Сравнение алгоритмов MCTS, MCDDQ, MCDDQ-SA, Greedy в рамках задачи параллельного планирования загрузки машин на производстве

А.А. Абузяров, А.А. Макаров

Российский государственный университет имени А.Н. Косыгина (Технологии. Дизайн. Искусство)

Аннотация: В данной статье рассматривается проблема планирования задач в производственных системах с несколькими машинами, работающими параллельно. Предложены четыре подхода к решению этой задачи: чистый метод Монте-Карло с поиском по дереву (Monte Carlo Tree Search, MCTS), гибридный агент MCDDQ, сочетающий обучение с подкреплением на основе Double Deep Q-Network (DDQN) и Monte Carlo Tree Search (MCTS), усовершенствованный агент MCDDQ-SA, интегрирующий алгоритм имитации отжига (Simulated Annealing, SA) для повышения качества решений, а также жадный алгоритм (Greedy). Разработана модель среды, учитывающая скорости машин, длительности задач. Проведено сравнительное исследование эффективности методов на основе метрик makespan (максимальное время завершения) и idle time (время простоя). Результаты демонстрируют, что MCDDQ-SA обеспечивает наилучший баланс между качеством планирования и вычислительной эффективностью за счет адаптивного исследования пространства решений. Представлены аналитические инструменты для оценки динамики алгоритмов, что подчеркивает их применимость в реальных производственных системах. Статья предлагает новые перспективы для применения гибридных методов в задачах управления ресурсами.
Ключевые слова: машинное обучение, Q-обучение, глубокие нейронные сети, MCTS, DDQN, имитация отжига, планирование, жадный алгоритм

Планирование задач на технологических машинах является одной из ключевых задач в области управления ресурсами и оптимизации производственных процессов в промышленных системах [1]. В условиях неоднородности скоростей машин, характерных для производственных линий, таких как сборочные цеха, предприятия по переработке материалов или пищевые производства (например, линии по упаковке или переработке продукции) и необходимости минимизации общего времени выполнения (makespan) и времени простоя (idle time) традиционные подходы часто оказываются недостаточно эффективными. Данная статья посвящена исследованию гибридных методов, сочетающих машинное обучение и алгоритмы оптимизации, для решения задачи параллельного планирования. Мы предлагаем четыре подхода: чистый метод Монте-Карло с поиском по

дереву (MCTS), агент MCDDQ на основе Double Deep Q-Network (DDQN) и MCTS, жадный алгоритм (Greedy), а также созданный нами усовершенствованный агент MCDDQ_SA с интеграцией имитации отжига (Simulated Annealing). Цель исследования — разработка и сравнение этих методов для достижения оптимального распределения задач с учетом сбалансированности нагрузки и вычислительной эффективности. Результаты демонстрируют потенциал гибридных подходов для применения в современных системах планирования.

Объект управления

Проблемы параллельного планирования машин (Parallel Machine Scheduling Problems) — это класс задач в теории оптимизации и операционном исследовании, в которых необходимо эффективно распределить набор рабочих заданий (процессов, операций) между несколькими параллельными машинами с целью минимизации некоторой функции, связанной с временем выполнения или другими критериями [2].

Особенности задачи:

- Задача является NP-трудной [3], так как представляет собой комбинаторную оптимизацию распределения задач по машинам с учетом их скоростей.
- Неоднородность скоростей машин требует поиска компромисса между назначением задач на более быстрые машины и равномерным распределением нагрузки.
- Решение задачи направлено на достижение краткосрочной эффективности (быстрое завершение задач) и долгосрочной сбалансированности (минимизация простоя).

Задача параллельного планирования (модель среды), реализованная в данной работе, заключается в оптимальном распределении набора задач по

набору технологических машин с учетом их неоднородных характеристик. Целью является минимизация общего времени завершения всех задач (makespan) и времени простоя машин (idle time), а также обеспечение сбалансированной нагрузки между машинами.

Формальная постановка

- **Входные данные**:
  - Множество технологических машин $M = \{M1, M2, \ldots, Mn\}$, где n — количество машин.
  - Набор производственных операций (задач) $T = \{T1, T2, \ldots, Tn\}$, где n — количество операций.
  - Вектор скоростей машин $S = [S1, S2, \ldots, Sn]$, где S — скорость обработки операций на машинах M.
  - Вектор длительностей операций $D = [D1, D2, \ldots, Dt]$, где d — базовая длительность операции T.
- **Ограничения**:
  - Каждая операция T должна быть назначена ровно на одну машину M.
  - Время обработки операции T на машине M вычисляется как $t_{ij} = d_j / s_i$.
  - Машина может обрабатывать только одну операцию одновременно, но операции назначаются последовательно, и время работы машины аккумулируется.
- **Целевая функция**:
  - Основная цель: минимизация makespan,
    $makespan = max_{i \in M} t_i$,
    где t_i — общее время работы машины M.

- Дополнительная цель: минимизация времени простоя (idle time), вычисляемого как:
  $idle\_time = \sum_{i=1}^{n_m} \max(0, makespan - t_i)$,
  где:
  - $makespan - t_i$ — время, в течение которого машина i простаивает после завершения своих задач до достижения общего времени выполнения ($makespan$).
- Балансировка нагрузки: минимизация дисперсии времени работы машин.

Награда

- Промежуточная награда (в коде: intermediate_reward):
- $intermediate\_reward = -0{,}05 * load\_variance - 0{,}1 * current\_makespan$
  - где $load\_variance$ — дисперсия нагрузки, $current\_makespan$ — текущий максимум времени работы.
- Итоговая награда (при завершении всех назначений):
- $reward = -makespan - 3{,}0 * idle\_time - 0{,}1 * load\_variance$.

Модель среды реализована по стандарту OpenAI Gym и представляет собой программную реализацию описанной задачи планирования, предназначенную для обучения и тестирования алгоритмов. Она включает состояние, отражающее текущее распределение операций по машинам и времена их работы, а также действия, определяющие назначение очередной операции на выбранную машину. Работа модели начинается с

инициализации, где все операции не назначены, а времена работы машин равны нулю. На каждом шаге алгоритм назначает операцию на машину, обновляя состояние и вычисляя метрики (makespan, idle time, дисперсию нагрузки). Модель предоставляет награды для оптимизации, поддерживает визуализацию текущего состояния и позволяет интегрировать алгоритмы обучения, такие как DDQN и MCTS, для сравнительного анализа их эффективности в условиях производственных систем с неоднородными машинами.

Алгоритм MCTS

Метод Монте-Карло с поиском по дереву (Monte Carlo Tree Search, MCTS) — это эвристический алгоритм поиска, который сочетает случайные симуляции с построением дерева решений для выбора оптимального действия в задачах с большим пространством состояний. MCTS состоит из четырех основных этапов: выбор (Selection), расширение (Expansion), симуляция (Simulation) и обратное распространение (Backpropagation). Для выбора наиболее перспективных узлов в этапе выбора часто используется критерий UCB (Upper Confidence Bound), который балансирует между исследованием (exploration) и эксплуатацией (exploitation) [4].

Формула UCB записывается как:

$$UCB = Q + c * \sqrt{\frac{\ln(N)}{n}}, \quad (1)$$

где:

- Q — средняя награда узла (оценка качества действия),
- N — общее количество посещений родительского узла,
- n — количество посещений текущего узла,

- c — коэффициент исследования (exploration weight), регулирующий баланс между исследованием и эксплуатацией.

Алгоритм итеративно строит дерево, начиная с корневого узла, представляющего текущее состояние, и на каждой итерации выбирает действие, которое максимизирует UCB, пока не достигается конечное состояние или не исчерпывается бюджет итераций [5].

Greedy

Описание: Жадный алгоритм (Greedy) назначает задачи на машины, минимизируя текущее время завершения для каждой задачи [6].

Процесс назначения:

- Для каждой задачи (в порядке от 0 до num_tasks-1):
  - Перебираются все машины (machine_idx от 0 до num_machines-1)
  - Вычисляется время завершения задачи на каждой машине:
  - $t_m = machine\_times[machine\_idx] + \frac{task\_duration[i]}{machine\_speeds[m]}$

  где:
    - $machine\_times[machine\_idx]$ — текущее время работы машины.
    - $task\_durations[task\_idx]$ — длительность задачи.
    - $machine\_speeds[machine\_idx]$ — скорость машины.
  - Выбирается машина с минимальным временем завершения.
  - Задача назначается на эту машину (env.step((task_idx, best_machine))).

Алгоритм MCDDQ

MCDDQ (MCTS with DDQN) — это гибридный алгоритм, разработанный для решения задач оптимального планирования, таких как распределение задач на параллельных машинах. Он сочетает обучение с подкреплением на основе Double Deep Q-Network (DDQN) с методом Монте-Карло с поиском по дереву (MCTS). Основная идея заключается в использовании предварительно обученной нейронной сети для оценки действий (Q-значений), которая затем направляет процесс поиска MCTS, заменяя случайные симуляции на целенаправленное исследование пространства решений. Такой подход позволяет эффективно балансировать между вычислительной сложностью и качеством получаемого расписания [7].

Алгоритм MCDDQ состоит из двух ключевых компонентов:

1. **Обучение с подкреплением (DDQN)**:
    - Используется для построения модели, которая предсказывает долгосрочную ценность действий в заданном состоянии.
    - Нейронная сеть обучается на основе опыта, собранного в среде, минимизируя функцию потерь с учетом двойной оценки Q-значений (Double Q-Learning) [8].
2. **MCTS с эвристикой**:
    - Применяет обученную DDQN-сеть как эвристику для выбора действий вместо случайных симуляций, что ускоряет сходимость к оптимальному решению.
    - Использует критерий UCB для выбора перспективных узлов в дереве поиска.

После обучения агента DDQN, на этапе выполнения (поиска решения) алгоритм использует MCTS для выбора оптимального действия в текущем состоянии:

- **Выбор и расширение**: для каждого rollout из заданного числа (например, max_rollouts = 1000) определяется список доступных действий (неназначенная задача и машина). Действие выбирается по критерию UCB формула (1), где в данном случае Q является предсказанием Q-сети
- **Симуляция**: Копия среды (copy.deepcopy(env)) используется для выполнения действий до завершения всех назначений. Награда аккумулируется как сумма промежуточных наград среды.
- **Обратное распространение**: Статистика посещений (root_visits) и средняя награда (root_q) обновляются для всех узлов в пути симуляции.
- **Итоговое действие:** выбирается действие с максимальной средней наградой в корневом состоянии.

Алгоритм MCDDQ_SA

MCDDQ_SA (MCTS with DDQN and Simulated Annealing) — это усовершенствованный гибридный алгоритм, созданный нами для оптимизации задач планирования, таких как распределение задач на параллельных машинах. Он расширяет базовый алгоритм MCDDQ, интегрируя обучение с подкреплением на основе Double Deep Q-Network (DDQN) с приоритизированным буфером опыта [9], метод Монте-Карло с поиском по дереву (MCTS) и алгоритм имитации отжига (Simulated Annealing, SA). Основное новшество заключается в добавлении SA для адаптивного исследования пространства решений, что позволяет избежать локальных минимумов и улучшить качество расписания при сохранении эффективности.

MCDDQ_SA объединяет три подхода:

1. **Обучение с подкреплением (DDQN)**: Нейронная сеть обучается для предсказания Q-значений действий, предоставляя эвристическую основу для поиска.
2. **MCTS**: использует Q-сеть для направленного поиска по дереву решений с применением критерия UCB.
3. **Simulated Annealing (SA)**: добавляет стохастический механизм выбора действий, зависящий от температуры, для баланса между исследованием и эксплуатацией [10].

После обучения агента DDQN, на этапе выполнения (поиска решения) алгоритм работает следующим образом:

- **Monte Carlo Tree Search (MCTS)**:
  - MCTS выполняет симуляции (rollouts) для оценки действий, балансируя исследование и эксплуатацию с помощью формулы Upper Confidence Bound (UCB):

$$\mathrm{UCB} = 0.7 * \mathrm{Q(s.a)} + 0.3 * \mathrm{E(s,a)} + w_{exp} * \sqrt{\frac{lnN(s)}{N(s,a)}}$$

  - где $\boldsymbol{Q}(s,a)$ — Q-значение от DDQN;
  - $\boldsymbol{E}(\boldsymbol{s},\boldsymbol{a})$ — энергия действия от SA;
  - $\boldsymbol{N}(\boldsymbol{s})$ и $\boldsymbol{N}(\boldsymbol{s},\boldsymbol{a})$ — количество посещений состояния и действия;
  - $w_{exp} = 30 * (1 - step/num\_tasks)$ — адаптивный вес исследования.
- **Simulated Annealing (SA)**:

- SA вводит вероятностный выбор действий на основе энергии, которая учитывает текущую загрузку машин, баланс нагрузки и длительность задач [11]. Энергия действия $\boldsymbol{E(s,a)}$ вычисляется, как:

$$E(s,a) = -machine\_times[m] + 10 * load\_balance$$

где:

- $machine\_times[m]$ — текущее время работы машины, на которую назначается задача. Меньшее время предпочтительнее, поэтому используется отрицательный знак;
- $load\_balance$ — метрика баланса нагрузки, рассчитанная как: $load\_balance = -var(machine\_times + [machine\_times[machine\_idx] + \frac{task\_durations[task\_idx]}{machine\_speeds[machine\_idx]}])$

  Здесь var — дисперсия загрузки машин после гипотетического назначения задачи. Дисперсия вычисляется для массива, где время на машине $machine\_idx$ увеличивается на время обработки задачи $\frac{task\_durations[task\_idx]}{machine\_speeds[machine\_idx]}$. Отрицательная дисперсия используется, чтобы предпочтение отдавалось назначениям, минимизирующим неравномерность загрузки.

- Вероятность выбора действия a определяется по формуле:

$$P(a) = \frac{\exp(-E(a)/T)}{\sum_{a' \in A} \exp(-E(a'/T))}$$

где:

- $E(a)$ — энергия действия a.

- T — текущая температура, изначально равная 2000, с минимальным значением 0.5.
- A — множество доступных действий (неназначенные задачи и доступные машины).

- $\exp(-E(a)/T)$ — обеспечивает более высокую вероятность для действий с меньшей энергией, но при высокой температуре допускает выбор менее оптимальных действий, способствуя исследованию.
- Нормировка $(\sum_{a' \in A} \exp(-E(a'/T)))$ гарантирует, что P(a) является корректным распределением вероятностей.

- Назначение вероятностей: P(a) используется для вероятностного выбора действий в SA-режиме. Когда влияние SA велико (высокая температура или $w_{sa} > 0.05$), действия выбираются пропорционально P(a), что позволяет алгоритму исследовать альтернативные решения. При низкой температуре вероятности становятся почти равномерными ($P(a) \approx 1/|A|$), и выбор действия больше зависит от UCB.

- Температура уменьшается с адаптивной скоростью охлаждения:

$$cooling\_rate = 0.995 - 0.02 * \frac{current\_step}{num\_tasks}$$

где current_step — количество назначенных задач.

- При отсутствии улучшений makespan в течение 20 итераций температура сбрасывается до $0.5 * T_{initial}$, что предотвращает застревание в локальных минимумах.
- **Гибридный выбор действий**:

- Действия выбираются с учетом веса SA $(w_{sa}) = \max(0.3, T/T_{initial})$:
  - Если $T > 0.7 * T_{initial}$ или с вероятностью 0.01, действие выбирается вероятностно по P(a) (SA-режим).
  - Иначе выбирается действие с максимальным комбинированным значением:

$$(1 - w_{sa}) * UCB + w_{sa} * P(a)$$

Это обеспечивает баланс между детерминированным выбором (UCB) и вероятностным (SA).

- Лучшее действие определяется на основе среднего значения наград, скорректированного бонусом SA $(0.2 * P(a))$, что дополнительно учитывает вероятности SA при обновлении статистики MCTS.

Тестирование алгоритмов

Тестирование алгоритмов планирования задач на технологических машинах (MCDDQ, MCTS, MCDDQ_SA и Greedy) проводилось в два этапа с различными конфигурациями среды для оценки их производительности по метрикам makespan (максимальное время завершения), idle time (время простоя) и времени выполнения. Результаты визуализированы с использованием коробчатой диаграммы (диаграмма типа «ящик с усами»), введенной Дж. Тьюки для быстрого способа визуализации распределения данных.

Этапы тестирования

1. **Первый этап: 3 машины, 5 задач**

- Среда инициализировалась с параметрами: 3 машины с произвольными скоростями и 5 задач с заданными длительностями. Это небольшой набор данных позволил оценить базовую эффективность алгоритмов в упрощенной конфигурации.
- Каждый алгоритм запускался 5 раз для получения статистически значимых данных. Использовались фиксированные значения скоростей машин и длительностей задач.
- machine_speeds = [1.0, 0.8, 1.2] – скорости машин.
- task_durations = [10, 20, 30, 40, 50]) – длительности задач.

2. **Второй этап: 10 машин, 20 задач**

- Среда масштабировалась до 10 машин и 20 задач, что отражает более сложный сценарий, требующий эффективного распределения и балансировки нагрузки.
- Аналогично первому этапу, каждый алгоритм тестировался 5 раз для сбора статистических данных, с учетом случайных вариаций в начальных условиях
- machine_speeds = [1.28, 0.79, 1.35, 1.31, 0.73, 1.25, 0.53, 0.79, 1.46, 0.66] – скорости машин.
- task_durations = [76.54, 29.02, 84.37, 79.23, 23.56, 71.99, 12.44, 29.03, 91.98, 20.32, 55.85, 89.91, 37.37, 48.92, 91.36, 47.27, 32.85, 70.48, 87.38, 16.12] – длительности задач.

Методология тестирования

- Для каждого теста среда сбрасывалась (env.reset()), а алгоритмы применялись последовательно: MCDDQ, MCTS, MCDDQ_SA и Greedy.

- Метрики собирались после завершения каждого запуска: makespan (максимальное время работы машин), idle time (суммарное время простоя машин) и время выполнения (runtime).
- Для оценки стабильности результатов вычислялись средние значения и стандартные отклонения по 5 запускам.
- Результаты визуализировались с помощью коробчатой диаграммы («ящик с усами»), где:
    - Средняя линия в коробчатой диаграмме указывает медиану.
    - Границы ящика — 25-й и 75-й процентили.
    - «Усы» показывают минимальные и максимальные значения в пределах 1.5 межквартильного размаха.
    - Точки за пределами «усов» — выбросы.

Результаты тестирования

3 машины, 5 задач

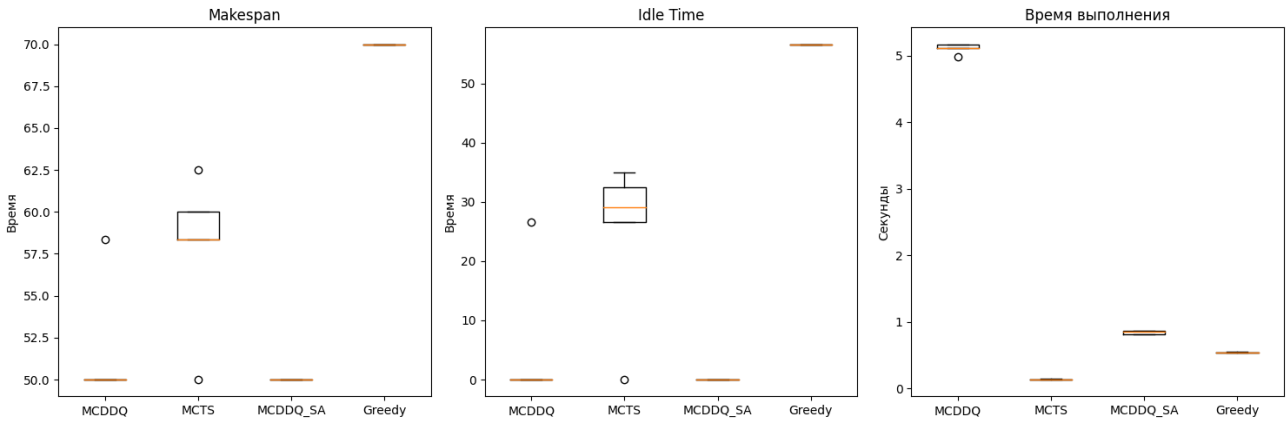

Рис. 1. – Boxplot результата тестирования этапа 1

MCDDQ (DDQN + MCTS):

- Средний Makespan: 51.67 ± 3.33
- Средний Idle Time: 5.33 ± 10.67
- Среднее время выполнения: 5.11 сек

MCTS:

- Средний Makespan: 57.83 ± 4.20
- Средний Idle Time: 24.67 ± 12.66
- Среднее время выполнения: 0.14 сек

MCDDQ_SA (DDQN + MCTS + SA):

- Средний Makespan: 50.00 ± 0.00
- Средний Idle Time: 0.00 ± 0.00
- Среднее время выполнения: 0.84 сек

Greedy:

- Makespan: 70.00 ± 0.00
- Idle Time: 56.67 ± 0.00
- Среднее время выполнения: 0.54 сек

10 машин, 20 задач

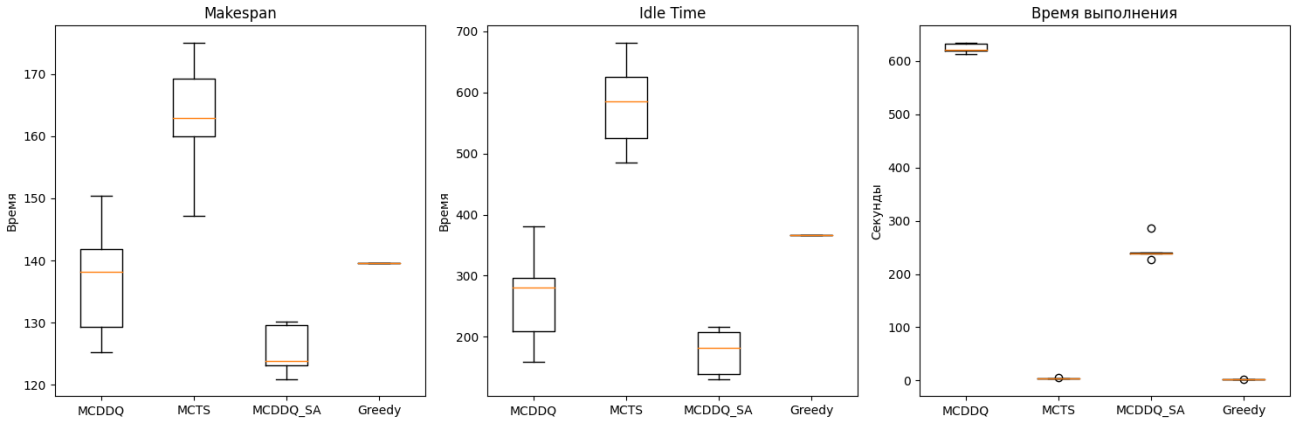

Рис. 2. – Boxplot результата тестирования этапа 2

MCDDQ (DDQN + MCTS):

- Средний Makespan: 136.99 ± 8.95
- Средний Idle Time: 265.00 ± 76.29
- Среднее время выполнения: 624.46 сек

MCTS:

- Средний Makespan: 162.88 ± 9.40
- Средний Idle Time: 580.76 ± 69.72
- Среднее время выполнения: 4.52 сек

MCDDQ_SA (DDQN + MCTS + SA):

- Средний Makespan: 125.52 ± 3.66
- Средний Idle Time: 174.96 ± 35.24
- Среднее время выполнения: 246.40 сек

Greedy:

- Makespan: 139.55 ± 0.00
- Idle Time: 366.72 ± 0.00
- Среднее время выполнения: 2.15 сек

Анализ графиков

- **Makespan**: Для 3 машин и 5 задач MCDDQ_SA показывает наименьший средний makespan (50.00), что значительно лучше, чем у Greedy (70.00). Для 10 машин и 20 задач MCDDQ_SA снова лидирует (125.52), демонстрируя меньшую вариабельность (3.66) по сравнению с MCDDQ (8.95) и MCTS (9.40). Greedy сохраняет стабильность, но с высоким значением (139.55).
- **Idle Time**: MCDDQ_SA минимизирует idle time до 0.00 для малого набора и 174.96 для большого, что указывает на эффективное распределение задач. Greedy демонстрирует наибольший idle time (56.67 и 366.72), особенно при увеличении масштаба.
- **Время выполнения**: Greedy и MCTS оказываются самыми быстрыми (0.54 и 0.14 сек для малого набора, 2.15 и 4.52 сек для большого), тогда

как MCDDQ и MCDDQ_SA требуют больше времени (5.11 и 0.84 сек, 624.46 и 246.40 сек соответственно) из-за сложных вычислений MCTS и SA.

Выводы

- MCDDQ_SA демонстрирует наилучший баланс между качеством (минимизация makespan и idle time) и стабильностью решений, особенно в больших наборах данных (10 машин, 20 задач), где его преимущество наиболее заметно.
- Greedy обеспечивает минимальное время выполнения, но значительно уступает по качеству, что делает его менее подходящим для сложных сценариев.
- MCTS и MCDDQ показывают средние результаты, но проигрывают MCDDQ_SA в оптимизации makespan и idle time.
- Диаграммы boxplot подтверждают, что MCDDQ_SA имеет наименьшую вариабельность и лучшие медианные значения, что подчеркивает его применимость в реальных системах с высокой нагрузкой.

В данной работе проведено исследование методов планирования задач на технологических машинах, предложены и протестированы четыре алгоритма: жадный алгоритм (Greedy), метод Монте-Карло с поиском по дереву (MCTS), гибридный агент MCDDQ (Double Deep Q-Network + MCTS) и усовершенствованный агент MCDDQ-SA (с добавлением имитации отжига, SA). Разработанная модель среды учитывает скорости машин и длительности задач, обеспечивая гибкую основу для сравнительного анализа.

Результаты тестирования, проведенного на двух наборах данных (3 машины, 5 задач и 10 машин, 20 задач), демонстрируют, что MCDDQ-SA превосходит остальные методы по ключевым метрикам: минимизации makespan (максимального времени завершения) и idle time (времени простоя), а также по стабильности решений, что подтверждено диаграммами boxplot. В частности, MCDDQ-SA показал средний makespan 50.00 для малого набора и 125.52 для большого, а также минимизировал idle time до 0.00 и 174.96 соответственно, что значительно лучше, чем у Greedy (70.00 и 139.55 для makespan, 56.67 и 366.72 для idle time). Однако этот алгоритм требует больше времени выполнения (0.84 сек и 246.40 сек), что связано с вычислительной сложностью MCTS и SA. Greedy оказался самым быстрым (0.54 сек и 2.15 сек), но продемонстрировал наименее эффективные результаты по makespan и idle time, что делает его пригодным только для простых задач с минимальными требованиями к оптимизации. MCTS и MCDDQ показали промежуточные результаты, но уступили MCDDQ-SA в качестве решений, особенно при увеличении масштаба задачи.

Проведенное исследование подтвердило адаптивность и эффективность MCDDQ-SA, подчеркивая его способность к балансировке нагрузки и избеганию локальных минимумов за счет интеграции метода имитации отжига. Эти характеристики делают алгоритм перспективным для применения в реальных производственных системах, логистике и управлении ресурсами, где требуется оптимизация времени выполнения и равномерное распределение задач. Дальнейшие исследования могут сосредоточиться на масштабируемости алгоритма для еще более сложных сценариев и его адаптации к динамическим изменениям в реальном времени.

Литература (References)

1. Pinedo. M. Scheduling: Theory, Algorithms, and Systems (5th ed.). Springer, 2016, pp. 105-110.
2. Graham R. L., Lawler E. L., Lenstra J. K., & Rinnooy Kan, A. H. G. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 1979 5, pp. 287-326.
3. Garey, M. R. Computers and Intractability: A Guide to the Theory of NP-Completeness, 1979, pp. 340.
4. Kocsis L. Bandit Based Monte-Carlo Planning. Research Gate URL: researchgate.net/publication/221112399_Bandit_Based_Monte-Carlo_Planning.
5. Naderzadeh Yashar, Grosu Daniel. PPB-MCTS: A novel distributed-memory parallel partial-backpropagation Monte Carlo tree search algorithm, Knowledge Based Systems. URL: sciencedirect.com/science/article/abs/pii/S0743731524001084.
6. Ying Kuo-Ching, Lin Shih-Wei. Minimizing makespan in two-stage assembly additive manufacturing: A reinforcement learning iterated greedy algorithm, Knowledge Based Systems. URL: sciencedirect.com/science/article/abs/pii/S1568494623002089.
7. Lohse Oliver, Haag Aaron. Enhancing Monte-Carlo Tree Search with MultiAgent Deep QNetwork in Open Shop Scheduling. Research Gate URL: researchgate.net/publication/367509729_Enhancing_MonteCarlo_TreeSearch_with_Multi-Agent_Deep_QNetwork_in_Open_Shop_Scheduling.
8. Hasselt H. V., Guez A., Silve D. Deep reinforcement learning with double Q-learning. Research Gate. URL:

researchgate.net/publication/282182152_Deep_Reinforcement_Learning_with_Double_Q-Learning.

9. Schaul, T. Prioritized experience replay. Research Gate. URL: researchgate.net/publication/284219262_Prioritized_Experience_Replay.
10. Kirkpatrick, S. Optimization by simulated annealing. Springer, 1983, pp. 671-680
11. Kosanoglu Fuat, Mahir Atmis. A deep reinforcement learning assisted simulated annealing algorithm for a maintenance planning problem. Research Gate. URL: researchgate.net/publication/35923433A_deep_reinforcement_learning_assisted_simulated_annealing_algorithm_for_a_maintenance_planning_problem.

Дата поступления: 8.04.25
Дата публикации: 25.05.2025