MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 32, NO. 2, 2025
journal homepage: www.mais-journal.ru

DISCRETE MATHEMATICS IN RELATION TO COMPUTER SCIENCE

A Patterning Algorithm for the Dynamic Bin Packing Problem with Placement Groups

E. A. Brazhnikov[1], A. A. Panin[1], A. V. Ratushnyi[1]

DOI: 10.18255/1818-1015-2025-2-110-131

[1]Sobolev Institute of Mathematics of the Siberian Branch of the RAS, Novosibirsk, Russia

MSC2020: 68W25, 68R01
Research article
Full text in Russian

Received March 25, 2025
Revised May 20, 2025
Accepted May 21, 2025

We consider an NP-hard problem of dynamically distributing virtual machines to servers with placement groups. For each virtual machine, parameters such as required number of resources and creation and deletion timestamps are known. Each server is a composition of NUMA nodes and is placed in a rack. Large virtual machines that are placed on two nodes of a single server, and small virtual machines that impose additional conditions on their placement, are considered. Placement groups are associations of subsets of virtual machines with conflict conditions between the subsets. The goal is to pack all virtual machines using the minimum number of server racks within the considered time horizon. A heuristic based on the column generation method is proposed to solve this problem. We analyze a set of static problems at different time points necessary to form a common set of patterns used in the construction of upper bounds. The results of computational experiments on real open instances show a minor difference between the lower and upper bounds.

Keywords: bin packing problem; virtual machines; heuristics; placement groups; column generation

INFORMATION ABOUT THE AUTHORS

Brazhnikov, Evgeniy A.	ORCID iD: 0009-0004-3414-5176. E-mail: brazhnikov.eugene@gmail.com Engineer
Panin, Artem A. (corresponding author)	ORCID iD: 0000-0002-1844-6276. E-mail: aapanin1988@gmail.com Senior researcher, PhD
Ratushnyi, Alexey V.	ORCID iD: 0000-0001-5173-3536. E-mail: alexeyratushny@gmail.com Leading engineer

Funding: FWNF–2022–0019.

For citation: E. A. Brazhnikov, A. A. Panin, and A. V. Ratushnyi, “A patterning algorithm for the dynamic bin packing problem with placement groups”, *Modeling and Analysis of Information Systems*, vol. 32, no. 2, pp. 110–131, 2025.
DOI: 10.18255/1818-1015-2025-2-110-131.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 32, № 2, 2025
сайт журнала: www.mais-journal.ru

DISCRETE MATHEMATICS IN RELATION TO COMPUTER SCIENCE

Алгоритм шаблонизации для динамической задачи упаковки в контейнеры с группами размещения

Е. А. Бражников[1], А. А. Панин[1], А. В. Ратушный[1]

DOI: 10.18255/1818-1015-2025-2-110-131

[1]Институт математики им. С. Л. Соболева СО РАН, Новосибирск, Россия

УДК 519.8
Научная статья
Полный текст на русском языке

Получена 25 марта 2025 г.
После доработки 20 мая 2025 г.
Принята к публикации 21 мая 2025 г.

Рассматривается NP-трудная задача динамического распределения виртуальных машин по серверам с группами размещения. Для каждой виртуальной машины известны такие параметры, как необходимое количество ресурсов и временные метки создания и удаления. Каждый сервер представляет собой композицию NUMA-узлов и размещается в некоторой стойке. Рассматриваются большие виртуальные машины, размещаемые на два узла одного сервера, и маленькие, что накладывает дополнительные условия для их размещения. Группы размещения представляют собой объединения подмножеств виртуальных машин с условиями конфликта между подмножествами. Задача состоит в том, чтобы упаковать все виртуальные машины с использованием минимального количества стоек серверов в течение рассматриваемого временного горизонта. Для решения данной задачи предлагается эвристика, основанная на методе генерации столбцов. Анализируется набор статических задач в различные моменты времени, необходимых для формирования общего набора шаблонов, используемых при построении верхних оценок. Результаты вычислительных экспериментов на реальных открытых примерах указывают на незначительные расхождения между нижними и верхними границами.

Ключевые слова: задача упаковки в контейнеры; виртуальные машины; эвристики; группы размещения; генерация столбцов

ИНФОРМАЦИЯ ОБ АВТОРАХ

Бражников, Евгений Александрович	ORCID iD: 0009-0004-3414-5176. E-mail: brazhnikov.eugene@gmail.com Стажёр-исследователь
Панин, Артём Александрович (автор для корреспонденции)	ORCID iD: 0000-0002-1844-6276. E-mail: aapanin1988@gmail.com Старший научный сотрудник, канд. физ.-мат. наук
Ратушный, Алексей Владленович	ORCID iD: 0000-0001-5173-3536. E-mail: alexeyratushny@gmail.com Инженер-исследователь

Финансирование: FWNF–2022–0019.

Для цитирования: E. A. Brazhnikov, A. A. Panin, and A. V. Ratushnyi, “A patterning algorithm for the dynamic bin packing problem with placement groups”, *Modeling and Analysis of Information Systems*, vol. 32, no. 2, pp. 110–131, 2025. DOI: 10.18255/1818-1015-2025-2-110-131.

Введение

В последние десятилетия наблюдается существенный прогресс в развитии инфраструктуры облачных вычислений. Компании всё чаще обращаются к облачным сервисам для выполнения различных вычислительных задач, успешно применяя технологии виртуализации [1]. Последняя предоставляет удобство и гибкость взаимодействия с аппаратными ресурсами. С другой стороны, поставщики облачных ресурсов сталкиваются с задачей обслуживания множества виртуальных машин (ВМ). Быстрый рост числа ВМ влечет за собой рост числа используемых серверов и вызывает необходимость разработки эффективных методов распределения ВМ, т. е. методов решения динамической задачи упаковки в контейнеры (Temporal Bin Packing Problem, TBPP) [1—5]. В TBPP оптимизируется использование ресурсов, включая электроэнергию, что представляет собой критически важные аспекты с точки зрения эффективного функционирования облачных инфраструктур.

Сама по себе задача об упаковке в контейнеры является одной из самых исследованных задач в области исследования операций. Однако продолжают появляться её новые вариации, учитывающие динамику запросов на размещение ВМ и различные условия, накладывающие ограничения на распределение предметов по контейнерам. Например, авторы работы [6] фокусируются на онлайн-постановке задачи упаковки в контейнеры, которая часто возникает в компаниях, предоставляющих услуги облачных вычислений. Они проводят тщательный анализ производительности онлайн-алгоритмов как в худшем, так и среднем случаях. В исследовании [7] также рассматривается задача онлайн-планирования в распределённых вычислениях. Авторы предлагают гибридный подход, объединяющий машинное обучение и динамическое программирование, для удовлетворения ограничения рюкзака на общий вес. Для оценки прототипа использована симуляция задачи о 0–1 рюкзаке с неизвестными функциями полезности. Авторы работы [8] рассматривают задачи управления ресурсами для ВМ, используя подходы black-box и gray-box. Black-box подразумевает ограниченную информацию о внутреннем состоянии системы, тогда как gray-box предполагает более детальное знание. В работе представлен алгоритм Sandpiper, который адаптируется к различным уровням доступности информации о виртуальных машинах. В исследовании [9] рассматриваются методы снижения энергопотребления компьютерного кластера с помощью гибких стратегий управления состоянием узлов (включение и выключение) и порядком выполнения ожидающих задач.

Кроме того, продолжается разработка и исследование всё большего числа новых подходов. Так в работе [10] исследуются матэвристические методы для решения задачи TBPP. В ней рассматриваются как полиномиальная формулировка задачи, так и расширенная. Для них сравниваются различные эвристические алгоритмы, включая жадные методы и методы на основе генерации столбцов. Авторы [11] предлагают новый подход к планированию, чтобы минимизировать время выполнения задач. Они представляют алгоритмы, основанные на анализе характеристик задач, структуре кластера и требованиях пользователя. В исследовании [12] рассматриваются эвристические методы распределения ресурсов с учетом энергосбережения. Авторы стремились предложить подходы, которые учитывают особенности энергетической эффективности при принятии решений и способствуют балансу между производительностью вычислительных систем и ресурсосбережением. В работе [13] предлагается подход lookahead control, который предполагает анализ будущего состояния системы для принятия решений в текущий момент времени. Такой алгоритм позволяет предварительно реагировать на изменения в нагрузке, обеспечивая оптимальное распределение вычислительных ресурсов с учетом энергосбережения и уровня производительности. Авторы [14] исследуют задачу размещения виртуальных машин с учетом затрат на энергопотребление и миграцию. Они представляют алгоритм pMapper, который основывается на анализе энергетической эффективности и стоимости миграции. В работе [15] проводится анализ времени запуска виртуальных машин в облачных вычислениях. Авторы осуществляют эмпирическое исследование, оценивая

различные факторы, которые могут влиять на время запуска. Работа включает в себя анализ различных параметров и условий, таких как размеры ВМ, характеристики сети и оборудования, а также эффект использования различных технологий виртуализации.

Данная работа является расширенной версией работы [16] и продолжает развитие предложенных алгоритмов. Одним из основных отличий является замена постановки задачи с конфликтами между типами виртуальных машин на постановку с группами размещений, при этом задача остается NP-трудной как обобщение классической задачи упаковки в контейнеры. Для вычисления нижних границ, которые необходимы для оценки качества алгоритмов построения приближенных решений на реалистичных примерах большой размерности, мы провели адаптацию идеи генерации для учёта конфликтов. Для получения верхней оценки и приближенного решения мы предлагаем несколько алгоритмов, основанных на шаблонизации упаковки серверов. Нижние и верхние границы также могут быть использованы для оценки эффективности онлайн-алгоритмов. Основной вклад данного исследования состоит в разработке математической модели в форме целочисленного линейного программирования (ЦЛП) и анализе нескольких вариаций алгоритма шаблонизации упаковки. Мы также приводим результаты численных экспериментов, полученных на открытых данных.

Статья организована следующим образом. В разделе 1 вводятся необходимые обозначения и математическая модель рассматриваемой задачи. В разделе 2 мы приводим описание процедуры генерации столбцов и её модификации. Раздел 3 посвящён алгоритмам построения верхних оценок и решения задачи. В разделе 4 содержатся описание используемых данных, численных экспериментов и их результаты. В заключении мы кратко подводим итоги и обсуждаем возможные направления дальнейших исследований.

1. Математическая модель

Для описания математической модели введем множество однородных серверов S, достаточное для размещения всех ВМ. Сервера используют «архитектуру с неравномерным доступом к памяти» (NUMA) [17]. Каждый сервер $s \in S$ разделен на N узлов. На узле $n \in N$ доступно количество C_{nr} ресурса $r \in R$, где R — множество всех ресурсов. Мы считаем, что все серверы идентичны, однако узлы одного сервера могут отличаться между собой, т. е. допускается, что $C_{n_1 r} \neq C_{n_2 r}$ при $n_1 \neq n_2$.

Каждый из серверов располагается на некоторой стойке $f \in F$. На каждой стойке нельзя размещать больше D серверов. Так как мы стремимся минимизировать число стоек, а не число серверов, без ограничения общности будем считать, что каждая из них заполнена серверами полностью. Множество серверов стойки f обозначим S_f.

Требуется разместить множество виртуальных машин M, состоящее из двух подмножеств: M^{small} и M^{large}. Множество M^{small} содержит маленькие ВМ, каждая из которых размещается на одном узле сервера. M^{large} — множество больших ВМ, которые состоят из двух идентичных частей, размещаемых на различных узлах одного сервера. Каждая виртуальная машина $m \in M$ имеет некоторое требование b_{mr} ресурса $r \in R$ для каждого узла. Таким образом, маленькая (большая) машина m занимает b_{mr} $(2 \cdot b_{mr})$ ресурса r на некотором сервере во все моменты времени $t \in T$ такие, что $\alpha_m \leqslant t < \omega_m$, где α_m и ω_m являются, соответственно, временами создания и удаления m.

Кроме этого, некоторые подмножества виртуальных машин объединены в группы. Каждая группа $g \in G$ разбита на множество подгрупп P_g, называемых "партициями". Каждая пара подгрупп (p_i, p_j) из множества P_g группы g конфликтует друг с другом на уровне стоек. Т.е. никакие две виртуальные машины $m_1 \in p_i$ и $m_2 \in p_j$ не могут располагаться на одной стойке f, если они пересекаются по времени. Для удобства, мы будем обозначать множество машин из подгруппы p группы g как $M_{gp} \subset M$. Также определим множества виртуальных машин $M_t = M_t^{small} \cup M_t^{large}$, существующих в определенный момент времени t.

Для определения математической модели, введём следующие пременные:

- $x_{fsnm} \in \{0, 1\}$ равняется 1, если маленькая ВМ или первая часть большой ВМ $m \in M$ располагается на узле n сервера s стойки f, и 0 иначе;
- $y_{fsnm} \in \{0, 1\}$ равняется 1, если вторая часть большой ВМ $m \in M^{large}$ располагается на NUMA-узле n сервера s стойки f, и 0 иначе;
- $z_f \in \{0, 1\}$ равняется 1, если стойка f была активна, т. е. её ресурсы использовались хотя бы в один момент времени $t \in T$, и 0 иначе.

Используя эти обозначения, математическую модель можно представить в следующем виде:

$$\sum_{f \in F} z_f \to \min, \tag{1}$$

$$\sum_{n=1}^{N} x_{fsnm} = \sum_{n=1}^{N} y_{fsnm}, f \in F, s \in S_f, m \in M^{large}, \tag{2}$$

$$x_{fsnm} + y_{fsnm} \leqslant 1, f \in F, s \in S_f, n \in N, m \in M^{large}, \tag{3}$$

$$\sum_{f \in F} \sum_{s \in S_f} \sum_{n \in N} x_{fsnm} = 1, m \in M, \tag{4}$$

$$\sum_{m \in M_t} b_{mr} x_{fsnm} + \sum_{m \in M_t^{large}} b_{mr} y_{fsnm} \leqslant C_{nr}, f \in F, s \in S_f, n \in N, t \in T, \tag{5}$$

$$\sum_{s \in S_f} \sum_{n \in N} x_{fsnm} \leqslant z_f, f \in F, m \in M, \tag{6}$$

$$\sum_{s \in S_f} \sum_{n \in N} (x_{fsnm_1} + x_{fsnm_2}) \leqslant 1, \tag{7}$$

$$f \in F, g \in G, t \in T, p_1, p_2 \in P_g, m_1 \in M_{gp_1} \cap M_t, m_2 \in M_{gp_2} \cap M_t,$$
$$x_{fsnm}, y_{fsnm}, z_f \in \{0, 1\}, f \in F, s \in S, n \in N, m \in M. \tag{8}$$

Целевая функция (1) определяет размер пула ресурсов (число используемых стоек), который мы стремимся минимизировать. Ограничения (2) запрещают размещать большие виртуальные машины на NUMA-узлы разных серверов, а ограничения (3) запрещают класть их на один NUMA-узел. Согласно ограничению (4) все виртуальные машины должны быть размещены. Ресурсные ограничения отражены в (5). Ограничения (6) запрещают использовать стойки, которые не входят в пул ресурсов. Ограничения (7) запрещают в каждый момент времени класть виртуальные машины из разных подгрупп одной группы на одну и ту же стойку, но позволяют класть на одну стойку непересекающиеся по времени ВМ. В (8) задаются области определения переменных.

2. Генерация шаблонов и нижние оценки

Каждой ВМ m сопоставим вектор $(b_{m1}, \ldots, b_{mR})$, который мы назовём типом. Обозначим за L множество всех уникальных типов ВМ из M. Этот вектор представляет набор значений, описывающих требования данной ВМ. Каждая виртуальная машина m имеет некоторый интервал времени существования $[\alpha_m, \omega_m)$, который определяется пользователем. Число таких уникальных интервалов может совпадать с числом виртуальных машин. В то же время, число уникальных типов обычно очень ограничено, и считается, что $|L| \ll |M|$. Это связано с тем, что на практике чаще всего используются типовые конфигурации виртуальных машин, соответствующие стандартным сценариям использования. С примерами реальных типов виртуальных машин можно ознакомиться, например, в [18].

Для получения нижних оценок мы воспользовались модифицированной процедурой генерации столбцов, описанной в [4], для похожей задачи. Рассматривается некоторый момент времени $\hat{t}$.

Например, можно рассмотреть $\hat{t} = \arg\max_{t \in T} \sum_{m \in M_t} b_{mr}$ для некоторого r. В фиксированный момент времени виртуальные машины, имеющие одинаковый тип $l \in L$, неотличимы. Обозначим J множество всех возможных упаковок одного сервера. Элемент $j \in J$ можно представить в виде вектора $(a_{lj}, l \in L)$, где a_{lj} равняется целому числу виртуальных машин типа l. Пусть некоторое подмножество $J' \subset J$ является достаточным набором шаблонов для того, чтобы можно было упаковать все виртуальные машины данного примера.

Для описания процедуры получения нижней оценки сначала введём несколько дополнительных моделей. Первая модель называется координирующей задачей (master problem) линейного программирования и отвечает за назначение виртуальных машин на серверы по шаблонам. Переменная x_j равняется количеству серверов, которые используют шаблон j.

$$\min \sum_{j \in J'} x_j, \tag{9}$$

$$\sum_{j \in J'} a_{lj} x_j \geqslant n_l, l \in L, \tag{10}$$

$$x_j \geqslant 0, j \in J'. \tag{11}$$

Целевая функция (9) минимизирует число использованных серверов. Неравенства (10) гарантируют, что все виртуальные машины будут упакованы. Здесь $\hat{n}_l$ равняется числу виртуальных машин, имеющих тип l.

Пусть x_j^* — значения переменных x_j в оптимальном решении задачи (9)-(11), а λ_l^* — значения соответствующих двойственных переменных. Если неравенство $\sum_{l \in L} a_{lj} \lambda_l^* \leqslant 1$ выполняется для всех шаблонов $j \in J$, то решение x такое, что $x_j = x_j^*$ при $j \in J'$ и 0 иначе, является оптимальным решение задачи (9)–(11) для всего множества шаблонов J, а $LB = \lceil \sum_{j \in J'} x_j^* \rceil$ является нижней оценкой на оптимальное значение (1) при $D = 1$. Стоит отметить, что LB подсчитывается для числа серверов, однако её можно легко адаптировать для целевой функции (1), поделив на вместимость каждой стойки D.

Проверять неравенство выше для всех $j \in J$ достаточно сложно из-за экспоненциальной мощности множества J. Чтобы упростить задачу, можно рассмотреть задачу генерации шаблонов (pricing problem) для поиска следующего допустимого шаблона, которого ещё нет в J' и который мог бы уменьшить значение целевой функции (9).

Пусть переменные y_l^n определяют количество маленьких виртуальных машин (или первых частей больших машин) типа $l \in L$, расположенных на NUMA-узле $n \in N$, а z_l^n определяют количество вторых частей больших машин типа $l \in L^{large}$, расположенных на NUMA-узле $n \in N$. Здесь L^{small} и L^{large} являются наборами типов виртуальных машин, соответствующих множествам M^{small} и M^{large}. Тогда модель для задачи генерации шаблонов может быть записана следующим образом:

$$\max \alpha = \sum_{n \in N} \sum_{l \in L} \lambda_l^* y_l^n, \tag{12}$$

$$y_l^n \leqslant \sum_{\substack{k \in N \\ k \neq n}} z_l^k, \ l \in L^{large}, n \in N, \tag{13}$$

$$\sum_{n \in N} z_l^n = \sum_{n \in N} y_l^n, \ l \in L^{large}, \tag{14}$$

$$\sum_{l \in L} b_{lr} y_l^n + \sum_{l \in L^{large}} b_{lr} z_l^n \leqslant C_{nr}, \ r \in R, n \in N, \tag{15}$$

$$y_l^n \geqslant 0, integer, l \in L, n \in N, \tag{16}$$

$$z_l^n \geqslant 0, integer, l \in L^{small}, n \in N. \tag{17}$$

Целевая функция (12) отвечает за ценность упаковки. Равенства (13) и (14) гарантируют правильность расположения больших виртуальных машин. Неравенства (15) ограничивают доступность ресурсов на сервере.

Данный подход не учитывает наличие конфликтов между группами. Кажется, что их добавление способно положительно повлиять на нижнюю оценку. Будем называть тройку (l, g, p) расширенным типом виртуальных машин. Множество расширенных типов обозначим также L. Такая модификация позволяет нам достаточно удобно добавить необходимые ограничения в модель (12)–(17). Добавление всех возможных конфликтов не всегда целесообразно из-за существенного роста вычислительной сложности. Имеет смысл ограничить число групп, которые используются для подобного расширения числа типов. Например, можно рассмотреть группы с наибольшим числом виртуальных машин в определенный момент времени интервала T.

Пусть $\delta_{ij} = 1$, если два расширенных типа i и $j \in L$ не конфликтуют, и $\delta_{ij} = 0$ иначе. Для модификации модели (12)–(17) предлагается добавить следующие ограничения:

$$\chi_l \geqslant \sum_{n \in N} \frac{y_l^n}{M},\ l \in L, \tag{18}$$

$$\chi_i + \chi_j \leqslant 1 + \delta_{ij}, i, j \in L, i \neq j, \tag{19}$$

$$\chi_l \in \{0, 1\}, l \in L. \tag{20}$$

Пусть α^* — значение целевой функции в оптимальном решении задачи (12)–(17). Если $\alpha^* \leqslant 1$, то мы считаем, что смогли найти оптимальное решение задачи (9)–(11) и соответственно нижнюю оценку для задачи (1)–(8). Если $\alpha^* > 1$, то мы получили новый шаблон упаковки, которые можно включить во множество J'. Общая схема поиска нижней оценки представлена ниже.

Алгоритм вычисления нижней оценки:

Шаг 1: Пусть $\alpha^* = +\infty$.

Шаг 2: Решаем задачу, двойственную к (9)–(11) и сохраняем значения λ_l.

Шаг 3: Решаем задачу (12)–(20). Сохраняем значение целевой функции α^* и полученный шаблон.

Шаг 4: Если $\alpha^* \leqslant 1$, то мы нашли нижнюю оценку $LB = \lceil \sum_{i \in L} n_i \lambda_i^* \rceil$, иначе добавляем новый шаблон в J' и возвращаемся на шаг 2.

Заметим, что добавление конфликтов в задачу (12)–(17) действительно способно увеличить значение LB. Рассмотрим две ВМ, первая из которых занимает весь ресурс $r_1 \in R$ на сервере, а вторая весь ресурс $r_2 \in R$. В случае когда они принадлежат разным партициям одной группы, приведённая выше модификация генерации столбцов позволит нам получить нижнюю оценку равную двум серверам.

3. Верхние оценки

Для построения верхней оценки для задачи (1)–(8) мы предлагаем двухэтапный эвристический алгоритм, который основан на решении статической задачи для некоторого момента времени t. Алгоритм опирается на разумное предположение о том, что плотная упаковка виртуальных машин в момент максимальной нагрузки оказывает значительный эффект на остальные моменты времени и минимизирует общее количество используемых стоек. Шаблоны, построенные методом генерации столбцов, обеспечивают высокую плотность упаковки. Отсюда вытекает основная идея решения статической задачи — упаковка ВМ по шаблонам. Кроме этого, вычислительные эксперименты показывают, что выбор подходящего момента времени часто является непростой задачей. По этой причине будем рассматривать момент времени t как параметр алгоритма, придерживаясь предположения о том, что качество упаковки в данный момент времени ключевым образом влияет на верхнюю оценку.

3.1. Базовый алгоритм

Первый этап алгоритма заключается в плотной упаковке виртуальных машин из множества M_t в заданное число стоек. Так как мы рассматриваем только один момент времени, то, не ограничивая общности, можно отказаться от информации о временах создания и удаления и работать только с типами ВМ. В начале создаются пустые сервера, на которые назначаются шаблоны, полученные генерацией столбцов без конфликтов [4] для множества M_t. На первом этапе алгоритма упаковка происходит только в соответствии с шаблонами, а количество серверов каждого шаблона можно определить, решив задачу (9)–(11) с целыми переменными. Кроме серверов на первом этапе создаётся заданное количество стоек, которые изначально пусты. Их количество определяется нижней границей.

Следующим шагом ВМ назначаются на серверы с учётом конфликтов на уровне партиций, а сами серверы размещаются на стойки. Для конфликтующих виртуальных машин предварительно создаётся порядок обработки: группы, а также подгруппы внутри групп сортируются по убыванию суммарной нагрузки. Далее подгруппы разделяются на множества P_1 и P_2. Множество P_1 состоит из подгрупп, которые требуют для размещения более K серверов, где K – параметр алгоритма. Все остальные подгруппы составляют P_2. В первую очередь с помощью эвристического алгоритма размещаются подгруппы из множества P_1, а также назначаются серверы на стойки. Затем подгруппы из множества P_2 распределяются по свободным местам уже размещённых серверов. Виртуальные машины из P_2, которые не были распределены из-за заполненности серверов или ограничения конфликтов, располагаются на оставшихся неразмещённых серверах, после чего решается модель ЦЛП для размещения данных серверов на стойки с учётом конфликтов. Свободные виртуальные машины, которые не конфликтуют с другими, размещаются на самом последнем шаге статического этапа. Конкретные значения параметра K, которые использовались при тестировании, описаны в разделе 4. Алгоритм упаковки множества P_1 приведён ниже.

Процедура упаковки подгрупп из множества P_1:

Шаг 1: Выбираем ещё не просмотренные в порядке убывания нагрузки группу g и подгруппу p такие, что $p \in P_g \cap P_1$.

Шаг 2: Размещаем ВМ партиции p в активные серверы (на которых уже размещена хотя бы одна ВМ) согласно заданным на них шаблонам следующим образом:

1. Выделяем множество активных (с хотя бы одним размещенным сервером и хотя бы одной размещенной ВМ) стоек F_p, не содержащих ВМ из других подгрупп группы g.
2. Вычисляем вектор (k_l), где k_l — число неразмещённых ВМ типа l в текущей подгруппе p.
3. Для каждой необработанной стойки $f \in F_p$ вычисляем вектор (s_{fl}), где s_{fl} — число свободных мест для виртуальных машин типа $l \in L$.
4. Вычисляем значение $m_f = \sum_{l \in L} \min(s_{fl}, k_l)$, которое равняется числу неразмещённых ВМ подгруппы p, которые стойка f может вместить по шаблонам уже размещенных на ней серверов.
5. Сортируем необработанные стойки по убыванию m_f.
6. Выбираем следующую необработанную стойку f. Распределяем неразмещённые ВМ в свободные места активных серверов стойки и считаем её обработанной.

Шаг 3: Если все ВМ подгруппы p размещены, то возвращаемся на шаг 1. Иначе решаем модель (9)-(11) с целыми переменными для оставшихся ВМ из p и ограничением, что можно использовать только неразмещённые пустые серверы. Затем размещаем эти серверы с уже распределенными машинами из p на стойки следующим образом:

1. Сортируем стойки из F_p по убыванию числа свободных мест для серверов.

2. Пока не будут размещены все серверы или не будут обработаны все стойки, обходим стойки в данном порядке и на каждую размещаем столько серверов, сколько возможно. Возвращаемся на шаг 1.

Подгруппы из множества P_2 распределяются на частично заполненные стойки как на шаге 2. Оставшиеся после этого ВМ размещаются аналогично шагу 3, но теперь серверы размещаются на стойки посредством решения задачи ЦЛП. Для её описания введём необходимые обозначения:

- $\overline{S}$ — множество неразмещённых серверов;
- $\overline{G}$ — множество групп с подгруппами из множества P_2;
- $\overline{S}_{gp}$ — множество серверов, содержащих ВМ из подгруппы $p \in P_g$ группы $g \in \overline{G}$;
- $\overline{D}_f$ — количество свободных мест для серверов на стойке f.

Определим переменные:

- $x_{fs} \in \{0, 1\}$ равняется 1, если сервер s размещён на стойке f, и 0 иначе.

Получаем математическую модель:

$$\sum_{f \in F, s \in \overline{S}} x_{fs} \to \max, \tag{21}$$

$$\sum_{f \in F} x_{fs} \leqslant 1, s \in \overline{S}, \tag{22}$$

$$\sum_{s \in S} x_{fs} \leqslant \overline{D}_f, f \in F, \tag{23}$$

$$x_{fs_1} + x_{fs_2} \leqslant 1, f \in F, g \in G, p_1, p_2 \in P_g : p_1 < p_2, s_1 \in \overline{S}_{gp_1}, s_2 \in \overline{S}_{gp_2}, \tag{24}$$

$$x_{fs} \in \{0, 1\}, f \in F, s \in \overline{S}. \tag{25}$$

Целевая функция (21) определяет число серверов из $\overline{S}$, которые можно разместить, не создавая конфликтов. Ограничение (22) запрещает размещать сервер более чем на одной стойке. Неравенства (23) ограничивают вместимость стоек. Наконец, ограничение (24) определяет условие на конфликты между подгруппами.

Целесообразно выбирать параметр K так, чтобы большая часть подгрупп попала во множество P_1, то есть чтобы значение K было относительно невелико. Это связано с тем, что на практике, если большая часть подгрупп принадлежит множеству P_2, то ограничений типа (24) становится слишком много, а время решения задачи исчисляется сутками.

Стоит отметить, что статический этап алгоритма не гарантирует размещения всех ВМ из множества M_t. Но основные шаги статического этапа на этом заканчиваются. На втором этапе алгоритма происходит упаковка ВМ из множеств $M_t^{before} = \{m \in M | \omega_m \leqslant t\}$ и $M_t^{after} = \{m \in M | \alpha_m > t\}$, а после этого при необходимости создаются дополнительные стойки и упаковываются неразмещённые ВМ, в том числе те, которые не были размещены на первом этапе.

Рассмотрим шаги упаковки ВМ из M_t^{after}. Предварительно создаётся порядок обработки групп и виртуальных машин. Пусть G_t^{after} — множество групп, которые содержат ВМ из M_t^{after}. Группы сортируются в порядке убывания числа ВМ из M_{after}, которые они содержат. Помимо этого виртуальные машины в каждой группе отдельно сортируются по возрастанию α_t. Таким же образом создаётся порядок для свободных ВМ. Идея подобной сортировки в том, чтобы в первую очередь упаковать виртуальные машины, наиболее близкие по периоду жизни к моменту времени t.

Группы из G_t^{after} и виртуальные машины из M_t^{after}, обрабатываются в предложенном порядке жадной процедурой First Fit (первый подходящий), которая размещает ВМ по серверам стоек, не создавая конфликтов. При этом шаблоны серверов на данном шаге сохраняются и каждая ВМ размещается в свободную на время её существования ячейку сервера. Сохранение шаблонов необходимо для того, чтобы поддерживать плотный вариант упаковки и в другие моменты времени

помимо t. После упаковки ВМ из групп размещений производится упаковка свободных ВМ в определённом ранее порядке. Аналогичным образом обрабатывается множество M_t^{before}, но сортировка ВМ происходит по убыванию ω_t.

В силу того, что шаблоны были сгенерированы для одного момента t, часть виртуальных машин не будет упакована. Следующим шагом происходит отказ от шаблонной структуры серверов, а стойки дополняются пустыми серверами до предельного значения. Отказ от шаблонной структуры серверов может также дать возможность упаковать некоторые из неразмещённых ВМ множества M_t.

В соответствии с идеей алгоритма в первую очередь после отказа от шаблонов должна производится попытка упаковать неразмещённые ВМ из множества M_t. Также для уменьшения числа используемых серверов и стоек выгоднее сначала размещать конфликтующие ВМ. Аналогично множеству G_t^{after} определяется множество G_t и создаётся порядок обработки групп из этого множества. Виртуальные машины каждой обрабатываемой группы размещаются на стойки и серверы с помощью процедуры First Fit с учётом ограничения конфликтов. После этого аналогично упаковке в шаблоны производится упаковка неразмещённых конфликтующих ВМ из множеств M_t^{after} и M_t^{before}. Свободные виртуальные машины распределяются также с помощью процедуры First Fit в аналогичном порядке. При необходимости создаются новые стойки и сервера.

3.2. Модификации базового алгоритма

Помимо стандартного алгоритма были предложены и протестированы и другие, основанные на данном, но использующие альтернативные процедуры на некоторых шагах.

3.2.1. Альтернативная процедура упаковки для множества P_1

При размещении виртуальных машин из одной подгруппы на статическом этапе может оказаться важным задействовать как можно меньшее число стоек. Например, если алгоритм разместит 5 виртуальных машин одной подгруппы на 5 стойках, когда была возможность уместить их на одной, то места для других подгрупп может не остаться из-за конфликтов. Предлагается рассмотреть обратную стратегию, которая старается максимизировать плотность упаковки стоек, т. е. рассматривается все пространство стойки с учетом пустых мест под серверы. Однако даже такая процедура не лишена недостатков. Если некоторые стойки заполняются до предела, то алгоритм будет иметь меньшую ширину выбора для упаковки последующих групп. По этой причине были протестированы обе стратегии.

3.2.2. Альтернативные шаблоны упаковки для серверов

Одним из способов адаптировать шаблоны под задачу (1)–(8) является генерация столбцов с конфликтами, описанная в разделе 2. В проведённых экспериментах в качестве групп, участвующих в расширении типов, использовались 2 группы, содержащие наибольшее число ВМ. Уточнения требует только тот факт, что при переходе на второй этап алгоритмов места в шаблонах под конфликтующие типы (l, g, p) преобразуются в места под обычные типы l, то есть на втором этапе происходит отказ от шаблонов с конфликтами. Это необходимо, потому что конфликтующие шаблоны ограничивают конфликты только в момент времени t.

Ещё одной модификацией является использование универсальных шаблонов. Если прошлый подход адаптировал ограничение на конфликты, то данный частично адаптирует статические шаблоны под динамическую задачу. В некоторых случаях распределение типов виртуальных машин может значительно отличаться в различные моменты времени. Это приводит к ситуации, когда шаблоны в динамической задаче быстро теряют эффективность при последовательном отступе от ключевого момента времени. Для решения данной проблемы числа n_i из ограничения (10) можно вычислять в виде усреднённых по окрестности радиуса r_{cg} момента t.

Table 1. Analysis of instances **Таблица 1.** Анализ примеров

Семейство примеров	Число моментов времени	Число ВМ	Число групп с партициями	Среднее число ВМ в группе
DMP_l_2	379.68	38074.36	354.36	69.60
DMP_l_4	311.80	42577.24	425.72	63.27
DMP_m_2	343.88	32460.80	273.60	69.51
DMP_m_4	276.00	32767.40	271.36	62.99
DMP_s_2	274.48	22066.76	124.88	68.95
DMP_s_4	235.72	23220.84	121.88	62.06
LPO_l_2	338.28	64703.12	605.92	106.85
LPO_l_4	280.40	68349.76	671.84	101.80
LPO_m_2	153.72	28825.24	272.20	105.89
LPO_m_4	115.04	27328.32	271.76	100.59
LPO_s_2	70.28	13090.92	124.56	105.20
LPO_s_4	55.24	12781.92	124.92	102.20
MP_l_2	308.88	44820.88	641.40	69.96
MP_l_4	217.60	49286.20	737.04	66.91
MP_m_2	136.16	19213.04	273.84	70.08
MP_m_4	84.28	18679.64	278.64	66.98
MP_s_2	63.32	8838.44	126.16	69.98
MP_s_4	37.24	8213.20	121.92	67.46

3.2.3. Выбор момента времени

Момент времени с максимальной нагрузкой не всегда является удачным, и в некоторых случаях имеет смысл руководствоваться другими правилами. Допустим, у нас имеется некоторое решение, которое было получено выбором момента t в качестве начального. Возможна ситуация, когда в момент времени $t_1 \neq t$ задействуется большее число активных стоек r_{t_1}, чем в момент t. Предполагается, что применение алгоритма в другие моменты t, в т. ч. $r_t = \max_{\tau \in T} r_\tau$, может улучшить итоговый результат. Рассмотрим итеративную процедуру MultiStart, которая на каждой итерации строит верхнюю оценку, но в некоторый ещё не рассмотренный момент времени. Моментов, где задействовано наибольшее число стоек, может быть очень много, обработка всех занимала бы длительное время. Пусть t_{prev} — момент времени, использованный на предыдущей итерации. На каждой следующей итерации алгоритм применяется для ближайшего к t_{prev} момента времени с максимальным r_t. Отметим, что если максимальное значение достигается в момент времени t_{prev} и сохраняется в некото- рой окрестности, то выбирается ближайший момент времени, на котором достигается максимальное значение r_t вне указанной окрестности. Это позволяет избежать запуска новой итерации в момент, где распре- деление виртуальных машин остаётся практически неизменным. Процедура останавливается, если выбирается момент, использованный на предыдущих итерациях, или в случае, когда для данного решения максимум активных стоек единственный и достигается в момент t_{prev}. В качестве ответа выдаётся лучшее решение по всем итерациям.

4. Численные эксперименты

Алгоритмы реализованы в среде Python. Эксперимент проводился на компьютере с процессором AMD Ryzen 5 5600H и памятью 16 GB. Для проведения численных экспериментов были использованы открытые примеры [19], которые имитируют поведение реальной работы облачного сервиса. Различная информация о примерах приведена в таблице 1. Каждый пример принадлежит некоторому множеству, которое определяется 3 параметрами — семейством, размером и числом NUMA-узлов. Обозначения l, m и s отвечают за размер примера, т. е. за количество виртуальных машин

Table 2. Analysis of the parameter K **Таблица 2.** Анализ параметра K

Алгоритм	Best	$K = 0$	$K = 4$	$K = 9$	$K = 14$	$K = 19$	$K = 30$
Classic	6.98 %	7.50 %	7.98 %	8.26 %	8.32 %	8.33 %	8.34 %
Classic MultiStart	6.21 %	**6.70 %**	7.26 %	**7.48 %**	7.54 %	7.55 %	7.54 %
Dense	7.09 %	7.58 %	7.88 %	8.29 %	8.31 %	8.33 %	8.32 %
Dense MultiStart	**6.18 %**	6.71 %	**7.18 %**	7.49 %	7.54 %	7.54 %	7.53 %
Mean	7.02 %	7.55 %	8.00 %	8.29 %	8.30 %	8.30 %	8.29 %
Mean MultiStart	6.25 %	6.75 %	7.25 %	7.51 %	**7.49 %**	**7.50 %**	**7.50 %**
Dense Mean	7.13 %	7.61 %	7.90 %	8.31 %	8.27 %	8.29 %	8.28 %
Dense Mean MultiStart	6.21 %	6.74 %	**7.18 %**	7.49 %	7.50 %	**7.50 %**	7.51 %
CGC	8.23 %	8.58 %	9.52 %	9.60 %	9.69 %	9.72 %	9.74 %
CGC MultiStart	7.55 %	7.93 %	8.86 %	8.99 %	9.10 %	9.12 %	9.13 %
Dense CGC	8.63 %	8.99 %	9.66 %	9.62 %	9.74 %	9.73 %	9.76 %
Dense CGC MultiStart	7.95 %	8.35 %	9.05 %	8.98 %	9.15 %	9.14 %	9.16 %

и количество моментов времени. Так обозначения DMP, LPO, MP характеризуют семейства примеров, где LPO — примеры только с достаточно большими группами партиций, MP — примеры как с большими группами партиций, так и с маленькими, а DMP — расширение примеров MP с добавлением виртуальных машин, которые не принадлежат ни одной группе, т. е. не имеют никаких конфликтов. Буквы l, m и s обозначают размер примера, то есть количество виртуальных машин и моментов времени. Цифры 2 и 4 указывают, сколько NUMA-узлов доступно на серверах. Каждая группа состоит из 25 примеров, а суммарно бенчмарк включает 450 примеров. В таблице 1 в столбцах приводятся средние значения по всем примерам группы.

Далее в таблицах приводится анализ работы нескольких алгоритмов, которые включают в себя следующие:

1. Classic. Используются стандартные шаблоны и стандартная процедура упаковки ВМ множества P_1.
2. Dense Packing. Используются стандартные шаблоны, но применяется альтернативная процедура упаковки ВМ множества P_1.
3. Mean. Используются универсальные шаблоны и стандартная процедура упаковки ВМ множества P_1.
4. Dense Mean. Используются универсальные шаблоны и альтернативная процедура упаковки ВМ множества P_1.
5. CGC. Используются шаблоны с конфликтами и стандартная процедура упаковки ВМ множества P_1.
6. Dense CGC. Используются шаблоны с конфликтами и альтернативная процедура упаковки ВМ множества P_1.

В таблице 2 представлены результаты численных экспериментов, отражающие зависимость эффективности каждого алгоритма от параметра K. В клетках приводятся средние значения $GAP = (UB - LB)/UB$, вычисленные по всем примерам бенчмарка. В столбце Best приводится средний GAP, вычисленный с помощью выбора оптимального значения K для каждого конкретного примера. Было замечено, что результаты при $K > 30$ совпадают с результатами, полученными при $K = 30$, поэтому такие случаи не рассматриваются далее в анализе. В экспериментах с универсальными шаблонами использовался радиус $r_{cg} = 10$. Время работы решателя SCIP [20] версии 8.0.4, используемого для решения ЦЛП проблем, здесь и далее было ограничено 5-ю минутами.

В среднем, все рассматриваемые алгоритмы показывают лучшую производительность при параметре $K = 0$. Дополнительно, для каждого значения K было подсчитано число примеров, на которых результат совпадает с лучшим из всех результатов для этого же примера. Во всех случаях при $K = 0$

Table 3. Algorithms without CGC and MultiStart

Таблица 3. Алгоритмы без процедур CGC и MultiStart

Группа примеров	Classic		Dense		Mean		Dense Mean	
	Best	$K = 0$	Best	$K = 0$	Best	$K = 0$	Best	$K = 0$
LPO_s_2	6.93 %	7.90 %	6.83 %	7.98 %	7.15 %	8.04 %	7.61 %	8.57 %
LPO_s_4	11.59 %	12.55 %	10.51 %	10.67 %	11.59 %	12.55 %	10.51 %	10.67 %
LPO_m_2	6.01 %	7.05 %	6.39 %	7.15 %	6.01 %	7.05 %	6.39 %	7.15 %
LPO_m_4	6.30 %	6.55 %	6.26 %	6.69 %	6.39 %	6.64 %	6.26 %	6.69 %
LPO_l_2	12.55 %	12.93 %	12.58 %	12.93 %	12.55 %	12.93 %	12.58 %	12.93 %
LPO_l_4	8.51 %	8.69 %	8.44 %	8.73 %	8.51 %	8.69 %	8.44 %	8.73 %
MP_s_2	7.66 %	8.33 %	7.11 %	8.06 %	7.49 %	8.33 %	7.11 %	8.06 %
MP_s_4	10.35 %	10.52 %	11.64 %	11.85 %	11.07 %	11.24 %	11.64 %	11.85 %
MP_m_2	7.17 %	8.00 %	7.51 %	8.36 %	7.17 %	8.00 %	7.51 %	8.36 %
MP_m_4	5.67 %	6.40 %	6.67 %	7.00 %	5.57 %	6.30 %	6.67 %	7.00 %
MP_l_2	10.29 %	10.78 %	10.41 %	10.82 %	10.29 %	10.78 %	10.41 %	10.82 %
MP_l_4	5.42 %	5.73 %	5.49 %	5.80 %	5.38 %	5.69 %	5.49 %	5.80 %
DMP_s_2	4.46 %	4.93 %	4.53 %	5.48 %	4.46 %	4.93 %	4.53 %	5.48 %
DMP_s_4	2.78 %	3.15 %	2.61 %	2.89 %	2.78 %	3.23 %	2.61 %	2.89 %
DMP_m_2	7.13 %	7.67 %	7.40 %	7.84 %	7.13 %	7.67 %	7.40 %	7.84 %
DMP_m_4	1.98 %	2.26 %	2.20 %	2.42 %	1.98 %	2.26 %	2.20 %	2.42 %
DMP_l_2	8.65 %	8.97 %	8.78 %	9.31 %	8.65 %	8.97 %	8.78 %	9.31 %
DMP_l_4	2.22 %	2.56 %	2.22 %	2.48 %	2.22 %	2.56 %	2.22 %	2.48 %
GAP_{Mean}	6.98 %	7.50 %	7.09 %	7.58 %	7.02 %	7.55 %	7.13 %	7.61 %

достигнуто наибольшее значение. Например, для Classic MultiStart при $K = 0$ это число составляет 326, а минимально оно при $K = 19$ и равняется 228. С увеличением значения параметра K практически монотонно происходит ухудшение средних результатов алгоритмов. Это означает, что алгоритмы имеют достаточно высокую эффективность, когда подгруппы не разделяются на большие и маленькие по количеству занимаемых серверов. Значение параметра $K = 0$ может быть предпочтительным, когда мы не обладаем достаточным временем для перебора различных случаев.

Рассмотрим пример статической задачи, который может объяснить ухудшение. Пусть имеется 2 группы, в каждой по 2 подгруппы. Подгруппа 1.1 для размещения требует 3/4 ресурса сервера, а подгруппа 1.2 требует 1/4. Подгруппа 2.1 требует 2/4 ресурса, а подгруппа 2.2 требует 1/4. Имеется 2 стойки f_1 и f_2, на каждой к моменту упаковки данных групп осталось по 1 ячейке для пустого сервера. При $K = 0$ для упаковки групп данных стоек хватит, согласно базовому алгоритму для 1.1 будет выделен пустой сервер и подгруппа будет упакована на f_1, аналогично 1.2 на f_2, а 2.1 и 2.2 займут свободное место на f_2 и f_1, соответственно. Имеет место такая последовательность упаковки, потому что группы сортируются по нагрузке, после чего внутри самих групп сортируются подгруппы. Теперь пусть $K = 1/2$, тогда подгруппа 1.1 будет добавлена в P_1 и упакована на первом этапе в f_1, а подгруппы 1.2, 2.1 и 2.2, напротив, будут добавлены в P_2. На втором этапе в первую очередь происходит упаковка в размещённые на стойках сервера. На f_1 для 1.1 размещён сервер, а на f_2 осталась только пустая ячейка для сервера. Группы и подгруппы внутри снова сортируются по нагрузке, поэтому сначала будет паковаться группа 2 и подгруппа 2.1, которая займёт только оставшуюся часть сервера на f_1 в размере $\frac{1}{4}$, но не будет упакована полностью. В итоге к следующему шагу алгоритма не будут полностью упакованы 2.1 и 2.2. Для их упаковки потребуется дополнительная стойка, так как f_1 полностью заполнена, а на одной f_2 подгруппы размещаться не могут из-за ограничений.

Результаты сравнения алгоритмов детализированы в трех таблицах: 3, 4 и 5, где также приведено разбиение на группы примеров. В каждой таблице столбцы содержат усреднённые значения

Table 4. Algorithms with MultiStart **Таблица 4.** Алгоритмы с процедурой MultiStart

Группа примеров	Classic MultiStart		Dense MultiStart		Mean MultiStart		Dense Mean MultiStart	
	Best	$K = 0$	Best	$K = 0$	Best	$K = 0$	Best	$K = 0$
LPO_s_2	6.36 %	7.21 %	6.09 %	7.20 %	6.41 %	7.81 %	6.64 %	7.79 %
LPO_s_4	10.25 %	11.29 %	9.32 %	9.34 %	10.42 %	11.13 %	9.32 %	9.34 %
LPO_m_2	5.75 %	6.46 %	5.95 %	6.53 %	5.75 %	6.46 %	5.95 %	6.53 %
LPO_m_4	5.49 %	5.74 %	5.21 %	5.55 %	5.40 %	5.74 %	5.21 %	5.55 %
LPO_l_2	11.05 %	11.71 %	11.14 %	11.78 %	11.05 %	11.71 %	11.14 %	11.78 %
LPO_l_4	7.03 %	7.18 %	6.88 %	7.33 %	7.03 %	7.18 %	6.88 %	7.33 %
MP_s_2	6.52 %	7.56 %	6.38 %	6.73 %	6.48 %	7.56 %	6.38 %	6.73 %
MP_s_4	10.19 %	10.15 %	10.08 %	10.43 %	10.70 %	10.51 %	10.08 %	10.43 %
MP_m_2	5.85 %	6.57 %	6.10 %	7.32 %	5.85 %	6.57 %	6.10 %	7.32 %
MP_m_4	4.87 %	5.25 %	5.54 %	5.64 %	4.99 %	5.37 %	5.54 %	5.64 %
MP_l_2	8.92 %	9.58 %	9.01 %	9.52 %	8.92 %	9.58 %	9.01 %	9.52 %
MP_l_4	4.63 %	4.94 %	4.71 %	5.10 %	4.59 %	4.90 %	4.71 %	5.10 %
DMP_s_2	4.09 %	4.45 %	4.24 %	5.32 %	4.09 %	4.45 %	4.24 %	5.32 %
DMP_s_4	2.62 %	2.85 %	2.45 %	2.68 %	2.62 %	2.93 %	2.45 %	2.68 %
DMP_m_2	6.63 %	7.17 %	6.62 %	7.28 %	6.63 %	7.17 %	6.62 %	7.28 %
DMP_m_4	1.76 %	2.04 %	1.88 %	2.21 %	1.76 %	2.04 %	1.88 %	2.21 %
DMP_l_2	7.83 %	8.19 %	7.79 %	8.49 %	7.83 %	8.19 %	7.79 %	8.49 %
DMP_l_4	1.97 %	2.27 %	1.88 %	2.31 %	1.97 %	2.27 %	1.88 %	2.31 %
GAP_{Mean}	6.21 %	6.70 %	6.18 %	6.71 %	6.25 %	6.75 %	6.21 %	6.74 %

GAP по всем примерам из соответствующей группы. Кроме того, в последней строке каждой таблицы представлены значения *GAP*, усреднённые по всем примерам, что позволяет оценить общую эффективность алгоритмов. Таблица 3 содержит результаты алгоритмов, которые не применяют генерацию столбцов с учётом конфликтов, в то время как таблица 4 отражает результаты тех же алгоритмов с применением MultiStart. Данная процедура позволяет улучшить результат в среднем на 0.78 %.

В результатах всех алгоритмов наблюдается схожая зависимость от групп примеров. Влияние числа NUMA-узлов на результаты может быть обусловлено устройством генератора и требует более глубокого исследования. При фиксированном K наилучший результат $GAP_{Mean} = 6.70\,\%$ показал алгоритм Classic MultiStart, но при условии настройки параметра для каждого примера отдельно выиграл алгоритм Dense MultiStart с $GAP_{Mean} = 6.18\,\%$. Согласно данным таблицы 5, алгоритмы, использующие конфликтующие шаблоны, показали относительно плохие результаты. Отчасти такие результаты связаны с тем, что в генерации шаблонов участвовали всего 2 группы, тогда как в самых маленьких примерах число групп с конфликтами достигает 100. Тем не менее ранее был проведён эксперимент на примерах сильно меньшей размерности, где существовало в среднем 12 моментов времени, 20 групп с подгруппами и 6000 ВМ. Такой размер эксперимента позволил задействовать все группы в генерации шаблонов, однако средний результат алгоритмов с такими шаблонами также значительно проигрывал аналогам. Подобный результат может свидетельствовать о непригодности шаблонов с конфликтами для данного типа алгоритмов.

В таблице 6 указаны максимальное, минимальное и среднее время работы каждого алгоритма в секундах на всём множестве примеров. Алгоритмы, использующие генерацию столбцов с конфликтами, уступили аналогам без неё как в плане *GAP*, так и по времени работы. Дольше всех в среднем решаются примеры из семейства LPO, так как при больших группах партиций значительно увеличивается число конфликтов. Примеры из семейства DMP, напротив, решаются быстрее

Table 5. Algorihms based on the column generation with conflicts

Таблица 5. Алгоритмы, основанные на генерации столбцов с конфликтами

Группа примеров	CGC		CGC MultiStart		Dense CGC		Dense CGC MultiStart	
	Best	$K = 0$	Best	$K = 0$	Best	$K = 0$	Best	$K = 0$
LPO_s_2	10.29 %	11.10 %	9.18 %	9.79 %	9.99 %	11.03 %	9.11 %	10.26 %
LPO_s_4	12.83 %	13.32 %	12.19 %	12.85 %	13.44 %	13.84 %	12.75 %	12.84 %
LPO_m_2	6.93 %	7.61 %	6.30 %	6.99 %	7.45 %	7.91 %	6.50 %	7.19 %
LPO_m_4	7.12 %	7.13 %	6.67 %	6.50 %	7.66 %	7.90 %	7.15 %	7.48 %
LPO_l_2	12.79 %	13.23 %	11.38 %	11.83 %	12.76 %	13.17 %	11.26 %	11.63 %
LPO_l_4	8.58 %	8.88 %	7.45 %	7.74 %	8.73 %	8.87 %	7.37 %	7.70 %
MP_s_2	9.81 %	10.53 %	8.88 %	9.59 %	10.42 %	11.63 %	9.38 %	10.22 %
MP_s_4	14.40 %	15.19 %	14.08 %	14.86 %	14.91 %	15.19 %	14.91 %	15.19 %
MP_m_2	8.05 %	8.66 %	6.81 %	7.56 %	8.20 %	8.69 %	6.90 %	7.72 %
MP_m_4	8.08 %	8.19 %	7.91 %	8.11 %	9.62 %	9.45 %	9.45 %	9.36 %
MP_l_2	10.59 %	10.94 %	9.01 %	9.52 %	10.47 %	10.72 %	8.98 %	9.51 %
MP_l_4	5.65 %	5.85 %	4.91 %	5.06 %	5.65 %	5.84 %	5.03 %	5.22 %
DMP_s_2	4.80 %	5.17 %	4.50 %	4.93 %	5.09 %	5.32 %	4.79 %	5.13 %
DMP_s_4	4.33 %	4.22 %	4.33 %	4.22 %	5.39 %	5.53 %	5.39 %	5.53 %
DMP_m_2	7.51 %	7.67 %	7.05 %	7.29 %	7.56 %	7.86 %	7.25 %	7.41 %
DMP_m_4	3.59 %	3.49 %	3.59 %	3.49 %	4.65 %	4.91 %	4.65 %	4.91 %
DMP_l_2	8.94 %	9.38 %	7.86 %	8.48 %	8.96 %	9.35 %	7.89 %	8.38 %
DMP_l_4	3.88 %	3.91 %	3.83 %	3.87 %	4.43 %	4.67 %	4.35 %	4.59 %
GAP_{Mean}	8.23 %	8.58 %	7.55 %	7.93 %	8.63 %	8.99 %	7.95 %	8.35 %

Table 6. Running time

Таблица 6. Время работы алгоритмов

Алгоритм	Максимальное время	Минимальное время	Среднее время
Classic	346	**3**	105
Classic Iter	3047	**3**	330
Dense	**344**	**3**	**107**
Dense Iter	3019	**3**	330
Mean	558	**3**	118
Mean Iter	4727	**3**	367
Dense Mean	373	**3**	111
Dense Mean Iter	3252	**3**	349
CGC	3478	10	969
CGC iter	13476	13	1919
Dense CGC	6716	9	996
Dense CGC iter	15141	9	1941

всех из-за большого числа неконфликтующих ВМ. Примеры с 2 NUMA-узлами решаются в среднем по времени приблизительно одинаково для всех алгоритмов. Однако на примерах с 4 NUMA-узлами алгоритмы с CGC могут работать в среднем в 27 раз медленнее, чем на примерах с 2 NUMA-узлами, тогда как остальные алгоритмы замедляются лишь в 1.2–2 раза. Замедление происходит из-за роста числа переменных и ограничений в модели (12)–(17). Из-за наличия расширенных типов в модели (12)–(20) число переменных и ограничений растёт с увеличением количества NUMA-узлов в несколько раз быстрее.

Заключение

В настоящем исследовании была подробно рассмотрена динамическая задача упаковки виртуальных машин в серверные стойки с учетом групповых размещений и конфликтов между ними. Мы разработали математическую модель для данной NP-трудной задачи. Для её решения была предложена эвристическая процедура, основанная на методе генерации столбцов и распространении статического решения. Мы также провели анализ нескольких эвристик и сравнили их по времени выполнения и качеству работы на обширном бенчмарке. Кроме того, мы изучили влияние параметров на эффективность работы разработанных алгоритмов. Предложенные методы и модели могут быть обобщены на другие варианты задачи упаковки с группами размещения.

В дальнейших работах планируется более тщательное исследование предложенных алгоритмов. Например, планируется протестировать разделение групп целиком на большие и маленькие, вместо разделения подгрупп. Также интересно было бы адаптировать алгоритмы для шаблонов с конфликтами, т. к. пока что они показывают относительно плохой результат.

References

[1] Z. Farahmandpour, M. Seyedmahmoudian, and A. Stojcevski, “A review on the service virtualisation and its structural pillars”, *Applied Sciences*, vol. 11, no. 5, p. 2381, 2021.

[2] M. de Cauwer, D. Mehta, and B. O’Sullivan, “The temporal bin packing problem: An application to workload management in data centres”, in *IEEE 28th International Conference on Tools with Artificial Intelligence*, 2016, pp. 157–164.

[3] M. Dell’Amico, F. Furini, and M. Iori, “A branch-and-price algorithm for the temporal bin packing problem”, *Computers & Operations Research*, vol. 114, p. 104 825, 2019.

[4] A. Ratushnyi and Y. Kochetov, “A column generation based heuristic for a temporal bin packing problem”, in *Mathematical Optimization Theory and Operations Research*, 2021, pp. 96–110.

[5] M. Sakhno, “A grouping genetic algorithm for the temporal vector bin packing problem”, in *19th International Asian School-Seminar on Optimization Problems of Complex Systems*, 2023, pp. 94–99.

[6] D. Lazarev and N. Kuzyurin, “On online algorithms for bin, strip, and box packing, and their worst-case and average-case analysis”, *Programming and Computer Software*, vol. 45, no. 4, pp. 448–457, 2020, in Russian.

[7] V. Toporkov, D. Yemelyanov, and A. Bulkhak, “Machine learning-based online scheduling in distributed computing”, in *Parallel Processing and Applied Mathematics*, 2023, pp. 248–259.

[8] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, “Sandpiper: Black-box and gray-box resource management for virtual machines”, *Computer Networks*, vol. 53, no. 17, pp. 2923–2938, 2009.

[9] V. Ivannikov, D. Grushin, N. Kuzyurin, A. Pospelov, and S. A., “Software for improving the energy efficiency of a computer cluster”, *Programming and Computer Software*, vol. 36, pp. 327–336, 2010.

[10] F. Furini and X. Shen, “Matheuristics for the temporal bin packing problem”, in *Recent Developments in Metaheuristics*. 2018, pp. 333–345.

[11] D. Grushin and N. Kuzyurin, “On effective scheduling in computing clusters”, *Programming and Computer Software*, vol. 45, pp. 398–404, 2019.

[12] A. Beloglazov, J. Abawajy, and R. Buyya, “Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing”, *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[13] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, “Power and performance management of virtualized computing environments via lookahead control”, *Cluster Computing*, vol. 12, pp. 1–15, 2009.

[14] A. Verma, P. Ahuja, and A. Neogi, “pMapper: Power and migration cost aware application placement in virtualized systems”, in *Middleware*, 2008, pp. 243–264.

[15] M. Mao and M. Humphrey, “A performance study on the VM startup time in the cloud”, in *IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 423–430.

[16] A. Ratushnyi, “A pattern-based heuristic for a temporal bin packing problem with conflicts”, in *Mathematical Optimization Theory and Operations Research: Recent Trends*, 2023, pp. 161–175.

[17] C. Lameter, “NUMA (non-uniform memory access): An overview”, *Queue*, vol. 11, no. 7, pp. 40–51, 2013.

[18] *Placement groups for your Amazon EC2 instances*. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html.

[19] *Temporal bin packing problem with placement groups*. [Online]. Available: http://old.math.nsc.ru/AP/benchmarks/Temporal%20Bin%20Packing/binpack.html.

[20] *SCIP. Solving constraint integer programs*. [Online]. Available: https://www.scipopt.org/.

Приложение. Псевдокод используемых алгоритмов

Sort. Sorting a list of objects — **Sort.** Сортировка списка объектов

Input: Список объектов *List*, список ключей *Keys*, флаг порядка сортировки *IsDescendingOrder*;
Output: Отсортированный список объектов;
if *isDescendingOrder = false* **then**
  Сортируем *List* по возрастанию соответствующих ключей из списка *Keys*;
end
else
  Сортируем *List* по убыванию соответствующих ключей из списка *Keys*;
end
return *List*

FirstFit. First Fit procedure — **FirstFit.** Процедура «Первый подходящий»

Input: Упорядоченный список предметов *Items*, упорядоченный список контейнеров *Containers*, флаг возможности добавлять новые стойки *ExpansionAllowed*;
for ***every*** *item* ∈ *Items* **do**
  IsUnplaced ← *item* не упакован;
  for ***every*** *container* ∈ *Containers* **do**
    ItemFits ← *item* помещается в *container*;
    NoPlacementConflict ← не нарушается ограничение на группы размещений;
    if *IsUnplaced = true* ***and*** *ItemFits = true* ***and*** *NoPlacementConflict = true* **then**
      Пакуем *item* в *container*; `// если container - стойка, а item - ВМ, то пакуем в первый подходящий сервер`
      IsUnplaced ← *false*;
    end
  end
  if *IsUnplaced = true* ***and*** *ExpansionAllowed = true* **then**
    f ← новая стойка, заполненная пустыми серверами;
    F.append(*f*); `// добавляем в множество стоек`
    Пакуем *item* в *f*;
  end
end

SelectServers. Procedure for selecting suitable servers

Input: Список виртуальных машин VM, список серверов $Servers$
Output: Список выделенных серверов
Добавляем в (9)-(11) ограничение на целочисленность переменных;
Добавляем в (9)-(11) ограничение, что можно использовать только сервера из $Servers$;
Определяем матрицу (a_{lj}) в соответствии с шаблонами серверов из $Servers$;
Определяем числа n_l в соответствии с типами ВМ из списка VM;
$SelectedServers \leftarrow$ получаем подходящие неразмещённые серверы из решения модели;
return *SelectedServers*

SelectServers. Процедура выделения подходящих серверов

Algorithm 1. Static stage of the basic algorithm

Input: множество виртуальных машин M_t, которые существуют в момент $t \in T$, параметр $K \geq 0$
$Servers, LB \leftarrow$ применяем метод генерации столбцов без конфликтов для множества M_t;
`// получили список серверов с шаблонами и нижнюю границу на число серверов`
$LB \leftarrow \lceil LB/RackCapacity \rceil$; `// делим на вместимость одной стойки по серверам и получаем нижнюю границу на число стоек`
Создаём LB пустых стоек без серверов;
for ***every*** $g \in G$ **do**
  for ***every*** $p \in P_g$ **do**
    $w_{gp} \leftarrow$ суммарная нагрузка всех ВМ $m \in M_t$, которые принадлежат подгруппе p;
  end
  $w_g \leftarrow \sum_{p \in P_g} w_{gp}$;
end
$G^{sorted} \leftarrow Sort(G, (w_g)_{g \in G}, true)$; `// группы с нулевой суммарной нагрузкой в момент` t `не учитываются`
for ***every*** $g \in G$ **do**
  $P_g^{sorted} \leftarrow Sort(P_g, (w_{gp})_{g \in G, p \in P_g}, true)$;
end
$P_1, P_2 \leftarrow SplitSubgroups(G, (P_g)_{g \in G}, K)$; `// Разделяем подгруппы в зависмости от` K`: если для упаковки подгруппы требуется > K серверов, то подгруппа добавляется в` P_1`, иначе в` P_2
$Algorithm1(G^{sorted}, (P_g^{sorted})_{g \in G}, P_1)$; `// упаковка подгрупп из` P_1
Пакуем подгруппы из P_2 с помощью решения задачи (21)-(25);
$Unconstrained \leftarrow$ свободные ВМ;
$FirstFit(Unconstrained, F, false)$; `//` F `- множество стоек`

Algorithm 1. Статический этап базового алгоритма

SortGroups. Sorting groups by number of VMs

Input: множество групп $\widetilde{G}$, множество виртуальных машин $\widetilde{M}$
for ***every*** $g \in \widetilde{G}$ **do**
  $h_g \leftarrow$ кол-во ВМ из $\widetilde{M}$ в этой группе g;
end
$Sort(\widetilde{G}, (h_g)_{g \in \widetilde{G}}, true)$; `// сортировка по убыванию числа ВМ`

SortGroups. Сортировка групп по числу ВМ

Algorithm 2. The procedure for packing partitions from the set P_1

Algorithm 2. Процедура упаковки подгрупп из множества P_1

Input: Отсортированный по убыванию суммарной нагрузки ВМ список групп G^{sorted}, отсоритрованные по убыванию суммарной нагрузки ВМ списки подгрупп P_g^{sorted}, множество подгрупп P_1;

for ***every*** $g \in G^{sorted}$ **do**
  for ***every*** $p \in P_g^{sorted}$ **do**
    if $p \in P_1$ **then**
      Выделяем множество активных стоек F_p; `// на которых размещена хотя бы одна ВМ`
      for ***every*** $p' \in P_g^{sorted}$ **do**
        if $p' \neq p$ **then**
          for ***every*** $f \in F_p$ **do**
            if *f содержит ВМ из p'* **then**
              Удаляем f из F_p; `// для соблюдения ограничения на группы размещений`
            end
          end
        end
      end
      for ***every*** $l \in L$ **do**
        $k_l \leftarrow$ число ВМ типа l в подгруппе p;
      end
      for ***every*** $f \in F_p$ **do**
        for ***every*** $l \in L$ **do**
          $s_{fl} \leftarrow$ число незанятых мест для ВМ типа l на стойке f;
        end
        $m_f \leftarrow \sum_{l \in L} min(s_{fl}, k_l)$;
      end
      $F_p^{sorted} \leftarrow Sort(F_p, (m_f)_{f \in F_p}, true)$; `// По убыванию`
      for ***every*** $f \in F_p^{sorted}$ **do**
        $UnplacedItems \leftarrow$ неразмещённые ВМ из p;
        $FreeSlots \leftarrow$ незанятые места серверов стойки f;
        $FirstFit(UnplacedItems, FreeSlots, false)$;
      end
      if *в подгруппе p остались неразмещённые ВМ* **then**
        for ***every*** $f \in F_p$ **do**
          $c_f \leftarrow$ число незанятых для серверов мест стойки f;
        end
        $F_p^{sorted} \leftarrow Sort(F_p, (c_f)_{f \in F_p}, true)$; `// По убыванию`
        $Servers \leftarrow$ неразмещённые серверы;
        $SelectedServers \leftarrow SelectServers(p, Servers)$;
        Распределяем оставшиеся ВМ из p на серверы из $SelectedServers$; `// Все ВМ будут распределены`
        for ***every*** $f \in F_p^{sorted}$ **do**
          Размещаем на стойку f столько серверов из $SelectedServers$, сколько возможно; `// ВМ из неразмещённых серверов будут доупакованы на динамическом этапе`
        end
      end
    end
  end
end

PackGroups. Packing a list of groups

Input: упорядоченный список групп $\widetilde{G}$, упорядоченный список виртуальных машин $\widetilde{M}$, флаг упаковки свободных виртуальных машин $PackUnconstrained$, флаг возможности добавлять новые стойки $ExpansionAllowed$;

for ***every*** $g \in \widetilde{G}$ **do**
  for ***every*** $m \in \widetilde{M}$ **do**
    if *m содержится в g* **then**
      $FirstFit((m), F, ExpansionAllowed)$; `// сначала учитывается порядок групп, потом порядок ВМ; если ВМ уже была упакована, это учитывается внутри процедуры` $FirstFit$
    end
  end
end
if $PackUnconstrained = true$ **then**
  for ***every*** $m \in \widetilde{M}$ **do**
    if *m свободная ВМ* **then**
      $FirstFit((m), F, ExpansionAllowed)$;
    end
  end
end

Algorithm 3. Dynamic stage of the basic algorithm

PackGroups. Упаковка списка групп

Algorithm 3. Динамический этап базового алгоритма

Input: разделённые по моменту $t \in T$ множества виртуальных машин M_t^{before}, M_t, M_t^{after};

$G_t \leftarrow$ группы, в которых есть ВМ из M_t;

$G_t^{before} \leftarrow$ группы, в которых есть ВМ из M_t^{before};

$G_t^{after} \leftarrow$ группы, в которых есть ВМ из M_t^{after};

$SortGroups(G_t, M_t)$;

$SortGroups(G_t^{before}, M_t^{before})$;

$SortGroups(G_t^{after}, M_t^{after})$;

$Sort(M_t^{after}, (\alpha_m)_{m \in M_t^{after}}, false)$;

$Sort(M_t^{before}, (\omega_m)_{m \in M_t^{before}}, true)$;

$PackGroups(G_t^{after}, M_t^{after}, false, false)$; `// сохранение шаблонов`

$PackGroups(G_t^{before}, M_t^{before}, false, false)$; `// сохранение шаблонов`

Возвращаемся к изначальной структуре серверов без шаблонов;

$PackGroups(G_t, M_t, false, true)$; `// отказ от шаблонов`

$PackGroups(G_t^{after}, M_t^{after}, false, true)$; `// отказ от шаблонов`

$PackGroups(G_t^{before}, M_t^{before}, false, true)$; `// отказ от шаблонов`

$PackGroups(G_t, M_t, true, true)$; `// упаковка свободных ВМ`

$PackGroups(G_t^{after}, M_t^{after}, true, true)$; `// упаковка свободных ВМ`

$PackGroups(G_t^{before}, M_t^{before}, true, true)$; `// упаковка свободных ВМ`

Algorithm 4. Alternative packing procedure for P_1

Algorithm 4. Альтернативная процедура упаковки для P_1

Input: Отсортированный по убыванию суммарной нагрузки ВМ список групп G^{sorted}, отсоритрованные по убыванию суммарной нагрузки ВМ списки подгрупп P_g^{sorted}, множество подгрупп P_1;

for ***every*** $g \in G^{sorted}$ **do**
  for ***every*** $p \in P_g^{sorted}$ **do**
    if $p \in P_1$ **then**
      Выделяем множество активных стоек F_p; `// на которых размещена хотя бы одна ВМ`
      for ***every*** $p' \in P_g^{sorted}$ **do**
        if $p' \neq p$ **then**
          for ***every*** $f \in F_p$ **do**
            if *f содержит ВМ из p'* **then**
              Удаляем f из F_p; `// для соблюдения ограничения на группы размещений`
            end
          end
        end
      end
      for ***every*** $l \in L$ **do**
        $k_l \leftarrow$ число ВМ типа l в подгруппе p;
      end
      for ***every*** $f \in F_p$ **do**
        for ***every*** $l \in L$ **do**
          $s_{fl} \leftarrow$ число незанятых мест для ВМ типа l на стойке f;
        end
        $m_f \leftarrow \sum_{l \in L} min(s_{fl}, k_l)$;
      end
      $F_p^{sorted} \leftarrow Sort(F_p, (m_f)_{f \in F_p}, true)$; `// По убыванию`
      for ***every*** $f \in F_p^{sorted}$ **do**
        $UnplacedItems \leftarrow$ неразмещённые ВМ из p;
        $FreeSlots \leftarrow$ незанятые места серверов стойки f;
        $FirstFit(UnplacedItems, FreeSlots, false)$;
        if *в подгруппе p остались неразмещённые ВМ* **then**
          $Servers \leftarrow$ неразмещённые серверы;
          $SelectedServers \leftarrow SelectServers(p, Servers)$;
          Распределяем оставшиеся ВМ из p на серверы из $SelectedServers$; `// Все ВМ будут распределены`
          Размещаем на стойку f столько серверов из $SelectedServers$, сколько возможно; `// ВМ из неразмещённых серверов будут записаны в` $UnplacedItems$ `на следующей итерации`
        end
      end
    end
  end
end

CalculateActiveRacks. Calculating the number of active racks

CalculateActiveRacks. Вычисление числа активных стоек

Input: данные об упаковке всех виртуальных машин *Solution*;
Output: Количество активных стоек в каждый момент времени (список) $(r_t)_{t \in T}$;
for ***every*** $t \in T$ **do**
    $r_t \leftarrow 0$;
    for ***every*** $f \in F$ **do**
        if *В Solution есть ВМ, которая существует в момент t и упакована на стойке f* **then**
            $r_t \leftarrow r_t + 1$;
        end
    end
end
return $(r_t)_{t \in T}$

Algorithm 5. MultiStart procedure

Algorithm 5. Процедура MultiStart

Input: множество виртуальных машин M, параметр $K \geq 0$, вариация алгоритма получения верхней оценки *Algorithm*;
Output: верхняя оценка на число стоек, допустимое решение задачи
$\hat{t} \leftarrow \arg\max_{t \in T}(\max_{r \in R} \sum_{m \in M_t} b_{mr})$; `// вычисляем момент максимальной нагрузки`
$prevMoments \leftarrow \emptyset$; `// список предыдущих моментов`
$minValue \leftarrow -\infty$;
$bestSolution \leftarrow \emptyset$;
while $\hat{t} \notin prevMoments$ **do**
    $value,\ solution \leftarrow Algorithm(M, \hat{t}, K)$;
    if $value > minValue$ **then**
        $minValue \leftarrow value$;
        $bestSolution \leftarrow solution$;
    end
    $(r_t)_{t \in T} \leftarrow CalculateActiveRacks(solution)$;
    if $r_{\hat{t}}$ *- единственный максимум в* $(r_t)_{t \in T}$ *или принадлежит единственному плато максимумов* **then**
        break; `// сработал один из критериев остановки`
    end
    $prevMoments.Append(\hat{t})$;
    $\hat{t} \leftarrow$ ближайший к $\hat{t}$ момент времени, в котором достигается максимум по числу активных стоек из $(r_t)_{t \in T}$;
end
return $minValue, bestSolution$

Algorithm 6. Basic algorithm for getting the upper bound

Algorithm 6. Базовый алгоритм получения верхней оценки

Input: множество виртуальных машин M, момент времени $t \in T$, параметр $K \geq 0$
Output: верхняя оценка на число стоек, допустимое решение задачи
$M_t \leftarrow \{m \in M | \alpha_m \leq t < \omega_m\}$;
$M_t^{before} \leftarrow \{m \in M | \omega_m \leq t\}$;
$M_t^{after} \leftarrow \{m \in M | \alpha_m > t\}$;
$Algorithm2(M_t, K)$; `// статический этап`
$Algorithm3(M_t^{before}, M_t, M_t^{after})$; `// динамический этап`
$Placements \leftarrow$ информация об упаковке всех ВМ из M;
return $|F|, Placements$