorithms based on animal and insect behaviour. This description represents some nodes and leaves of a large bionics ontology.**

Keywords – **swarm algorithms; bionics; ontological model; particle swarm optimization; ant colony optimization; artificial bee colony algorithm; glowworm swarm optimization; "cuckoo search" algorithm; bat algorithm.**