# 1. Условия задачи

Дан набор вычислительных задач и набор виртуальных машин. Для каждой задачи известен объём вычислений и требования к памяти, а для каждой виртуальной машины — производительность и ограничения по памяти. Для задач заданы отношения предшествования (вида «задача C должна выполняться после задач A и B»).

Необходимо назначить каждую задачу одной виртуальной машине так, чтобы:
- все ограничения по ресурсам были выполнены;
- все отношения предшествования были соблюдены;
- общее время выполнения всех задач (максимальное время завершения) было минимальным.

# 2. Формализация задачи

## 2.1. Структура решения

Пусть $n$ — количество задач, $m$ — количество виртуальных машин.

Для каждой задачи $i = 1, ..., n$ заданы:
- $T_i$ — объём вычислений (в условных единицах),
- $V_i$ — требуемый объём памяти,
- $P_i \subset \{1, ..., n\}$ — множество непосредственных предшественников (не содержит циклов).

Для каждой виртуальной машины $j = 1, ..., m$ заданы:
- $F_j$ — производительность (единиц вычислений в единицу времени),
- $M_j$ — доступный объём памяти.

Решение представляет собой назначение каждой задачи одной машине и определение порядка выполнения задач на каждой машине (должен быть совместим с отношениями предшествования).

### 2.1.1. Представление задачи на графах

Задача планирования вычислительных задач с учётом предшествований и ограничений ресурсов может быть формализована с использованием ориентированного ациклического графа $G = (V, E)$, где $V = \{v_1, v_2, ..., v_n\}$ — множество вершин, каждая из которых представляет задачу $i = 1, ..., n$, аннотированную атрибутами $T_i$ (объём вычислений) и $V_i$ (требуемый объём памяти). Множество рёбер $E \subset V \times V$ определяет отношения предшествования: ребро $(v_k, v_i) \in E$ существует, если задача $k$ является непосредственным предшественником задачи $i$ (т.е., $k \in P_i$), обеспечивая отсутствие циклов в графе.

Для интеграции виртуальных машин в модель вводится дополнительное множество вершин $U = \{u_1, u_2, ..., u_m\}$, где каждая вершина $u_j$ соответствует виртуальной машине $j = 1, ..., m$ с атрибутами $F_j$ (производительность) и $M_j$ (доступная память). Назначение задач машинам эквивалентно добавлению рёбер между вершинами $V$ и $U$, образуя двудольный граф $H = (V \cup U, E_H)$, где $E_H$ — множество рёбер назначения вида $(v_i, u_j)$, указывающее, что задача $i$ выполняется на машине $j$. Каждое такое ребро должно удовлетворять ограничению по памяти: $V_i \leq M_j$. Порядок выполнения задач на одной машине определяется топологическим порядком в подграфе, индуцированном задачами, назначенными на эту машине, с учётом исходных предшествований в $G$.

Время выполнения задачи $i$ на машине $j$ вычисляется как $t_i^j = \frac{T_i}{F_j}$, а момент начала $S_i$ — как максимум из моментов завершения предшественников и предыдущих задач на той же машине:

$$S_i = \max\left\{\max_{k \in P_i} C_k, \max\{C_l \mid l \text{ предшествует } i \text{ на машине } j\}\right\}$$

где

- $C_i = S_i + t_i^j$.

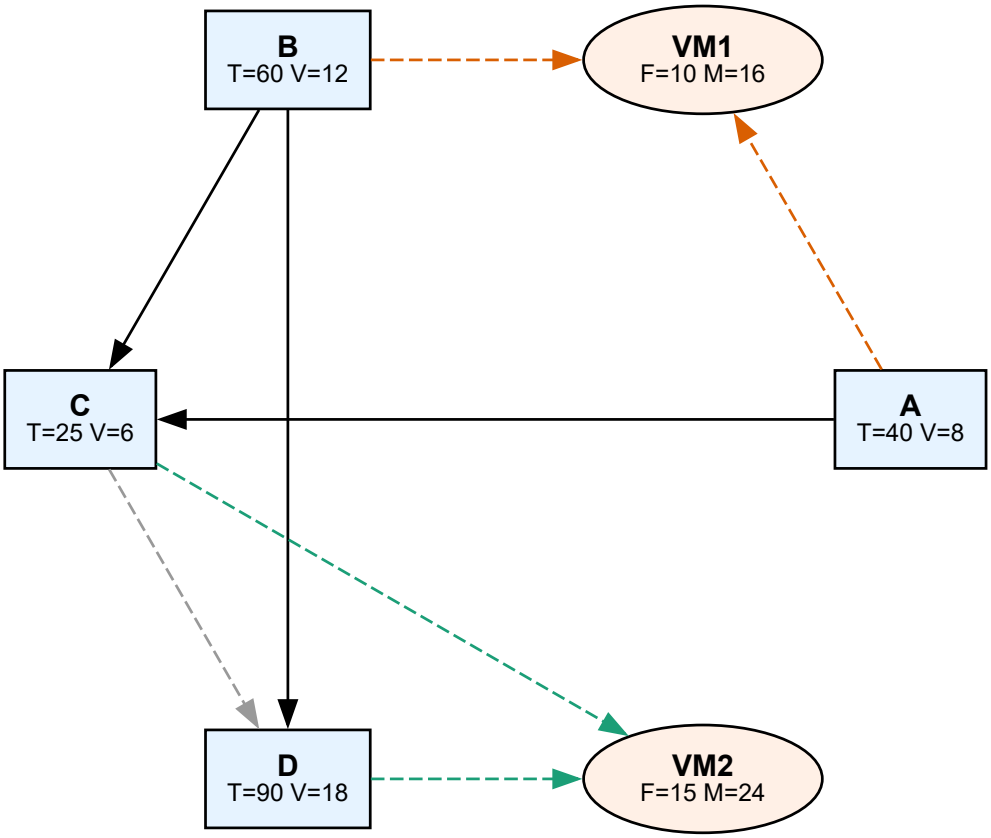


Рис. 1. Пример графа

### 2.1.2. Способы кодирования решения

1. Вектор назначения

   $$X = (x_1, x_2, ..., x_n), \quad x_i \in \{1, 2, ..., m\}$$

   где $x_i = j$ означает, что задача $i$ назначена на машину $j$.

2. Булева матрица назначения

   $$M_{ij} \in \{0, 1\}, \quad \sum_{j=1}^{m} M_{ij} = 1 \quad \forall i \in 1..n$$

## 2.2. Целевая функция

Цель — минимизация общего времени завершения всех задач:

$$f(X) = \max_{i=1..n} C_i$$

где
- $C_i$ — момент завершения задачи $i$,
- $C_i = S_i + t_i^{x_i}$,
- $t_i^j = \frac{T_i}{F_j}$ — время выполнения задачи $i$ на машине $j$,
- $S_i$ — момент начала выполнения задачи $i$.

Значение $S_i$ определяется как

$$S_i = \max\left\{ \max_{k \in P_i} C_k, \max\{C_k : \text{задача } k \text{ выполняется на той же машине перед } i\} \right\}$$

## 2.3. Способ учёта ограничений

Ограничения задачи делятся на два типа:
- жёсткие (должны выполняться в любом допустимом решении),
- мягкие (учитываются через штрафные функции).

Жёсткие ограничения:

1. Каждой задаче назначается ровно одна виртуальная машина:

$$x_i \in \{1, ..., m\} \quad \forall i = 1..n$$

2. Ограничение по памяти для каждой задачи:

$$V_i \le M_{x_i} \quad \forall i$$

3. Соблюдение порядка предшествования:

$$S_i \ge \max_{k \in P_i} C_k \quad \forall i \quad \text{с непустым } P_i$$

Варианты штрафных функций для мягкого учёта ограничений:

- Штраф за нарушение памяти по каждой задаче:

$$p_i^{\text{память}} = \max\left(0, V_i - M_{x_i}\right) \cdot K_{\text{память}}$$

- Штраф за нарушение предшествования:

$$p_i^{\text{предш}} = \max\left(0, \max_{k \in P_i} C_k - S_i\right) \cdot K_{\text{предш}}$$

- Общий штраф решения:

$$P(X) = \sum_{i=1}^{n} (p_i^{\text{память}} + p_i^{\text{предш}})$$

Итоговая функция, которую минимизирует алгоритм:

$$\tilde{f}(X) = \max_{i=1..n} C_i + P(X)$$

# 3. Выбор биоинспирированного алгоритма

## 3.1. Сравнение возможных алгоритмов

Для решения задачи распределения вычислительных задач с учётом предшествования и ограничений ресурсов рассмотрены биоинспирированные алгоритмы, подходящие для дискретной оптимизации. Проведён анализ генетического алгоритма, муравьиного алгоритма, алгоритма роя частиц и алгоритма искусственной пчелиной колонии.

Генетический алгоритм эволюционирует популяцию решений, применяя операции селекции, скрещивания и мутации. Он эффективен для комбинаторных задач, но требует тщательной настройки параметров и значительных вычислительных ресурсов для сходимости.

Муравьиный алгоритм моделирует поведение муравьёв, используя феромонные следы для усиления предпочтительных назначений в графе. Алгоритм хорошо обрабатывает графовые зависимости, однако его реализация сложна из-за управления матрицей феромонов, а сходимость может быть медленной в гетерогенных системах.

Алгоритм роя частиц имитирует движение стаи, где каждая частица обновляет позицию на основе личного и коллективного опыта. Дискретная модификация алгоритма роя частиц адаптируется к векторному представлению решений, демонстрируя высокую скорость сходимости и качество решений в задачах планирования.

Алгоритм искусственной пчелиной колонии, основанный на поведении пчёл, разделяет агентов для баланса между исследованием и использованием пространства поиска. Алгоритм конкурентоспособен, но требует больше вычислительных затрат на итерацию по сравнению с алгоритмом роя частиц.

Сравнительный анализ показал, что алгоритм роя частиц часто превосходит рассмотренные алгоритмы по скорости сходимости и минимизации общего времени выполнения (на 10–20% в типичных сценариях), сохраняя относительную простоту реализации.

## 3.2. Обоснование выбора алгоритма роя частиц

Выбор алгоритма роя частиц для решения поставленной задачи обусловлен следующими факторами:

1. Высокая скорость сходимости, что критично для NP-трудных задач с большим количеством задач.
2. Относительная простота адаптации дискретной версии алгоритма к векторному представлению решения $X$ и интеграции штрафной функции $P(X)$ для учёта ограничений.
3. Меньшее количество управляемых параметров по сравнению с генетическим алгоритмом и муравьиным алгоритмом, что упрощает настройку.
4. Лучшая масштабируемость на гетерогенные вычислительные ресурсы.

Таким образом, алгоритм роя частиц обеспечивает оптимальный баланс между качеством решения, скоростью работы и сложностью реализации в контексте задачи планирования с графом предшествования.

## 3.3. Описание алгоритма роя частиц

Алгоритм работает с роем частиц. Каждая частица представляет возможное решение — вектор назначений $X$. Частица характеризуется позицией (текущее решение) и скоростью (вектор изменений).

Инициализация: генерируются начальные позиции частиц случайным образом с последующей корректировкой для соблюдения жёстких ограничений по памяти.

На каждой итерации $t$ скорость и позиция частицы обновляются:

$$v_{\text{id}}^{t+1} = w \cdot v_{\text{id}}^{t} + c_1 \cdot r_1 \cdot (p_{\text{id}} - x_{\text{id}}^{t}) + c_2 \cdot r_2 \cdot (g_d - x_{\text{id}}^{t})$$

$$x_{\text{id}}^{t+1} = x_{\text{id}}^{t} + v_{\text{id}}^{t+1}$$

где:
- $w$ — коэффициент инерции,
- $c_1, c_2$ — коэффициенты ускорения,
- $r_1, r_2$ — случайные величины из интервала $[0, 1]$,
- $p_{\text{id}}$ — лучшее найденное положение частицы (личный опыт),
- $g_d$ — лучшее положение во всём рое (глобальный опыт).

В дискретной версии алгоритма скорость $v_{\text{id}}$ интерпретируется как вероятность изменения назначения, после чего применяется процедура дискретизации для получения целочисленного значения $x_{\text{id}}$.

Целевая функция для оценки частицы — $\tilde{f}(X) = \max_{i=1..n} C_i + P(X)$. Личные и глобальные лучшие позиции обновляются в соответствии с её значением.

Алгоритм завершает работу по достижении заданного числа итераций или при отсутствии улучшений глобального решения.

## 3.4. Адаптация алгоритма к задаче

Позиция частицы кодируется вектором назначения $X$. Для соблюдения ограничений по памяти используется процедура репарации решений. При вычислении целевой функции $\tilde{f}(X)$ применяется метод спискового расписания: для каждой виртуальной машины задачи упорядочиваются по моменту готовности, определяемому завершением задач-предшественников. Это позволяет вычислить моменты начала $S_i$ и завершения $C_i$ для каждой задачи, учитывая зависимости в графе. Штрафная функция $P(X)$ учитывает возможные нарушения ограничений, что позволяет алгоритму эффективно минимизировать общее время выполнения.


Рис. 2. Схема алгоритма роя частиц