УДК 004.023 DOI 10.21777/2500-2112-2025-3-83-95

МЕТОД ВЫБОРА И РАСПРЕДЕЛЕНИЯ ЗАДАЧ НА ОСНОВЕ МЕТОДОЛОГИИ SCRUM С ИСПОЛЬЗОВАНИЕМ ПОПУЛЯЦИОННЫХ АЛГОРИТМОВ

Трошкин Николай Романович[1],
e-mail: ni_troshkin@mail.ru,

Вишневская Татьяна Ивановна[1],
канд. физ.-мат. наук, доцент,
e-mail: vti@bmstu.ru,
[1]*Московский государственный технический университет имени Н.Э. Баумана, г. Москва, Россия*

В статье описывается проблема выбора и распределения задач между исполнителями в рамках проектов с использованием методологии Scrum. Предлагается решение данной проблемы путем сведения ее к задаче о множественном рюкзаке и решения с использованием популяционных алгоритмов, в частности ансамбля генетического алгоритма и алгоритма бактериального хемотаксиса. В статье приводится модель расчета приоритета и трудоемкости задачи для конкретного исполнителя. Описываются классы входных данных. Излагаются основные этапы реализации алгоритмов, значения их настроечных параметров. Разработана метрика оценки качества результатов работы предложенного метода. Определен набор параметров алгоритмов, при которых показатель качества будет высоким для большинства классов входных данных. Приводится сравнение времени работы программной реализации алгоритмов и их комбинации, а также качества получаемых результатов вычисления. Делаются выводы о применимости алгоритмов ансамбля к выделенным классам входных данных. Результаты исследования показывают эффективность использования разработанного метода для распределения работ по сравнению с составляющими его отдельными алгоритмами.

Ключевые слова: распределение задач по исполнителям, генетический алгоритм, алгоритм бактериального хемотаксиса, популяционные алгоритмы, задача о рюкзаке, комбинаторная оптимизация, управление проектами, методология Scrum

TASK SELECTION AND ALLOCATION METHOD BASED ON SCRUM METODOLOGY USING POPULATION ALGORITHMS

Troshkin N.R.[1],
e-mail: ni_troshkin@mail.ru,

Vishnevskaya T.I.[1],
candidate of physical and mathematical sciences, associate professor,
e-mail: vti@bmstu.ru,
[1]*Bauman Moscow State Technical University, Moscow, Russia*

The article describes the problem of task selection and distribution among executors in the context of software projects using Scrum methodology. The solution to this problem is proposed by reducing it to the problem of multiple knapsacks and solving it using population algorithms, in particular, the ensemble of genetic algorithm and bacterial chemotaxis algorithm. The article presents a model for calculating the priority of a task and its execution time for a particular performer. Classes of input data are described. The main stages of the algorithms and the values of their tuning parameters are explained. A metric for evaluating the quality of the results of the proposed method has been developed. The set of algorithms parameters, at which the quality index will be high for most classes of input data, is defined. The comparison of the operating time of the software implementation of the algorithms and their combinations, as well as the quality of the obtained calculation results, is given. Conclusions are drawn about

the applicability of the ensemble algorithms to the selected classes of input data. The results of the study show the effectiveness of using the developed method for distributing work in comparison with its individual algorithms.

Keywords: task allocation, genetic algorithm, bacterial chemotaxis algorithm, population algorithms, knapsack problem, combinatorial optimization, project management, Scrum methodology

Введение

Многие современные организации сталкиваются с необходимостью решать задачи нормирования и распределения работ между исполнителями. На некоторых проектах численность исполнителей превышает два десятка специалистов, что делает грамотное распределение заданий между ними нетривиальной задачей даже при использовании современных гибких методологий. Более того, при распределении, как правило, требуется учитывать приоритет задач и уровень квалификации персонала, что лишь добавляет ограничений и усложняет задачу.

Одной из методологий планирования программных проектов является методология гибкого управления проектами Scrum, суть которой заключается в разбиении проекта на небольшие этапы (спринты) длиной в 1–4 недели для достижения поставленной цели. На протяжении всего проекта ведется так называемый бэклог, содержащий все задачи, поставленные перед командой на данный момент. Как правило, задач в бэклоге существенно больше, чем команда может выполнить за спринт. Таким образом, перед менеджером возникает задача выбрать из бэклога наиболее приоритетные задачи и распределить их по сотрудникам, не перегружая их.

Обзор существующих методов

Проблема распределения работ между исполнителями исследовалась с разных перспектив, включая алгоритм Джонсона для производственных процессов с двумя [1] и тремя [2] станками. Однако поставленная в данной работе задача более сложна из-за неопределенности времени выполнения и большего числа исполнителей. В некоторых статьях предлагались решения узкоспециализированных задач, включающие нейронные сети [3] (многослойный персептрон и сети Кохонена), миварные экспертные системы [4], а также усложненный вариант генетических алгоритмов [5], но сложность предложенных методов затрудняет их применение к другим задачам. Авторами работы [6] были рассмотрены «жадные» алгоритмы, которые принимают локально оптимальные решения на различных стадиях проектов, что актуально для техподдержки с равной квалификацией сотрудников. Тем не менее большинство предложенных методов не учитывают специфику методологии Scrum, где задачи часто тщательно декомпозируются и гибко распределяются, а исполнители не привязаны к графику слишком жестко, поэтому важнее определить приоритетные задачи и распределить их, учитывая квалификацию членов команды, чем следовать строгим временным рамкам.

Применительно к другим предметным областям похожие математические задачи удавалось решать при помощи метода ветвей и границ [7]. Метод можно использовать для данной задачи, однако он подходит только для задач совсем небольшой размерности ввиду его экспоненциальной сложности, например, когда задач менее 20 при 2–3 исполнителях.

Формализованная постановка задачи

Проблему выбора и распределения задач можно свести к задаче о множественном рюкзаке [8]. Это задача комбинаторной оптимизации, которая заключается в следующем: имеется набор из n предметов, каждый из которых имеет массу w_j и ценность v_j. Требуется собрать m рюкзаков таким образом, чтобы каждый имел массу не больше некоторой предельной массы W_i, на которую рассчитан рюкзак, и при этом включал в себя максимально возможную суммарную ценность.

В применении к распределению задач между сотрудниками вместо рюкзаков рассматриваются исполнители, вместо грузоподъемности – длительность рабочего времени (например, i-ый сотрудник

работает T_i часов за спринт). В качестве ценных предметов, которыми наполняется рюкзак, будут рассматриваться задачи. Их весом t_{ij} будет считаться время выполнения j-ой задачи i-ым исполнителем, которое оценивается менеджером проекта в часах. «Ценность» задачи p_j будет определяться на основе ее приоритета, зависящего от определенных параметров задачи, – чем выше приоритет, тем важнее положить задачу в «рюкзак» исполнителю. Таким образом, добавив также ограничения, что одно задание может выполнять только один сотрудник, а время простоя каждого исполнителя должно быть минимальным, представить задачу можно с использованием математической модели в виде системы (1):

$$\begin{cases} h = \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \to \max, \\ g_i = \left| T_i - \sum_{j=1}^{n} t_{ij} x_{ij} \right| \to \min, \quad \forall i \in \overline{1,m}, \\ \sum_{i=1}^{m} x_{ij} \le 1, \qquad\qquad \forall j \in \overline{1,n}, \end{cases} \tag{1}$$

где p_j – приоритет j-ой задачи;

T_i – время в часах, которое должен отработать i-ый работник за спринт;

t_{ij} – время выполнения j-ой задачи i-ым исполнителем;

m – количество работников;

n – количество задач.

Величины x_{ij} являются искомыми, $x_{ij} = 1$ для некоторых i, j, если j-ая задача назначена i-му сотруднику, и 0 в противном случае.

Для решения поставленной задачи требуется знать время выполнения каждой задачи каждым сотрудником, так как исполнители могут выполнять одну и ту же работу за разное время, так как у всех разный опыт и набор умений. При большом количестве задач и сотрудников менеджеру будет сложно оценить вручную все необходимые параметры t_{ij}.

Пусть j-ая задача из бэклога принадлежит некоторому классу $c = skill(j)$ (например, если речь идет о разработке программного обеспечения, это может быть «разбор багов», «написание документации», «проверка кода», «написание нового модуля», задачи по конкретной технологии и т.д.). Предполагается, что в обязанности каждого исполнителя входит выполнение любого класса задач. Разница лишь в том, что каждый сотрудник выполняет задачи каждого класса с разной скоростью.

Введем матрицу навыков S размера $m \times l$, где $l << n$ – количество возможных классов задач. Элементом матрицы будет безразмерная величина $s_{ij} \in [0.1, 1]$, характеризующая, как сотрудник i справляется с задачами класса j. Значение 1 подразумевает экспертный уровень владения задачей. Так как $l << n$, то заполнить такую матрицу гораздо проще. Более того, навыки исполнителей меняются не каждый спринт, поэтому менеджеру на каждом спринте потребуется внести в такую матрицу лишь минимальные правки.

Тогда при оценке времени выполнения задачи менеджеру потребуется лишь определить класс задачи и ее базовую трудоемкость – время τ_j, с которым эту задачу выполнял был человек с уровнем соответствующего навыка, равным среднему по команде.

Для конкретного исполнителя трудоемкость задачи будет определяться по формуле (2):

$$t_{ij} = s_{skill(j)} \cdot \frac{\tau_j}{s_{i,skill(j)}}, \tag{2}$$

где $s_{skill(j)}$ – среднее значение в столбце $skill(j)$ матрицы S.

Помимо того, необходимо определить, как именно оценить степень приоритета задачи. Этим также занимается менеджер проекта. Требуется разработать такую модель расчета приоритета задачи, в которой исключается возможность игнорирования срочных задач, но при этом избежать бесконечного откладывания задач с невысоким приоритетом.

Задачу можно охарактеризовать несколькими параметрами. Опишем те из них, которые влияют на приоритет задачи больше всего и на основе которых она будет определяться:

– срок выполнения *u* – время (в днях) до наиболее поздней даты, когда задача должна быть полностью выполнена;

– критичность для проекта *c* – безразмерный показатель от 0 до 1, чем он больше, тем сильнее влияет задача на успешное достижение цели спринта или стратегических целей организации в целом, и тем выше должна быть вероятность, что задача будет выбрана из бэклога на очередной спринт;

– количество задач *d*, зависящих от данной, – чем их больше, тем важнее выполнить задачу.

Все параметры, кроме сроков выполнения, должны определяться для каждой задачи путем экспертной оценки перед планированием спринта. Наибольшим приоритетом должна обладать задача, критичная для всего проекта, от которой зависит больше всего других задач и при этом срок выполнения которой минимален. В то же время от спринта к спринту ситуация на проекте может быть разной: в одних случаях требуется сделать больший упор на более важные задачи, в других – на более срочные. Это также требуется корректировать в зависимости от нужд менеджера. Для этого введем еще три положительных коэффициента, которые будут управлять вкладом соответствующих параметров задачи в общий приоритет:

– *cu* для срока выполнения;

– *cc* для критичности;

– *cd* для количества зависимых задач.

На основе этих параметров предлагается модель (3) определения ценности p_j *j*-ой задачи на основе ее срока выполнения u_j, критичности c_j и числа d_j зависящих от нее задач:

$$p_j = cc \cdot c_j + cd \cdot d_j + cu \cdot \frac{1}{u_j}. \quad (3)$$

Классы входных данных

Входными данными для данной задачи является информация об исполнителях и задачах. В зависимости от соотношения числа исполнителей и задач можно выделить следующие классы входных данных:

1. Исполнителей много, положим, 10 и более, задач много, более 50. Случай можно назвать «стандартным» – из бэклога требуется выбрать некоторую часть задач и раздать исполнителям.

2. Исполнителей мало, менее 10, и задач мало, менее 50. Случай аналогичен первому, но задача имеет меньшую размерность.

3. Исполнителей мало, а задач много. В этом случае из бэклога будет выбрано не так много задач, поэтому особенно важно выбрать среди них наиболее критичные и приоритетные.

4. Исполнителей много, задач мало. Маловероятная ситуация с учетом ограничений, введенных при постановке задачи, в ней возможны простои сотрудников. Здесь важно, чтобы одна задача не досталась двум и более сотрудникам, и точно не было перегрузок.

Помимо соотношения размерностей входных данных, на распределение задач будут влиять такие их характеристики, как срочность и важность. Выделить классы входных данных с точки зрения срочности и важности набора задач можно с использованием матрицы Эйзенхауэра. В соответствии с матрицей, в первую очередь должны быть сделаны срочные и важные задачи, затем, когда будет время, важные, но несрочные задачи. Оставшиеся задачи делаются в последнюю очередь. Это должно учитываться при выборе задач из бэклога.

По срочности и важности входные данные о задачах можно поделить на следующие классы:

1. Число всех 4 типов задач по матрице Эйзенхауэра примерно одинаково (случайный набор задач).

2. Почти все задачи срочные и важные, что соответствует «авральной» ситуации на проекте.

3. Почти все задачи несрочные, среди них есть важные и неважные – обратная крайность, «спокойная» ситуация на проекте, когда в ближайшее время нет строгих дедлайнов.

Таким образом, все входные данные можно разделить на классы, множество которых равно декартовому произведению множества классов входных данных на основе их размерности и множества классов на основе типов распределяемых задач.

Формализуем качественные параметры срочности и важности. Срочной будем считать задачу,

для которой параметр *u* (срок выполнения в днях) меньше 10. Важной будем считать задачу, для которой параметр *c* (важность для проекта) превышает 0.5.

Как было описано выше, класс входного набора данных зависит от соотношения срочных и важных задач:

1. К классу «важные и срочные задачи» будем относить такой набор задач, в котором более 70 % задач – срочные и более 70 % задач – важные.

2. К классу «важные, но несрочные задачи» будем относить такой набор задач, в котором более 70 % задач – важные, но менее 30 % задач – срочные.

3. Если набор задач не удалось отнести ни к одному из двух классов, такой набор относится к классу «случайных задач».

Для упрощения изложения пронумеруем выделенные классы в соответствии с таблицей 1.

Таблица 1 – Выделенные классы входных данных

Номер класса *K*	**Число исполнителей *m***	**Число задач *n***	**Срочность и важность задач**
1	< 10	< 50	Срочные и важные
2	< 10	< 50	Важные, но несрочные
3	< 10	< 50	Случайные
4	< 10	≥ 50	Срочные и важные
5	< 10	≥ 50	Важные, но несрочные
6	< 10	≥ 50	Случайные
7	≥ 10	< 50	Срочные и важные
8	≥ 10	< 50	Важные, но несрочные
9	≥ 10	< 50	Случайные
10	≥ 10	≥ 50	Срочные и важные
11	≥ 10	≥ 50	Важные, но несрочные
12	≥ 10	≥ 50	Случайные

Описание предложенного метода

Решаемая задача относится к классу NP-полных, поэтому ее решение, как правило, ищется приближенными методами, в частности, эвристическими алгоритмами. Среди них для рассмотрения были выбраны генетический алгоритм и алгоритм бактериального хемотаксиса, обладающие полиномиальной сложностью и сравнительно небольшим числом настроечных параметров. На их основе разработан метод, являющийся ансамблем этих двух алгоритмов, позволяющий по имеющимся данным об исполнителях и задачах, к которым относятся время работы и квалификация сотрудника, описанные выше параметры приоритета, трудоемкость и класс задач, выбрать задачи из бэклога и распределить их между исполнителями. В зависимости от того, к какому классу относятся входные данные, выбирается один из алгоритмов. Применение того или иного алгоритма в зависимости от класса будет исследовано и уточнено далее. На рисунке 1 приводится верхнеуровневая IDEF0-диаграмма метода.

Блок A1 IDEF0-диаграммы, изображенной на рисунке 1, подразумевает два действия – определение класса *K* входных данных и непосредственно выбор алгоритма на основе этого класса. На рисунке 2 приводится блок-схема алгоритма, реализующего блок A1 и передающего управление в следующие блоки.

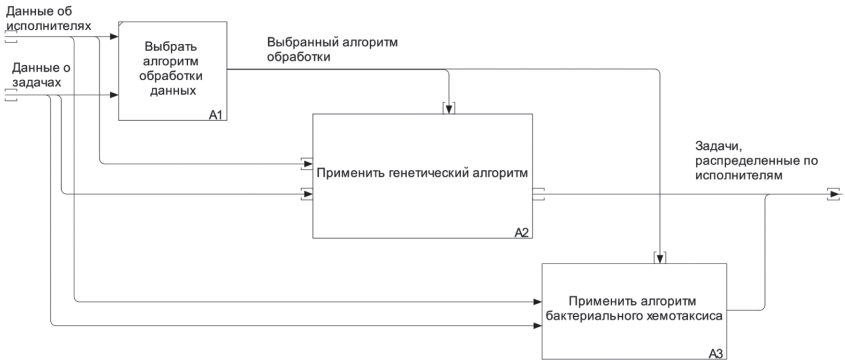

Рисунок 1 –Декомпозиция предложенного метода

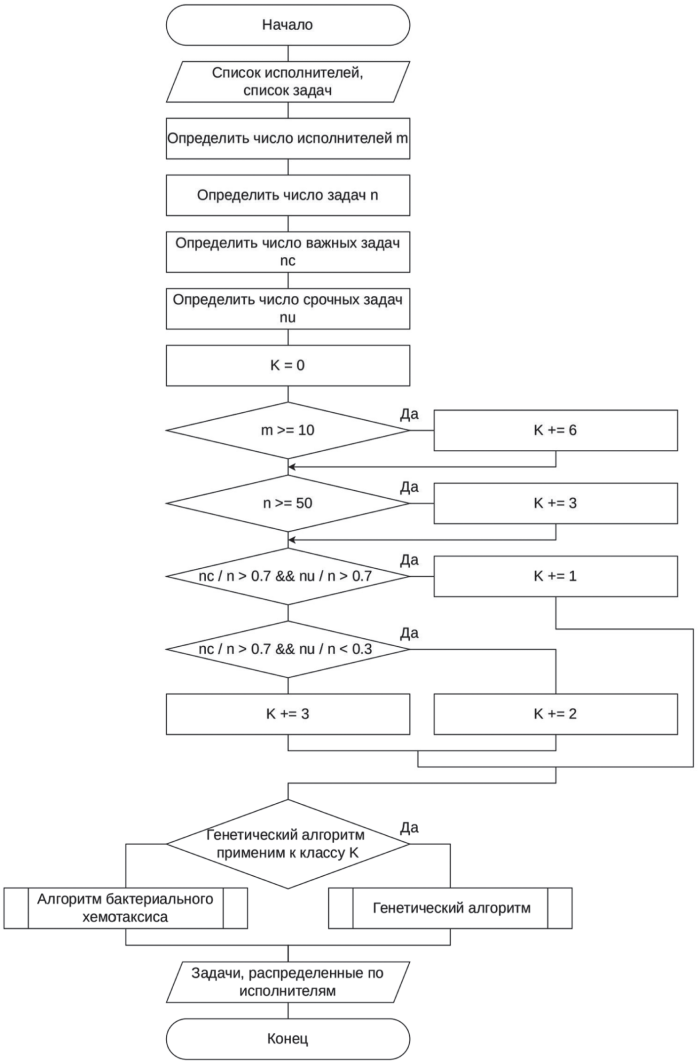

Рисунок 2 – Блок-схема алгоритма анализа класса входных данных

Генетический алгоритм

На рисунке 3 приводится диаграмма генетического алгоритма в нотации IDEF0. Алгоритм имеет три настроечных параметра: количество особей в популяции N, количество поколений G, в ходе которых развивается популяция, и вероятность мутации генов p.

Генетический алгоритм основан на селекции лучших особей в популяции в ходе естественного отбора[1]. Сначала генерируется начальная популяция заданной мощности. Каждая особь представляет собой некоторый вариант решения задачи – распределение задач по исполнителям, то есть матрицу размера m на n из 0 и 1. Далее начинается итеративный процесс на протяжении заданного числа поколений. Для каждой особи считается ее приспособленность на основе формул (4)–(5), которые основаны на суммарном приоритете выбранных задач за вычетом штрафа за нарушения ограничений (переработки и простой разработчиков, дублирование задач) [9]:

$$f(x)=\sum_{i=1}^{m}\sum_{j=1}^{n}x_{ij}p_j-\rho\cdot\left(\sum_{i=1}^{m}\left|\sum_{j=1}^{n}x_{ij}t_{ij}-T_i\right|+\sum_{j=1}^{n}n_{d_j}\right),\quad \rho=\max_j\left\{\frac{p_j}{t_j}\right\}, \tag{4}$$

$$n_{d_j}=\begin{cases}\dfrac{p_j}{p}\cdot\sum_{i=1}^{m}x_{ij}, & \sum_{i=1}^{m}x_{ij}>1,\\ 0, & \sum_{i=1}^{m}x_{ij}\le 1\end{cases} \tag{5}$$

В формулах выше p – средний приоритет задач в бэклоге;

ρ – поправочный коэффициент;

n_{d_j} – параметр, учитывающий число дубликатов задачи j;

t_j – усредненная трудоемкость j-ой задачи по всем исполнителям.

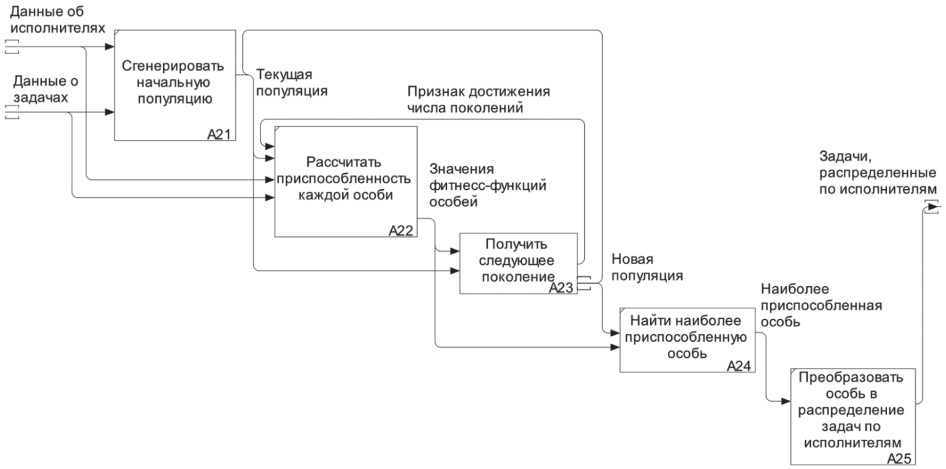

Рисунок 3 – Диаграмма генетического алгоритма в нотации IDEF0

На основе полученных данных происходит получение нового поколения, причем особи с большей приспособленностью будут оставлять потомство чаще других. Из последнего поколения находится наиболее приспособленная особь, которая и характеризует полученный результат.

При генерации нового поколения для каждой новой особи выбираются два родителя из текущего поколения. Каждый родитель получается турнирным методом – случайно выбираются две особи и одним из родителей становится особь с большим показателем приспособленности. Далее родительские особи скрещиваются, то есть обмениваются генами – элементами своих матриц. Потомок получает очередной ген от того или иного родителя с равной вероятностью, такое скрещивание называется однородным [10; 11]. После скрещивания гены потомка могут мутировать с заданной вероятностью – это может быть инверсия того или иного бита, либо перестановка двух соседних битов. На рисунке 4 приводится диаграмма описанного процесса в нотации IDEF0.

[1] *Гладков Л.А., Курейчик В.В., Курейчик В.М.* Генетические алгоритмы: учеб. пособие для вузов. – Москва: Физматлит, 2006. – 319 с.

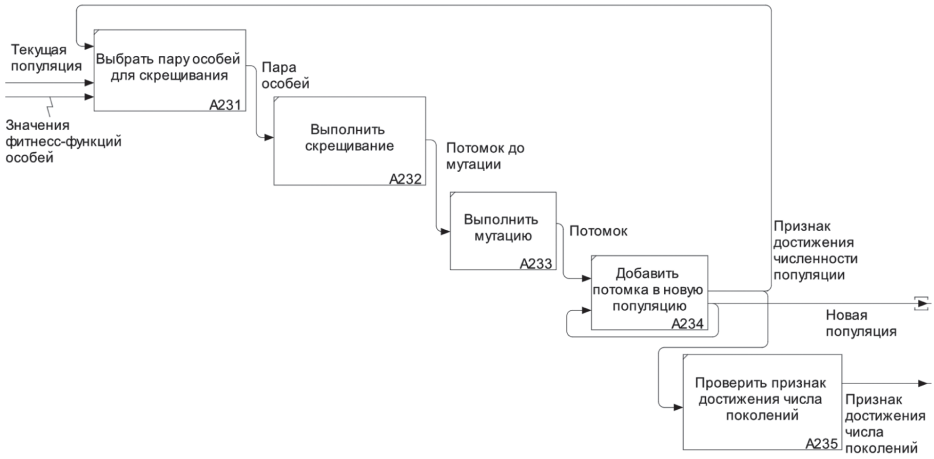

Рисунок 4 – Диаграмма алгоритма получения нового поколения

Алгоритм бактериального хемотаксиса

Алгоритм предложен Хансом Бремерманном [12] и основан на механизме поведения бактерий, который называется бактериальным хемотаксисом. Механизм заключается в двигательной реакции микроорганизмов на химический раздражитель среды: бактерия по концентрации привлекающего вещества или аттрактанта определяет, в какую сторону и с какой скоростью двигаться. При этом, если аттрактанта в данной точке пространства достаточно много, она задерживается в ней, а иначе ее скорость увеличивается. Применительно к задаче оптимизации значением концентрации аттрактанта может служить градиент целевой функции, который можно вычислить как среднее изменение значения фитнесс-функции за последние h итераций. А положение каждой бактерии в пространстве представляет собой некоторое приближенное решение задачи аналогично набору хромосом особи в генетическом алгоритме.

Канонический алгоритм строится на следующих идеях [13]:

1. Алгоритм описывает движение особей последовательностью прямолинейных траекторий, соединенных мгновенными поворотами. Движение характеризуется скоростью, направлением и продолжительностью.

2. Направление поворота бактерии определяется вероятностным распределением, что позволяет учитывать случайные изменения в движении.

3. Алгоритм использует информацию о градиентах функции, чтобы направлять бактерию к оптимальному решению, минимизируя количество итераций, необходимых для достижения цели.

В первоначальной версии алгоритма бактериальной оптимизации строилась весьма сложная модель с большим количеством настроечных параметров, учетом угла поворота бактерий, используется несколько случайных величин для формирования случайных изменений в движении [13].

Помимо этого, алгоритм обладал следующими недостатками:

1. Медленная скорость сходимости [14]. Бактерии, обнаружив локальный оптимум, начинают роиться вокруг него, не улучшая решение.

2. Невозможность выйти за пределы локального оптимума [15]. Так как предполагается, что бактерия задерживается в области с высоким значением функции приспособленности, это приводит к тому, что требуется вводить дополнительные операции ликвидации и рассеивания, чтобы дать возможность исследования новых областей определения функции на наличие других локальных экстремумов.

С целью упрощения вычислений и преодоления указанных недостатков было разработано множество модификаций алгоритма. В модификации, предлагаемой и используемой в данной работе, решено отказаться от вычисления угла поворота бактерии, а также скорости и длительности ее движения и заменить их перемещением на некоторое приращение ее координат. При этом приращение координат рассчитывается с учетом градиента, за счет чего длина шага становится переменной, что улучшает скорость сходимости алгоритма. Помимо изменений в расчете новых решений, алгоритм предполагает

перемешивание координат текущего положения бактерий с координатами лучшего решения. За счет этого могут появиться новые решения за пределами локального оптимума, что позволяет исследовать новые области внутри области определения целевой функции.

Настроечными параметрами алгоритма являются число итераций G, число бактерий в популяции N, размер хранимой истории h.

В качестве фитнесс-функции используется та же функция, что и в генетическом алгоритме, описанная формулами (4)–(5).

Для относительного изменения приспособленности сначала вычисляется среднее изменение приспособленности за последние h итераций по соотношению (6):

$$\Delta_{avg} = \frac{1}{h-1}\sum_{i=1}^{h-1} fh_i - fh_{i-1}, \quad (6)$$

где fh_i – i-ый элемент массива с историей значений приспособленности данной бактерии.

Относительное изменение приспособленности имеет вид (7):

$$\Delta = \frac{1-\left|f - fh_h\right|}{\Delta_{avg}}, \quad (7)$$

где f – текущее значение приспособленности,

fh_h – значение на предыдущей итерации.

На рисунке 5 приводится описание алгоритма бактериального хемотаксиса в виде блок-схемы.

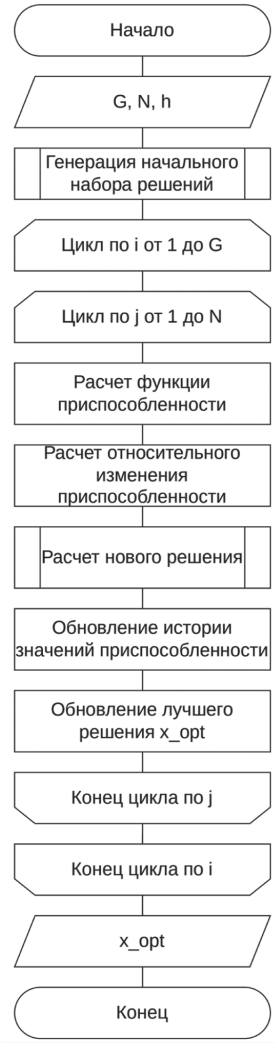

Рисунок 5 – Блок-схема алгоритма бактериального хемотаксиса

Исследование предложенного метода

Предложенный метод был реализован в виде Desktop-приложения на языке Java. Данные о задачах и исполнителях, а также результат работы программы представляются в виде *csv*-файлов.

Чтобы проверить корректность получаемого результата, была введена метрика оценки качества на основе трех критериев:

– у разработчиков не должно быть переработок или простоя больше, чем 5 % от их рабочего времени;

– среди выбранных задач из бэклога не должно быть дубликатов;

– среди выбранных задач должны быть наиболее приоритетные.

Количественно метрика вычисляется по формуле (8):

$$q = \frac{load + dbl + prior}{3}, \quad q \in [0,1], \tag{8}$$

где параметры имеют следующие значения, лежащие в отрезке [0; 1]:

– *load* – доля разработчиков, суммарная длительность задач которых не отличается от их рабочих часов больше, чем на 5 %;

– *dbl* – доля недублирующихся задач в итоговой выборке из бэклога;

– *prior* – доля задач, находящихся среди *n* наиболее приоритетных в общем списке, где *n* – общее число задач, выбранных из бэклога.

Для алгоритмов, составляющих метод, исследовано влияние настроечных параметров на качество результата путем изменения каждого из них при фиксации остальных двух параметров.

Для генетического алгоритма для большинства классов входных данных было получено, что наилучшие результаты получаются при 500 поколениях из 100–500 особей с вероятностью мутации около 0.05, хотя на некоторых классах имело место улучшение результатов при повышении вероятности мутации до 0.25 и даже 0.5.

Для алгоритма бактериального хемотаксиса было получено, что лучших результатов можно достичь при 500 особях, при этом чаще всего достаточно всего 10 итераций, а влияние размера хранимой истории на результат оказалось для данной задачи несущественным.

На рисунке 6 приводится диаграмма, отображающая лучшие показатели качества алгоритмов метода на разных классах входных данных. При исследовании для каждого алгоритма проводилось по 100 запусков для каждого класса при значениях настроечных параметров, описанных выше, и значение качества усреднялось по каждому набору, а затем выбиралось наилучшее.

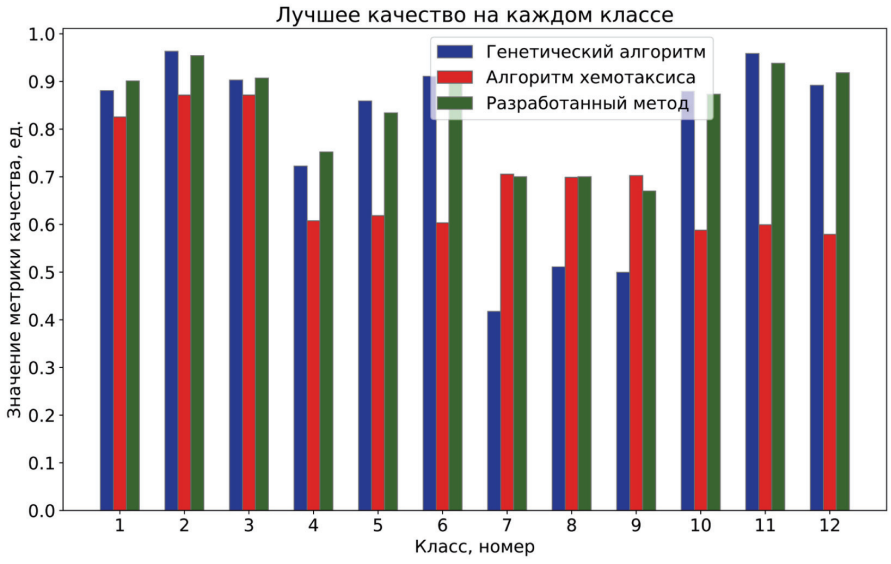

Рисунок 6 – Сравнение лучших результатов алгоритмов на каждом классе

Сравнение времени работы алгоритмов приводится на рисунке 7.

Разработанный метод является ансамблем двух представленных выше алгоритмов. Это означает, что в зависимости от того, к какому классу принадлежат входные данные, к ним применяется тот или иной алгоритм.

При выбранных параметрах реализация алгоритма бактериального хемотаксиса работает в 10, а для некоторых классов даже в 20–30 раз быстрее, чем реализация генетического алгоритма. Однако в случае, когда число задач велико (например, 200), он существенно уступает генетическому алгоритму в качестве результатов. Поэтому для таких входных данных (классы 4–6, 10–12) в ансамбле используется генетический алгоритм.

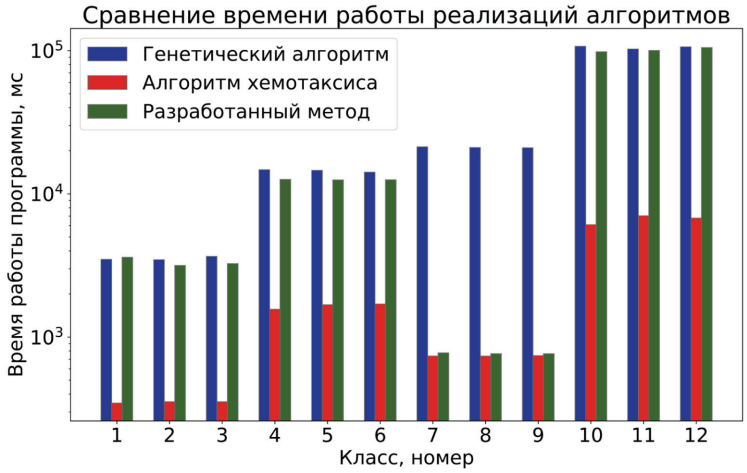

Рисунок 7 – Сравнение времени работы алгоритмов на каждом классе

При 20 исполнителях и 40 задачах (классы 7–9) точность алгоритма бактериального хемотаксиса, наоборот, значительно выше, чем у генетического алгоритма, а с учетом времени работы становится очевидно, что для этого класса следует использовать именно его.

Для классов 1–3 имеет место небольшая разница в точности двух алгоритмов. Если в требованиях к системе важнее скорость работы программы, можно немного пожертвовать качеством результатов и выбрать алгоритм бактериального хемотаксиса. Если в системе важнее более высокая точность, то для данных классов входных данных подойдет генетический алгоритм. Для примера в данной работе для этих классов применяется генетический алгоритм.

Среднее качество результатов, которое получилось при помощи разработанного метода, составляет примерно 0.84, в то время как генетический алгоритм в среднем дает качество 0.78, а алгоритм хемотаксиса – 0.69. Следовательно, разработанный метод, являющийся ансамблем двух алгоритмов, показывает большую универсальность и точность, нежели составляющие его алгоритмы по отдельности. Вместе с тем среднее время работы реализации ансамбля при тестировании примерно на 20 % меньше по сравнению с чисто генетическим алгоритмом. Алгоритм бактериального хемотаксиса работает значительно быстрее, но показывает при этом худшую точность. Таким образом, разработанный метод позволяет добиться компромисса между скоростью обработки данных и качеством получаемых результатов.

Заключение

В данной работе был описан метод выбора и распределения задач на основе методологии Scrum с использованием популяционных алгоритмов. Дано описание основных этапов реализации метода в виде детализированных IDEF0-диаграмм и схем алгоритмов. Проведена параметризация метода с целью выявить такие наборы настроечных параметров у алгоритмов, при которых значение введенной метрики качества было бы наиболее высоким. Проведено сравнение качества результатов, получаемых

с использованием каждого алгоритма, и сделан выбор, к каким классам входных данных подходит каждый из них. В результате исследования удалось подобрать такие параметры алгоритмов и такое распределение их по классам входных данных, при которых среднее значение качества – около 0.84 (при максимальном значении, равном 1), что показывает применимость разработанного метода к распределению задач между исполнителями. В дальнейшем планируется модифицировать метод, усовершенствовав функцию приспособленности и улучшив модель оценки приоритета задач для лучшего учета зависимостей между задачами и баланса между срочностью и важностью задач.

Список литературы

1. *Хитрова Т.И.* Проблемы распределения работ в процессе реализации инновационных задач // Baikal Research Journal. – 2020. – Т. 11, № 2.
2. *Шкурба В.В.* Задача трех станков. – Москва: Наука, 1976. – 96 с.
3. *Лукьянов Л.А., Спивак С.И., Христолюбов В.Л.* Нейронная сеть корректор для распределения работ в задаче внутрицехового планирования // Вестник Башкирского университета. – 2016. – № 4. – С. 859–863.
4. *Абросимова Н.Г., Арбузов А.П., Саврасов П.А., Аксенова М.В., Антонов А.И.* Разработка миварной экспертной системы для организации управления проектами IT-компании // Искусственный интеллект в автоматизированных системах управления и обработки данных: сборник статей Всероссийской научной конференции, Москва, 27–28 апреля 2022 года: в 2 т. – Москва: Изд-во МГТУ им. Н.Э. Баумана, 2022. – Т. 2. – С. 13–19.
5. *Орловский Н.М., Воробьев С.П.* Применение метода ветвей и границ и генетических алгоритмов к задаче планирования действий экипажа // Известия высших учебных заведений. Северо-Кавказский регион. Технические науки. – 2013. – № 6 (175). – С. 19–26.
6. *Новикова Т.П., Новиков А.И.* Алгоритм решения задачи оптимального распределения работ в сетевых канонических структурах // Лесотехнический журнал. – 2014. – Т. 4, № 4 (16). – С. 309–317.
7. *Сучкова Т.М., Волкова Л.Л.* О разработке рекомендательной системы для составления туристических маршрутов на основе тематических предпочтений пользователя // Искусственный интеллект в автоматизированных системах управления и обработки данных: сборник статей II Всероссийской научной конференции, Москва, 27–28 апреля 2023 года: в 5 т. – Москва: КДУ: Добросвет, 2023. – Т. 1. – С. 145–150.
8. *Kellerer H., Pferschy U., Pisinger D.* Knapsack Problems. – Springer-Verlag Berlin Heidelberg, 2004. – 560 p.
9. *Трошкин Н.Р., Вишневская Т.И.* Метод распределения задач между членами команды разработчиков программного обеспечения с использованием генетического алгоритма // Искусственный интеллект в автоматизированных системах управления и обработки данных: сборник статей III Всероссийской научной конференции, Москва, 30 октября – 1 ноября 2024 года. – Москва: КДУ, 2025. – Т. 1. – С. 127–133.
10. *Саймон Д.* Алгоритмы эволюционной оптимизации / пер. с англ. А.В. Логунова. – Москва: ДМК Пресс, 2020. – 1002 с.
11. *Рутковская Д., Пилиньский М., Рутковский Л.* Нейронные сети, генетические алгоритмы и нечеткие системы / пер. с польского И.Д. Рудинского. – Москва: Горячая линия – Телеком, 2006. – 452 с.
12. *Bremermann H.* Chemotaxis and optimization // Journal of The Franklin Institute-engineering and Applied Mathematics. – 1974. – Vol. 297. – P. 397–404.
13. *Muller S., Marchetto J., Airaghi S.* Optimization based on bacterial chemotaxis // Ieee Transactions on Evolutionary Computation. – 2002. – Vol. 6, No. 1. – P. 16–29.
14. *Венцов Н.Н., Долгов В.В., Мезина А.В.* Исследование подходов подстройки шага хемотаксиса в алгоритме бактериальной оптимизации // Современные наукоемкие технологии. – 2020. – № 6-1. – С. 25–30.
15. *Запорожец Д.Ю., Ксалов А.М., Пшенокова И.А.* Алгоритм решения задачи параметрической оптимизации на основе методов бактериальной оптимизации // Информатика, вычислительная техника и инженерное образование. – 2016. – № 1 (25). – С. 14–24.

References

1. *Hitrova T.I.* Problemy raspredeleniya rabot v processe realizacii innovacionnyh zadach // Baikal Research Journal. – 2020. – T. 11, № 2.

2. *Shkurba V.V.* Zadacha trekh stankov. – Moskva: Nauka, 1976. – 96 s.
3. *Luk'yanov L.A., Spivak S.I., Hristolyubov V.L.* Nejronnaya set' korrektor dlya raspredeleniya rabot v zadache vnutricekhovogo planirovaniya // Vestnik Bashkirskogo universiteta. – 2016. – № 4. – S. 859–863.
4. *Abrosimova N.G., Arbuzov A.P., Savrasov P.A., Aksenova M.V., Antonov A.I.* Razrabotka mivarnoj ekspertnoj sistemy dlya organizacii upravleniya proektami IT-kompanii // Iskusstvennyj intellekt v avtomatizirovannyh sistemah upravleniya i obrabotki dannyh: sbornik statej Vserossijskoj nauchnoj konferencii, Moskva, 27–28 aprelya 2022 goda: v 2 t. – Moskva: Izd-vo MGTU im. N.E. Baumana, 2022. – T. 2. – S. 13–19.
5. *Orlovskij N.M., Vorob'ev S.P.* Primenenie metoda vetvej i granic i geneticheskih algoritmov k zadache planirovaniya dejstvij ekipazha // Izvestiya vysshih uchebnyh zavedenij. Severo-Kavkazskij region. Tekhnicheskie nauki. – 2013. – № 6 (175). – S. 19–26.
6. *Novikova T.P., Novikov A.I.* Algoritm resheniya zadachi optimal'nogo raspredeleniya rabot v setevyh kanonicheskih strukturah // Lesotekhnicheskij zhurnal. – 2014. – T. 4, № 4 (16). – S. 309–317.
7. *Suchkova T.M., Volkova L.L.* O razrabotke rekomendatel'noj sistemy dlya sostavleniya turisticheskih marshrutov na osnove tematicheskih predpochtenij pol'zovatelya // Iskusstvennyj intellekt v avtomatizirovannyh sistemah upravleniya i obrabotki dannyh: sbornik statej II Vserossijskoj nauchnoj konferencii, Moskva, 27–28 aprelya 2023 goda: v 5 t. – Moskva: KDU: Dobrosvet, 2023. – T. 1. – S. 145–150.
8. *Kellerer H., Pferschy U., Pisinger D.* Knapsack Problems. – Springer-Verlag Berlin Heidelberg, 2004. – 560 p.
9. *Troshkin N.R., Vishnevskaya T.I.* Metod raspredeleniya zadach mezhdu chlenami komandy razrabotchikov programmnogo obespecheniya s ispol'zovaniem geneticheskogo algoritma // Iskusstvennyj intellekt v avtomatizirovannyh sistemah upravleniya i obrabotki dannyh: sbornik statej III Vserossijskoj nauchnoj konferencii, Moskva, 30 oktyabrya – 1 noyabrya 2024 goda. – Moskva: KDU, 2025. – T. 1. – S. 127–133.
10. *Sajmon D.* Algoritmy evolyucionnoj optimizacii / per. s angl. A.V. Logunova. – Moskva: DMK Press, 2020. – 1002 s.
11. *Rutkovskaya D., Pilin'skij M., Rutkovskij L.* Nejronnye seti, geneticheskie algoritmy i nechetkie sistemy / per. s pol'skogo I.D. Rudinskogo. – Moskva: Goryachaya liniya – Telekom, 2006. – 452 s.
12. *Bremermann H.* Chemotaxis and optimization // Journal of The Franklin Institute-engineering and Applied Mathematics. – 1974. – Vol. 297. – P. 397–404.
13. *Muller S., Marchetto J., Airaghi S.* Optimization based on bacterial chemotaxis // Ieee Transactions on Evolutionary Computation. – 2002. – Vol. 6, No. 1. – P. 16–29.
14. *Vencov N.N., Dolgov V.V., Mezina A.V.* Issledovanie podhodov podstrojki shaga hemotaksisa v algoritme bakterial'noj optimizacii // Sovremennye naukoemkie tekhnologii. – 2020. – № 6-1. – S. 25–30.
15. *Zaporozhec D.Yu., Ksalov A.M., Pshenokova I.A.* Algoritm resheniya zadachi parametricheskoj optimizacii na osnove metodov bakterial'noj optimizacii // Informatika, vychislitel'naya tekhnika i inzhenernoe obrazovanie. – 2016. – № 1 (25). – S. 14–24.

Статья поступила в редакцию: 30.04.2025
Статья поступила для публикации: 16.05.2025

Received: 30.04.2025
Accepted: 16.05.2025