ИНФОРМАТИКА И ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ

УДК 004.89:528.8 Научная статья
DOI: 10.35330/1991-6639-2024-26-5-73-83
EDN: IBXKOR

Распараллеливание алгоритма муравьиной колонии на примере задачи о рюкзаке с использованием Python

М. Р. Вагизов✉, С. П. Хабаров

Санкт-Петербургский государственный лесотехнический
университет им. С. М. Кирова
194021, Россия, Санкт-Петербург, Институтский пер., 5

Аннотация. В статье рассмотрен алгоритм муравьиной колонии и описан процесс его распараллеливания с использованием Python и модуля multiprocessing. На примере задачи о рюкзаке показано, что распределение задач между рядом процессов позволяет улучшить производительность алгоритма, сохраняя его эффективность. По сравнению с точными методами типа динамического программирования использование алгоритма муравьиной колонии показало значительное сокращение времени выполнения при приемлемом уровне отклонения от оптимального решения. Преимущество алгоритмов распараллеливания заключается в эффективном использовании вычислительной системы, где используются все доступные ядра процессоров, что приводит к ускорению выполнения большего числа итераций за то же время. Полученные результаты подтверждают потенциал ACO для решения сложных задач с ограниченным временем расчета.

Ключевые слова: алгоритм муравьиной колонии, задача о рюкзаке, параллельные вычисления, программирование на Python

Поступила 25.09.2024, одобрена после рецензирования 02.10.2024, принята к публикации 09.10.2024

Для цитирования. Вагизов М. Р., Хабаров С. П. Распараллеливание алгоритма муравьиной колонии на примере задачи о рюкзаке с использованием Python // Известия Кабардино-Балкарского научного центра РАН. 2024. Т. 26. № 5. С. 73–83. DOI: 10.35330/1991-6639-2024-26-5-73-83

MSC: 94-08 Original article

Parallelizing the ant colony algorithm for solving the knapsack problem as an example using Python

M.R. Vagizov✉, S.P. Khabarov

St. Petersburg State Forestry Engineering
University named after S.M. Kirov
194021, Russia, St. Petersburg, 5 Institutsky lane

Abstract. The paper considers the ant colony algorithm and describes the process of its parallelization using Python and multiprocessing module. Using the example of the knapsack problem, it is shown that distributing tasks among a number of processes allows to improve the performance of the algorithm while maintaining its efficiency. Compared to exact methods, like dynamic programming, the use of the ant colony algorithm showed a significant reduction in execution time with an acceptable level of deviation from the optimal solution. The advantage of parallelization algorithms is the efficient

utilization of the computing system, where all available processor cores are used, resulting in faster execution of more iterations in the same time. The results obtained confirm the potential of ACO for solving complex problems with limited computation time.

Keywords: ant colony algorithm, forest resource optimization, knapsack problem, heuristic algorithms

Submitted 25.09.2024, approved after reviewing 02.10.2024, accepted for publication 09.10.2024

For citation. Vagizov M.R., Khabarov S.P. Parallelizing the ant colony algorithm for solving the knapsack problem as an example using Python. *News of the Kabardino-Balkarian Scientific Center of RAS.* 2024. Vol. 26. No. 5. Pp. 73–83. DOI: 10.35330/1991-6639-2024-26-5-73-83

Введение

Алгоритм муравьиной колонии (ACO) является одним из самых мощных инструментов в комбинаторной оптимизации. Он используется в логистике, робототехнике, туризме, сетевом планировании и других областях [1–3]. При классическом подходе каждый муравей независимо строит свое решение, опираясь на информацию о феромонах и эвристические данные. И только после прохождения всех муравьев происходит обновление феромонов на основе их решений. В стандартной реализации это организуется как один цикл, где каждый муравей по очереди строит свое решение.

Метод исследования

Допустим, что в колонии 40 муравьев. Обычно для их работы запускается один цикл длиной 40 итераций, где каждый муравей поочередно строит свой маршрут и вносит вклад в общий процесс оптимизации. Но можно поступить и иначе: один большой цикл разбить на ряд малых. Например, на 4 цикла по 10 муравьев каждый. Принцип работы алгоритма не изменится: все 40 муравьев все равно обойдут свое пространство решений, а феромоны будут обновлены только после окончания всех 4 циклов. То есть последовательный процесс можно дробить на более мелкие части, которые могут выполняться независимо друг от друга. Это естественным образом ведет нас к идее распараллеливания, когда запуск всех 4 циклов одновременно в 4 параллельных потоках вместо их последовательного выполнения значительно ускорит процесс решения задачи.

Задача о рюкзаке – одна из наиболее изученных задач комбинаторной оптимизации, где нужно выбрать подмножество предметов с учетом их веса и стоимости. При распараллеливании решения этой задачи в каждый поток надо как передать параметры задачи (вес и стоимость предметов), так и определить часть муравьев, которая будет в нем работать. В нашем примере каждый поток будут обрабатывать 10 муравьев. После того как все потоки завершат свои вычисления, результаты объединяются, и феромоны обновляются так, как если бы это был единый последовательный процесс.

Распараллелить задачу в Python можно, используя модуль multiprocessing, который позволяет запускать несколько процессов параллельно, существенно ускоряя выполнение задачи, особенно на многоядерных системах. Рассмотрим пример кода для распараллеливания ACO применительно к задаче о рюкзаке, в которой веса и стоимости предметов генерируются случайным образом:

```
def model(n, range_v=(100, 500), range_w=(30, 100)):
    np.random.seed(42)
    v  = np.random.randint(range_v[0], range_v[1], size=n)
```

```
    w = np.random.randint(range_w[0], range_w[1], size=n)
    return v, w
```

Функции select_item и ant_solution аналогичны однопроцессорной работе, так как выполняют свои задачи для одного муравья и не требуют параллелизма.

```
def select_item(probabilities):
    return np.random.choice(len(probabilities), p=probabilities)
def ant_solution(pheromones, w, v, s, alpha, beta):
    num_items = len(w)
    selected_items = []
    current_weight = 0
    available_items = list(range(num_items))
    while available_items:
        pheromone = pheromones[available_items]
        heuristic = v[available_items] / (w[available_items] + 1e-10)
        probabilities = (pheromone ** alpha) * (heuristic ** beta)
        probabilities /= probabilities.sum()
        item = available_items.pop(select_item(probabilities))
        if current_weight + w[item] <= s:
            selected_items.append(item)
            current_weight += w[item]
    return selected_items
```

Функция select_item отвечает за выбор одного элемента из списка доступных на основе заданных вероятностей. Используется для принятия решения, какой предмет добавить в решение текущего муравья. Функция ant_solution строит решение для одного муравья, последовательно выбирая предметы на основе феромонов и эвристической информации, пока не будет достигнут максимально допустимый вес. Она управляет процессом выбора предметов, формируя одно возможное решение задачи.

Стандартная для алгоритма муравьиной колонии функция ant_colony_opt при переходе от однопроцессорной к многопроцессорной обработке проходит значительные изменения, разбиваясь на две ключевые функции:

- process_group – выполняет задачи для группы муравьев в отдельном процессе, осуществляя генерацию решений, обновление локальных феромонов и нахождение лучшего решения внутри своей группы;
- ant_colony_opt – координирует работу процессов, собирает результаты от групп муравьев, обновляет глобальные феромоны и управляет итерациями оптимизации для поиска лучшего глобального решения.

Функция ant_colony_opt охватывает общую логику работы ACO и демонстрирует, как взаимодействуют все компоненты. Только поняв, как она работает, можно перейти к более детальному знакомству с process_group, так как она реализует конкретную работу группы муравьев в многопроцессорной среде, что представляет собой детали реализации внутри каждого процесса.

```
def ant_colony_opt(pheromones, w, v, s, alpha, beta, rho, Q, iter, ants, proc):
    best_solution = None
    best_value = float('-inf')
    for i in range(iter):
        t0 = time.time()
        # Создание пула процессов
        pool = multiprocessing.Pool(processes=proc)
        results = []
        # Распараллеливание обработки муравьев
        for group_id in range(proc):
            result = pool.apply_async(process_group,
                args=(pheromones, w, v, s, alpha, beta, Q, ants, group_id, proc)
             )
           results.append(result)
        pool.close()
        pool.join()
        # Сбор и обработка результатов
        local_pheromones = np.zeros(len(w))
        for result in results:
            solution, value, group_pheromones = result.get()
            if value > best_value:
                best_solution = solution
                best_value = value
            local_pheromones += group_pheromones
        pheromones = (1 - rho) * pheromones + local_pheromones
        tk = time.time()
        print(f"Итерация {i+1},  Решение: {best_value},  t={tk-t0:.2f} сек")
    return best_solution, best_value
```

Организация процессов играет ключевую роль в ускорении вычислений и эффективном распределении задач среди доступных процессоров. На каждой итерации создается пул процессов с количеством рабочих процессов, заданным параметром proc, что позволяет одновременно выполнять несколько задач.

Внутри цикла для каждой группы муравьев, используя метод apply_async объекта multiprocessing.Pool, асинхронно вызывается функция process_group. Она будет обрабатывать группу муравьев, генерировать решения и обновлять локальные феромоны. Список results хранит все асинхронные задачи для последующего получения их результатов.

После запуска всех задач метод pool.close() закрывает пул процессов для новых задач, а метод pool.join() блокирует основной поток выполнения до тех пор, пока все процессы в пуле не завершат свою работу.

После завершения всех процессов результаты извлекаются с помощью метода result.get(). Сначала собираются решения и обновления феромонов от каждой группы. На их основе обновляются глобальные феромоны и находится лучшее решение для текущей итерации.

Функция process_group отвечает за выполнение работы группы муравьев в рамках одного процесса. В многопроцессорной обработке она выполняет задачи для конкретной группы муравьев и возвращает результаты, которые собираются и обрабатываются в функции ant_colony_opt.

```
def process_group(pheromones, w, v, s, alpha, beta, Q, ants, group_id, proc):
    local_pheromones = np.zeros(len(w))
    local_best_solution = None
    local_best_value = float('-inf')
    ants_per_group = ants // proc
    for _ in range(ants_per_group):
        solution = ant_solution(pheromones, w, v, s, alpha, beta)
        value  = np.sum(v[solution])
        weight = np.sum(w[solution])
        if value > local_best_value and weight <= s:
            local_best_solution = solution
            local_best_value = value
        for item in solution:
            local_pheromones[item] += Q / value
    print(f"----- Процесс {group_id}   Решение группы - {local_best_value}")
    return local_best_solution, local_best_value, local_pheromones
```

На этапе инициализации определяются: local_pheromones – массив для хранения феромонов, обновляемых внутри этого процесса, local_best_solution – локальное лучшее решение, найденное в данном процессе, и local_best_value – значение лучшего решения. Значение ants_per_group определяет количество муравьев, обрабатываемых в рамках данного процесса. Общее число муравьев в ACO делится между всеми доступными процессами (proc).

Для каждой муравьиной итерации вызывается функция ant_solution, которая строит решение на основе текущих феромонов, весов, ценностей и других параметров. Вычисляются ценность (value) и вес (weight) найденного решения. Если найденное решение лучше локального лучшего решения и его вес не превышает ограничения, оно сохраняется как лучшее решение для данного процесса.

На основе полученной цены решения обновляются локальные феромоны, выводится лучшее решение для данного процесса, а затем лучшее решение, его значение и обновленные феромоны функция возвращает в ant_colony_opt для последующей агрегации в рамках решения задачи о рюкзаке. Функция main() управляет процессом оптимизации ACO, отвечает за настройку параметров, инициализацию данных и запуск основной функции оптимизации.

```
def main():
    n = 100
    v, w = model(n)
    s = 500
    ants  = 100      # Количество муравьев
    proc  =  4       # Количество процессов для параллелизма
```

```
alpha = 1.0      # Влияние феромонов
beta  = 2.0      # Влияние эвристической информации
rho   = 0.5      # Коэффициент испарения феромонов
Q     = 100      # Константа для обновления феромонов
iter  =  10      # Количество итераций
pheromones = np.ones(len(w))
# Запуск оптимизации муравьиной колонии
start_time = time.time()
best_solution, best_value = ant_colony_optimization(
          pheromones, w, v, s, alpha, beta, rho, Q, iter, ants, proc )
print(f"Лучшее решение: {best_solution};  Общая цена: {best_value}")
print(f"Общее время выполнения: {time.time() - start_time:.2f} секунд")
```

Результат исследования

С целью оценки работоспособности проекта и эффективности работы ACO был проведен тест на наборе из 100 предметов, с которыми работали 100 муравьев в течение 10 итераций, показавший, что многопоточная работа заметно снижает общее время решения задачи (рис. 1).

```
N     = 100      alpha = 1.0
ants  = 100      beta  = 2.0
proc  =  4       rho   = 0.5
                 Q     = 100
                 iter  =  10

Iteration 1,   Time: 2.73 sec,   Best Value: 5252
Iteration 2,   Time: 2.81 sec,   Best Value: 5324
Iteration 3,   Time: 2.75 sec,   Best Value: 5392
Iteration 4,   Time: 2.77 sec,   Best Value: 5763
Iteration 5,   Time: 3.39 sec,   Best Value: 5763
Iteration 6,   Time: 2.86 sec,   Best Value: 5763
Iteration 7,   Time: 2.75 sec,   Best Value: 5763
Iteration 8,   Time: 2.78 sec,   Best Value: 5763
Iteration 9,   Time: 2.80 sec,   Best Value: 5779
Iteration 10,  Time: 2.78 sec,   Best Value: 5779
Лучшее решение:
 [39, 64, 1, 85, 90, 82, 29, 81, 30, 88, 54, 9, 67, 61, 28]
Общая ценность: 5779
Общее время выполнения: 28.42 секунд
```

```
----- Процесс 1  Решение группы - 4720
----- Процесс 0  Решение группы - 4876
----- Процесс 3  Решение группы - 4677
----- Процесс 2  Решение группы - 4618
Итерация 1,   Решение: 4876,   t=2.48 сек
----- Процесс 1  Решение группы - 5280
----- Процесс 0  Решение группы - 4868
----- Процесс 2  Решение группы - 5234
----- Процесс 3  Решение группы - 5198
Итерация 2,   Решение: 5280,   t=1.91 сек
          .   .   .
----- Процесс 1  Решение группы - 5754
----- Процесс 0  Решение группы - 5730
----- Процесс 2  Решение группы - 5762
----- Процесс 3  Решение группы - 5737
Итерация 10,   Решение: 5779,   t=1.27 сек
Лучшее решение:
 [39, 1, 54, 61, 82, 30, 81, 29, 88, 64, 67, 9, 90, 85, 28];
Общая ценность: 5779
Общее время выполнения: 15.22 секунд
```

Рис. 1. *Одно- и четырехпоточное решение задачи о рюкзаке*

Fig. 1. *One- and four-stream solution of the knapsack problem*

При этом надо отметить, что первые итерации занимают больше времени, чем последующие. Связано это с тем, что на старте необходимо создать и настроить пул процессов, что может занимать больше времени. Начальные затраты на запуск и распределение задач могут быть большими, но с течением времени эффективность обработки улучшается. В многопроцессорной обработке время на выполнение первых итераций может быть больше из-за инициализации и «разогрева» процессов (рис. 2).

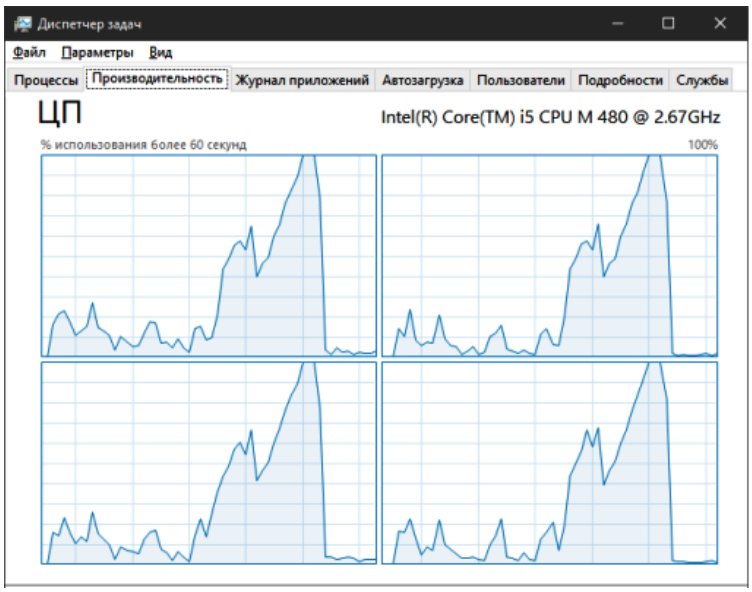

Рис. 2. *Загрузка 4 ядер при решении задачи о рюкзаке*

Fig. 2. *Loading of 4 cores when solving the knapsack problem*

С целью иллюстрации на рисунке 3 представлены результаты работы каждой из четырех групп муравьев на каждой из итераций, а также характер поиска наилучшего решения с помощью муравьиного алгоритма.

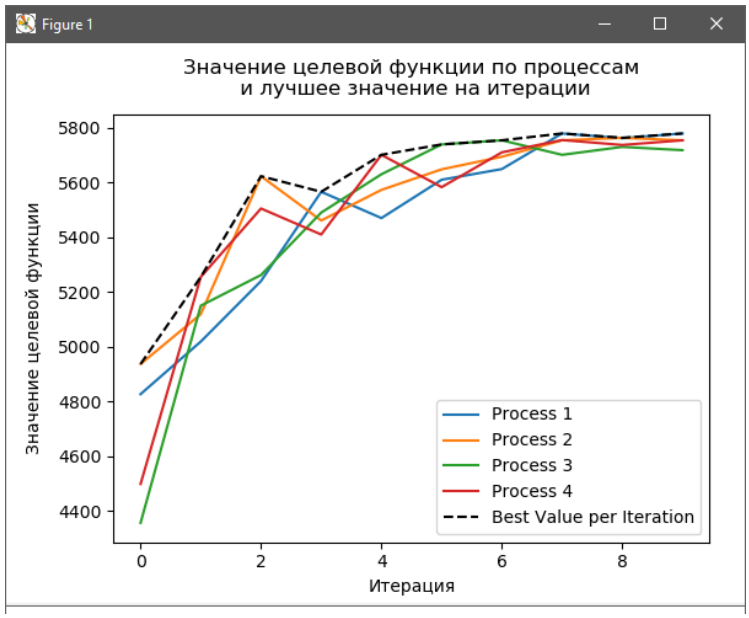

Рис. 3. *Задача о рюкзаке: 100 предметов, 10 итераций и 100 муравьев*

Fig. 3. *Knapsack problem: 100 items, 10 iterations and 100 ants*

Для более обоснованного вывода по качеству разработанного Python-кода было выполнено дополнительное тестирование на увеличенном наборе данных. Цель тестирования на наборе из 1000 предметов со 100 муравьями и 20 итерациями заключается в следующем:

- Проверка масштабируемости – исследование, как алгоритм муравьиной колонии справляется с задачами большей сложности и размера.

• Валидация результатов – тестирование на больших наборах данных позволяет убедиться в устойчивости и надежности алгоритма, выявить проблемы, которые не очевидны на меньших наборах данных.

В табл. 1 приведен анализ работы параллельного и простого алгоритмов муравьиной колонии при решении задачи о рюкзаке с увеличенным до 1000 набором предметов, который выполнили 100 муравьев за 20 итераций.

Таблица 1. Сравнительный анализ работы параллельного и простого ACO

Table 1. Comparative analysis of the operation of parallel and simple ACO

100 муравьев, 20 итераций, 1000 предметов	
Параллельный алгоритм ACO	Простой алгоритм ACO
Iteration 1, Best Value: 4724, Time: 22.62 сек. Iteration 2, Best Value: 4848, Time: 19.08 сек. Iteration 3, Best Value: 5035, Time: 20.03 сек. . . . . . Iteration 18, Best Value: 7000, Time: 23.97 сек. Iteration 19, Best Value: 7000, Time: 19.40 сек. Iteration 20, Best Value: 7000, Time: 18.45 сек. Лучшее найденное решение: [718, 957, 918, 538, 190, 646, 639, 200, 533, 337, 520, 318, 70, 797, 759] с общей ценностью 7000 Общее время выполнения: 397.23 секунд	Iteration 1, Time: 70.61 сек. Best Value: 4867 Iteration 2, Time: 74.44 sec, Best Value: 5027 Iteration 3, Time: 70.47 сек. Best Value: 5174 . . . . . Iteration 18, Time: 69.23 сек. Best Value: 6972 Iteration 19, Time: 68.30 сек. Best Value: 6972 Iteration 20, Time: 68.71 сек. Best Value: 6972 Лучшее найденное решение: [318, 918, 639, 823, 957, 190, 646, 249, 70, 759, 797, 538, 819, 337, 522] с общей ценностью 6972 Общее время выполнения: 1414.94 секунд

Из анализа полученных результатов следует, что параллельный алгоритм ACO значительно быстрее. Общее время выполнения составило 397 секунд по сравнению с 1415 секундами у простого алгоритма. То есть распараллеливание существенно сократило время выполнения задачи, особенно для больших наборов данных. Кроме этого, время выполнения одной итерации у параллельного алгоритма занимает от 18.45 до 23.97 секунд, что значительно меньше по сравнению с от 68.30 до 74.44 секундами у простого алгоритма. Это показывает более равномерную и эффективную обработку задач в параллельной реализации.

Параллельный алгоритм нашел решение в 7000 единиц стоимости, которое превосходит решение, найденное простым алгоритмом (6972). Это также свидетельствует о том, что распараллеливание не только ускоряет процесс, но и находит более качественные решения за разумное время.

На скриншоте диспетчера задач, который был получен во время работы ACO с 1000 предметами (рис. 4), видна равномерная загрузка всех четырех ядер процессора. Из чего можно сделать следующие выводы:

• Параллельный алгоритм эффективно распределяет вычислительную нагрузку между всеми доступными процессорами, что подтверждает правильность и эффективность распараллеливания.

• Равномерная загрузка процессоров свидетельствует о том, что параллельный алгоритм использует все доступные ресурсы системы, это приводит к значительному ускорению вычислений по сравнению с последовательным (простым) алгоритмом, т. е. улучшена производительность.

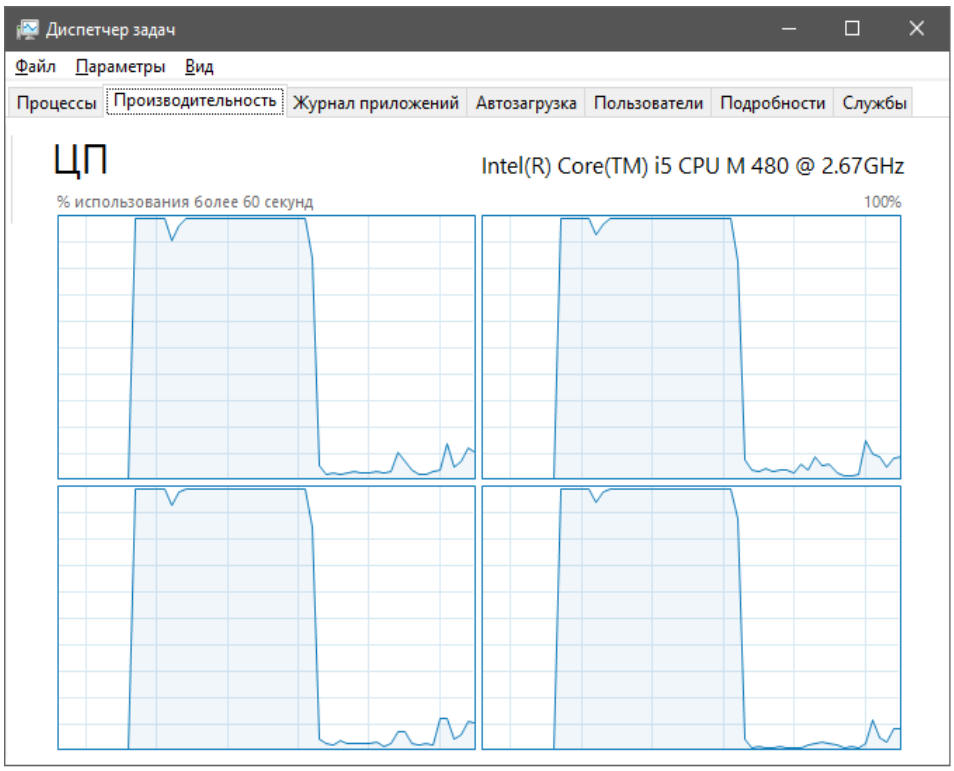

Рис. 4. *Загрузка 4 ядер процессора при 1000 предметах*

Fig. 4. *Loading of 4 processor cores with 1000 items*

Все это подтверждает эффективность использования параллельных вычислений для задач с высокой вычислительной сложностью.

Переход от последовательного выполнения к параллельному логично вытекает из понимания независимости муравьев в АСО и открывает новые возможности для решения сложных задач. Проведенное тестирование показало, что АСО может быстро находить решения, близкие к оптимальным, даже при малом количестве итераций. Это делает его полезным инструментом в условиях, когда время расчета играет критическую роль, и позволяет его использовать при решении широкого круга траекторных задач [4, 5, 6] или задач маршрутизации в беспроводных сетях [7]. По сравнению с точными методами типа динамического программирования АСО показал значительное сокращение времени выполнения при приемлемом уровне отклонения от оптимального решения.

Выводы

В качестве основных выводов по реализации процесса распараллеливания алгоритма муравьиной колонии на примере задачи о рюкзаке при помощи языка программирования Python можно отметить следующие положительные направления:

1. Распараллеливание позволяет одновременно запускать множество муравьев на разных процессорах, что значительно сокращает время поиска оптимального решения по сравнению с последовательным выполнением алгоритма.

2. При использовании распараллеливания алгоритм максимально задействует доступные вычислительные ресурсы (ядра процессора), что особенно важно при работе с большими наборами данных или сложными проблемами, требующими значительных вычислительных ресурсов.

3. Распараллеливание позволяет исследовать большее количество возможных решений за определенное время, что приводит к повышению вероятности нахождения более точного решения по сравнению с последовательным алгоритмом.

4. Распараллеливание делает алгоритм муравьиной колонии более устойчивым к случайным колебаниям в поведении муравьев. Благодаря тому, что муравьи работают независимо друг от друга на разных процессорах, ошибки одного муравья не влияют на других.

5. Библиотека multiprocessing в Python позволяет эффективно распараллелить алгоритм муравьиной колонии, что делает распараллеливание доступным для разработчиков любого уровня.

СПИСОК ЛИТЕРАТУРЫ

1. *Даринцев О. В., Мигранов А. Б.* Использование муравьиного алгоритма для поиска стратегии поведения группы мобильных роботов на рабочем поле с препятствиями // Многофазные системы. 2022. Т. 17. № 3–4. С. 177–186. DOI: 10.21662/mfs2022.3.016

2. *Минин А. А., Немтинов В. А.* Применение алгоритма муравьиных колоний для создания технологических процессов обработки резанием // Инженерные технологии. 2023. № 3. С. 31–36. EDN: LAMZQA

3. *Павловская К. А., Червинский В. В*. Маршрутизация в сетях MANET на основе муравьиных алгоритмов с учетом энергосбережения // Вестник Донецкого национального университета. Серия Г: Технические науки. 2023. № 1. С. 4–10. EDN: SWIUGB

4. *Вагизов М. Р., Хабаров С. П.* Построение программных траекторий движения на базе решения задачи «Машина Дубинса» // Информация и космос. 2021. № 3. С. 116–125. EDN: DDDWFN

5. *Коренев А. С.,* Скрыпка А. С., Хабаров С. П. Автономное судовождение на действующих судах // Морской вестник. 2022. № 1(81). С. 92–95. EDN: MIPIOP

6. *Хабаров С. П., Шилкина М. Л.* Геометрический подход к решению задачи для машин Дубинса при формировании программных траекторий движения // Научно-технический вестник информационных технологий, механики и оптики. 2021. Т. 21. № 5. С. 653–663. DOI: 10.17586/2226-1494-2021-21-5-653-663

7. *Думов М. И.* Моделирование беспроводных сетей в среде OMNeT++ с использованием INET framework // Научно-технический вестник информационных технологий, механики и оптики. 2019. Т. 19. № 6. С. 1151–1161. DOI: 10.17586/2226-1494-2019-19-6-1151-1161

REFERENCES

1. Darintsev O.V., Migranov A.B. Using the ant algorithm to search for a strategy for the behavior of a group of mobile robots on a work field with obstacles. *Multiphase systems*. 2022. Vol. 17. No. 3–4. Pp. 177–186. DOI: 10.21662/mfs2022.3.016. (In Russian)

2. Minin A.A., Nemtinov V.A. Application of the ant colony algorithm for creating technological processes of cutting. *Engineering technologies*. 2023. No. 3. Pp. 31–36. EDN: LAMZQA. (In Russian)

3. Pavlovskaya K.A., Chervinsky V.V. Routing in MANET networks based on ant algorithms taking into account energy saving. *Bulletin of Donetsk National University. Series G: Technical Sciences*. 2023. No. 1. Pp. 4–10. EDN: SWIUGB. (In Russian)

4. Vagizov M.R., Khabarov S.P. Constructing program trajectories of motion based on solving the Dubins Machine problem. *Information and Space*. 2021. No. 3. Pp. 116–125. EDN: DDDWFN. (In Russian)

5. Korenev A.S., Skrypka A.S., Khabarov S.P. Autonomous navigation on operating vessels. *Marine Bulletin*. 2022. № 1(81). Pp. 92–95. EDN: MIPIOP. (In Russian)

6. Khabarov S.P., Shilkina M.L. Geometric approach to solving the problem for Dubins machines in the formation of program trajectories of motion. *Bulletin of information technologies, mechanics and optics*. 2021. Vol. 21. No. 5. Pp. 653–663. DOI: 10.17586/2226-1494-2021-21-5-653-663. (In Russian)

7. Dumov M.I. Modeling of wireless networks in the OMNeT++ environment using the INET framework. *Scientific and Technical Bulletin of Information Technologies, Mechanics and Optics*. 2019. Vol. 19. No. 6. Pp. 1151–1161. DOI: 10.17586/2226-1494-2019-19-6-1151-1161. (In Russian)

Вклад авторов: все авторы сделали эквивалентный вклад в подготовку публикации. Авторы заявляют об отсутствии конфликта интересов.

Contribution of the authors: the authors contributed equally to this article. The authors declare no conflicts of interests.

Финансирование. Исследование проведено без спонсорской поддержки.

Funding. The study was performed without external funding.

Информация об авторах

Вагизов Марсель Равильевич, канд. тех. наук, заведующий кафедрой информационных систем и технологий, Санкт-Петербургский государственный лесотехнический университет им. С. М. Кирова;
194021, Россия, Санкт-Петербург, Институтский пер., 5;
bars-tatarin@yandex.ru, ORCID: https://orcid.org/0000-0003-4848-1619, SPIN-код: 4811-8943

Хабаров Сергей Петрович, канд. тех. наук, доцент кафедры информационных систем и технологий, Санкт-Петербургский государственный лесотехнический университет им. С. М. Кирова;
194021, Россия, Санкт-Петербург, Институтский пер., 5;
Serg.Habarov@mail.ru, ORCID: https://orcid.org/0000-0003-1337-0150, SPIN-код: 4365-2033

Information about the authors

Marsel R. Vagizov, Candidate of Technical Sciences, Head of the Department of Information Systems and Technologies, Saint Petersburg State Forest Engineering University named after S.M. Kirov;
194021, Russia, Saint Petersburg, 5 Institutsky lane;
bars-tatarin@yandex.ru, ORCID: https://orcid.org/0000-0003-4848-1619, SPIN-code: 4811-8943

Sergey P. Khabarov, Candidate of Technical Sciences, Associate Professor of the Department of Information Systems and Technologies, St. Petersburg State Forest Engineering University named after S.M. Kirov;
194021, Russia, Saint Petersburg, 5 Institutsky lane;
Serg.Habarov@mail.ru, ORCID: https://orcid.org/0000-0003-1337-0150, SPIN-code: 4365-2033