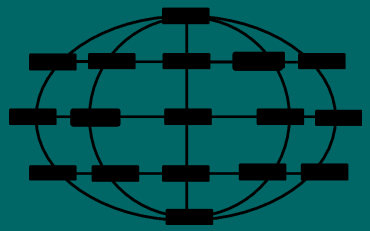
Методы построения и анализа алгоритмов

ЛЕКЦИЯ 3

Эвристические комбинаторные алгоритмы



Малышкин Виктор Эммануилович

Кафедра Параллельных Вычислительных Технологий

Новосибирский государственный технический университет

E_mail: **malysh@ssd.sscc.ru**

Телефон: **3308 994**

Новосибирск

Краткая классификация алгоритмов

ЗАДАЧИ	СВОЙСТВА
Численные Системные Комбинаторные Оптимизационные Прикладные (машиностроение, космос, наука, экономика, …) Теоретические Технологические И прочие и прочие	Параллельные Высокоточные Интенсивные вычисления Интенсивные данные Быстрые Динамические Настраивающиеся Самоорганизация Минимальное потребление ресурсов Надежные И прочие и прочие

Эвристические комбинаторные алгоритмы

- Эвристические алгоритмы дают решение хуже оптимального.
- Они обычно имеют полиномиальную сложность и быстро строят решение. Дополнительно в них всегда есть возможность для человека поучаствовать в построении хорошего расписания.
- Расписание получается обычно не оптимальное, а субоптимальное.

Общая постановка задачи построения расписания

Существует множество различных постановок задачи построения расписаний. Здесь рассмотрен простой её вариант (bin packing) в приложении к планированию заданий с учетом требуемых ресурсов.

- Пусть заданы m боксов (bin) $B_1, B_2, \ldots, B_m$, каждый объёмом c.
- Пусть также даны n единиц хранения $p_1, p_2, \ldots, p_n$, $n >> m$, объем каждого p_i, $1 \le i \le n$, обозначим $r(p_i)$. Предполагается, что $\forall i \; r(p_i) \le c$.
- Все p_i надо так разместить в B_i, чтобы некоторый функционал Φ достиг экстремума (равномерность, минимум не занятого объёма, и т.д.)

Пусть, к примеру, B_i трактуются как процессорные элементы мультикомпьютера, c – объём имеющихся в каждом B_i ресурсов, p_i – процессы, которые должны быть исполнены на мультикомпьютере и запрашивают для своего исполнения некоторое количество ресурсов $r(p_i)$.

Конкретные назначениия p_i на B_i задают отображение M (расписание). Допускаются отображения, в которых суммарный запрос ресурса в каждом B_i не превышает c.

Тогда имеем постановки задач построения расписания распределения ресурсов мультикомпьютера

Можно сформулировать 3 проблемы:

P1. Для фиксированного c найти минимальное число боксов m, в которые могут быть упакованы все p_i.

Это, например, случай планирования загрузки мультикомпьютера с целью так минимизировать число используемых для решения задачи процессоров, чтобы успеть её решить к заданному моменту времени T.

Здесь функционал качества расписания Φ - это минимум ресурсов, необходимых для решения задачи к моменту T.

P2. Для фиксированного m найти минимальное c такое, что все p_i могут быть упакованы в m боксов.

Это случай, когда надо найти минимальную производительность процессорных элементов мультикомпьютера с m процессорами, достаточную для решения задачи в заданное время.

Здесь производительность процессорных элементов мультикомпьютера играет роль функционала качества Φ

РЗ. Для фиксированных m и c найти максимальное $n' \leq n$ такое, что n' единиц хранения могут быть упакованы в m боксов.

Это случай, когда имеющийся мультикомпьютер должен быть максимально использован (задача *пакетной обработки*).

Все три проблемы имеют экспоненциальную сложность и для их решения должны использоваться переборные алгоритмы нахождения приближенного решения.

Общая схема решения таких *переборных* задач состоит в следующем:

. построении множества всех допустимых (удовлетворяющих ограничениям задачи) расписаний,

.вычислении на каждом построенном расписании s функционала $\Phi(s)$ и

.нахождении такого расписания s_0, на котором функционал $\Phi(s_0)$ достигает экстремального значения.

Вообще говоря, может быть найдено не одно такое расписание.

Рассмотрим сначала алгоритм построения множества всех допустимых расписаний для случая $m=n$. Каждое расписание представляется n-кой (набором) вида $(p_1,p_2,\ldots,p_n)$, где p_1 в первой позиции набора означает назначение процесса p_1 на исполнение на процессор B_1 и так далее.

Пусть на качество допустимых расписаний не накладываются никакие ограничения,

На каждый процессор назначается только один процесс. В число допустимых расписаний попадает, например, $(p_1,p_1,\ldots,p_1)$.

Тогда все множество допустимых расписаний строится тем же алгоритмом прибавления единицы, каким строятся все n-ричные числа от нуля до n^n-1. Число всех допустимых расписаний в данном случае равно n^n.

В программировании обычно $n>>m$. Поэтому надо разрешать назначение на один процессор нескольких процессов и тогда множество допустимых расписаний увеличится, так как в каждой позиции набора появится ещё возможность перебора и станут допустимыми расписания типа ($\{p_1,p_2\}$, $\{p_3,p_4\}$, ..., $\{p_{n-1},p_n\}$). Множества допустимых расписаний, конечно, очень велики и на практике работать с ними нельзя.

Ограничения задачи могут значительно сократить множество допустимых расписаний. Для параллельных программ возможно ограничение, чтобы каждый процесс был назначен не более чем на один процессор, при этом расписания типа ($p_1,p_1,\ldots,p_1$), ($\{p_1,p_2\}$, $\{p_1,p_3\}$, ..., $\{p_{n-1},p_n\}$) становятся не допустимыми. Ограничение естественное, если не надо обеспечивать высокую надежность вычислений. В противном случае оно недопустимо

Другое ограничение связано с учетом информационных зависимостей. Если она существует, например, между процессами p_1 и p_2, и процессоры B_1 и B_2 соседние (связаны физическим линком), а процессоры B_1 и B_3 не являются соседними, то расписания типа ($\{p_1,p_2\}$, $\{p_3\},\ldots,\{p_n\}$) и ($\{p_1\}$, $\{p2,p3\},\ldots,\{p_n\}$) допустимы (информационно зависимые процессы назначены на один или соседние процессоры), а расписание ($\{p_1\}$, $\{p3\}$, $\{p2\},\ldots,\{p_n\}$) – не допустимо. Такие сокращения множества допустимых расписаний очень полезны для уменьшения затрат на получение решения, так как отбраковка расписания происходит ещё на этапе его конструирования, часть расписаний не строится вообще, и значение функционала не вычисляется.

К сожалению, добавление новых ограничений при попытке сократить множество допустимых расписаний нередко быстро приводит к отсутствию решения (не существует расписания, удовлетворяющего всем ограничениям). В таком случае пытаются ослабить какие-то ограничения. Например, разрешить транзитные коммуникации в 2-окрестности процессора, затем в 3-окрестности и так до тех пора, пока не найдется приемлемое решение. Понятно, что множество допустимых расписаний в этом последнем случае расширяется.

Поэтому на практике задача решается эвристическими алгоритмами, которые строят приемлемое расписание в соответствии с некоторой стратегией.

Такие стратегии используют некоторое дополнительное знание о задаче. Рассмотрим следующий пример. Пусть заданы m идентичных процессоров B_1, B_2, ..., B_m. Каждый процессор B_i, $1 \le i \le m$, имеет оперативную память размером $|B_i|$. Заданы информационно независимые процессы p_1, p_2, ..., p_n, здесь $p_j:(m_j, t_j)$, каждый процесс p_j, $1 \le j \le n$, требует для своего исполнения память объемом m_j, время исполнения t_j. Необходимо построить (субоптимальное) расписание с приемлемым временем исполнения. Несколько процессов могут быть назначены в процессор B_i, если их суммарный запрос памяти не превышает $|B_i|$.

К примеру, расписание

$(\{p_1, p_2, p_3\}, \{p_5, p_9\}, \{p_{n-5}\}, \ldots, \{p_n\})$

показывает, что назначенные на процессор B_1 процессы $\{p_1, p_2, p_3\}$ будут и исполняться в порядке перечисления, а именно,: сначала исполнится процесс p_1, затем, по его завершении, стартует процесс p_2, и затем только выполнится p_3.

Стратегия конструирования расписания должна учесть свойства задачи.

Для сформулированной задачи ясно, что нехорошо оставлять назначение процессов, требующих для своего исполнения большого объема памяти, на конец – они могут не поместиться в памяти никакого процессора из-за того, что часть памяти процессоров уже занята ранее назначенными процессами.

Так же нехорошо откладывать в конец назначение долго исполняющихся процессов – может получиться так, что все прочие процессы закончили исполнение, а последний процесс ещё долго будет работать, задерживая завершение всего множества процессов. В обоих случаях могут построиться неудачные расписания.

Потому разумно для построения расписания использовать такую стратегию:

- Строятся два упорядоченных списка процессов:
  - L1 - список процессов, упорядоченных по убыванию объёма запрашиваемой памяти
  - L2 - список процессов, упорядоченных по убыванию времени исполнения,
- Процессы из первой трети списков L1 и L2 (они содержат процессы с самыми большими запросами ресурсов) назначаются на процессоры, более или менее равномерно заполняя память и потребляя время процессоров.
- Затем аналогично назначаются процессы из второй, а затем и последней трети списков L1 и L2.

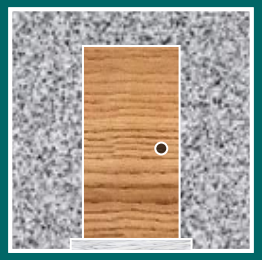
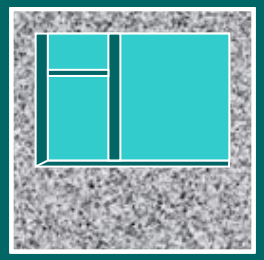
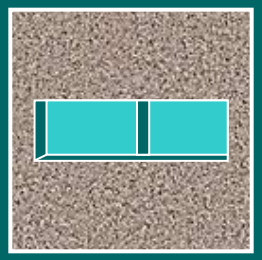
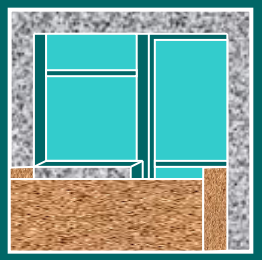
Построенное расписание на практике может быть и очень хорошим и не очень, но вряд ли оно будет совсем уж плохим.

Определить понятие приемлемое (хорошее) расписание не просто, однако нередко можно построить оценку стратегии и доказать, насколько построенное решение будет отличаться от оптимального, например, например, будет хуже оптимального не более чем на 30%. Решения в пределах этих 30% и могут считаться приемлемыми.

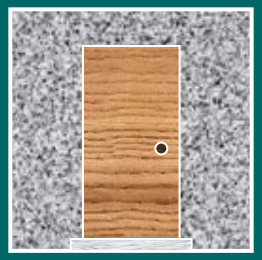
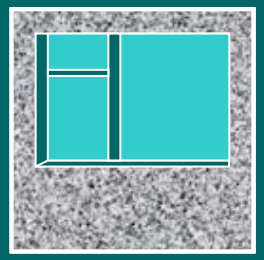
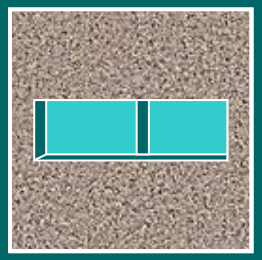
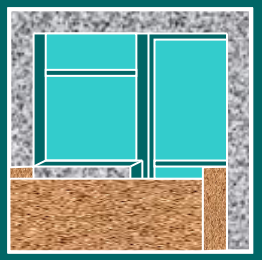
Это практически полезная стратегия, которая позволяет строить приемлемые расписания. Её часто применяют и в жизни. Если надо оптимально упаковать чемодан, то практичные люди сначала укладывают в чемодан самые крупные и самые тяжёлые вещи, а затем все остальное, и получается неплохо. Однако, если размеры и вес укладываемых вещей не соответствуют размерам и прочности чемодана, то никакая стратегия назначения не приведет к хорошему результату. А вот если загружать чемодан песком (каждая песчинка намного меньше чемодана и все они одного размера и веса), то при любой стратегии укладки песчинок чемодан будет загружен оптимально. Так что построение хорошего расписания зависит не только (а скорее не столько) от методов построения расписания, сколько от свойств задачи (процессов, процессоров).

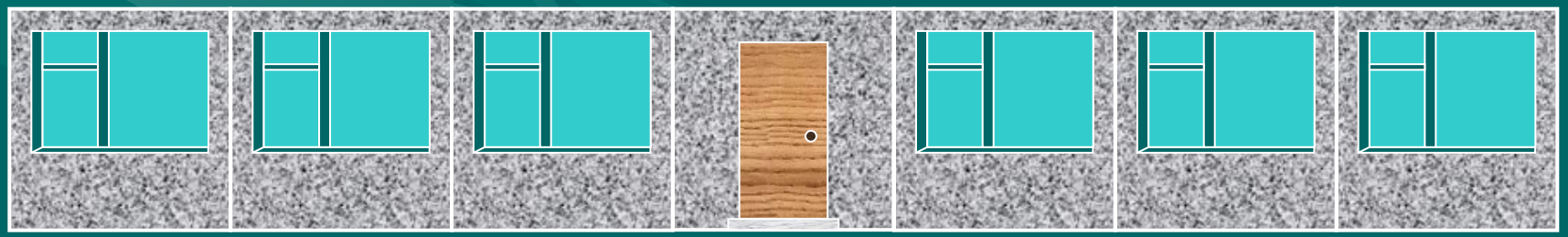
Построение расписания работы домостроительного комбината

Библиотека фрагментов

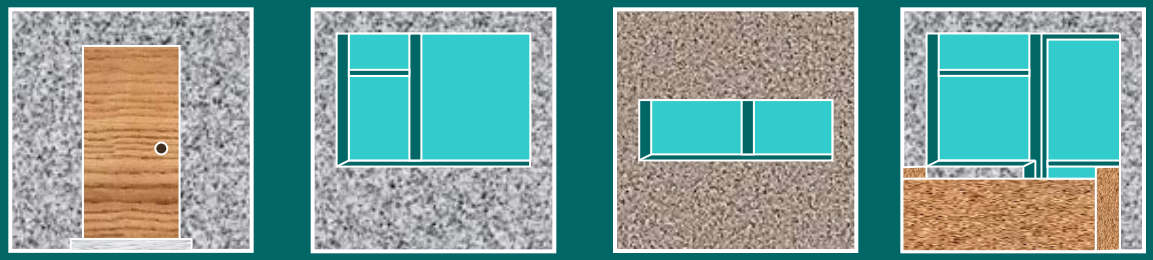





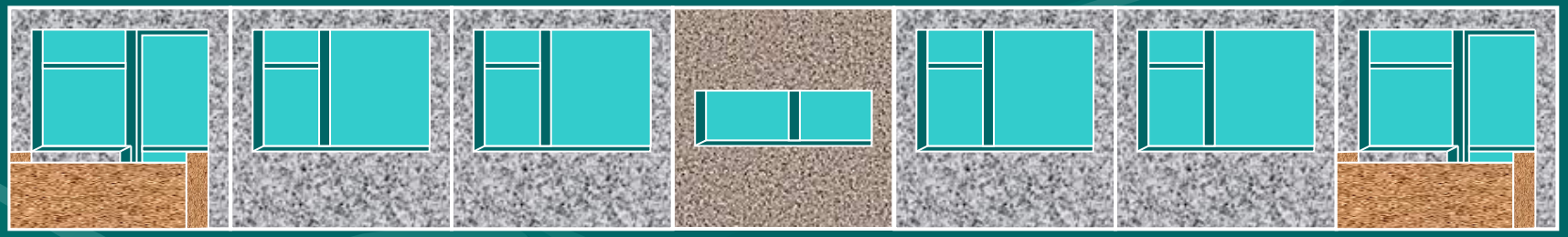
Библиотека фрагментов

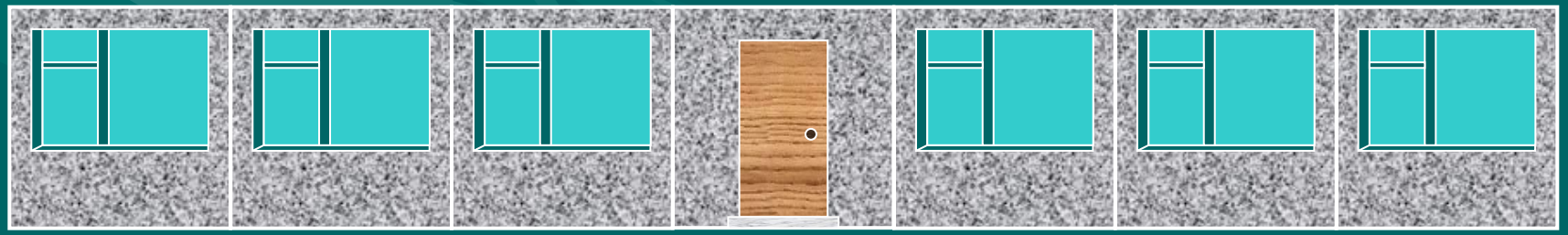







Библиотека фрагментов

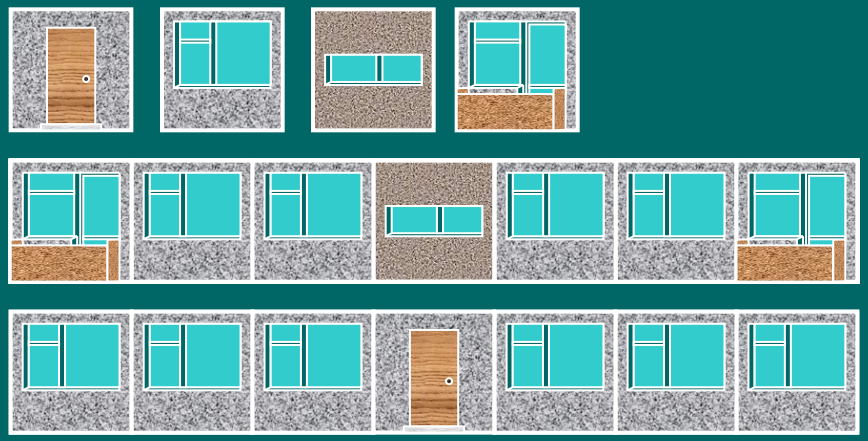




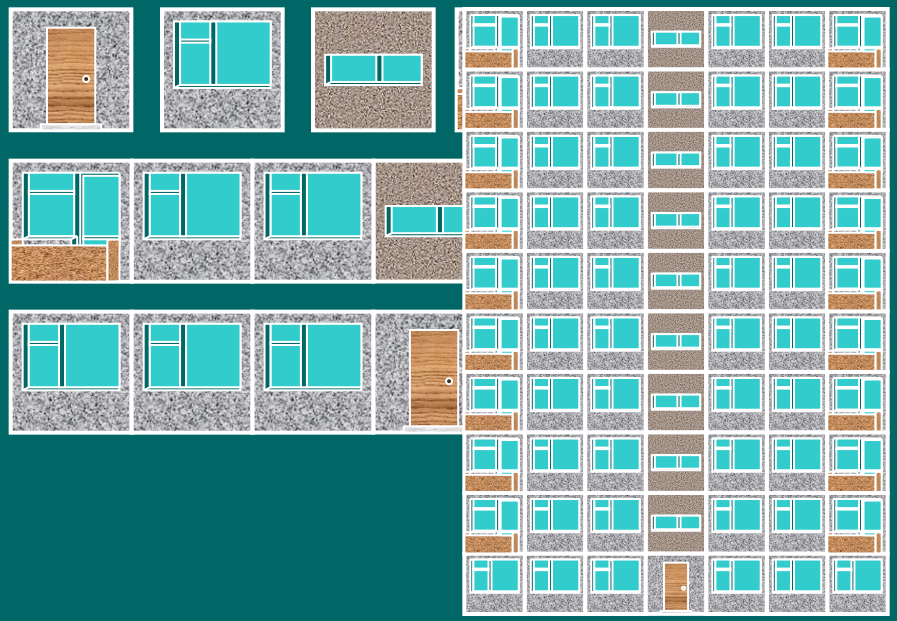


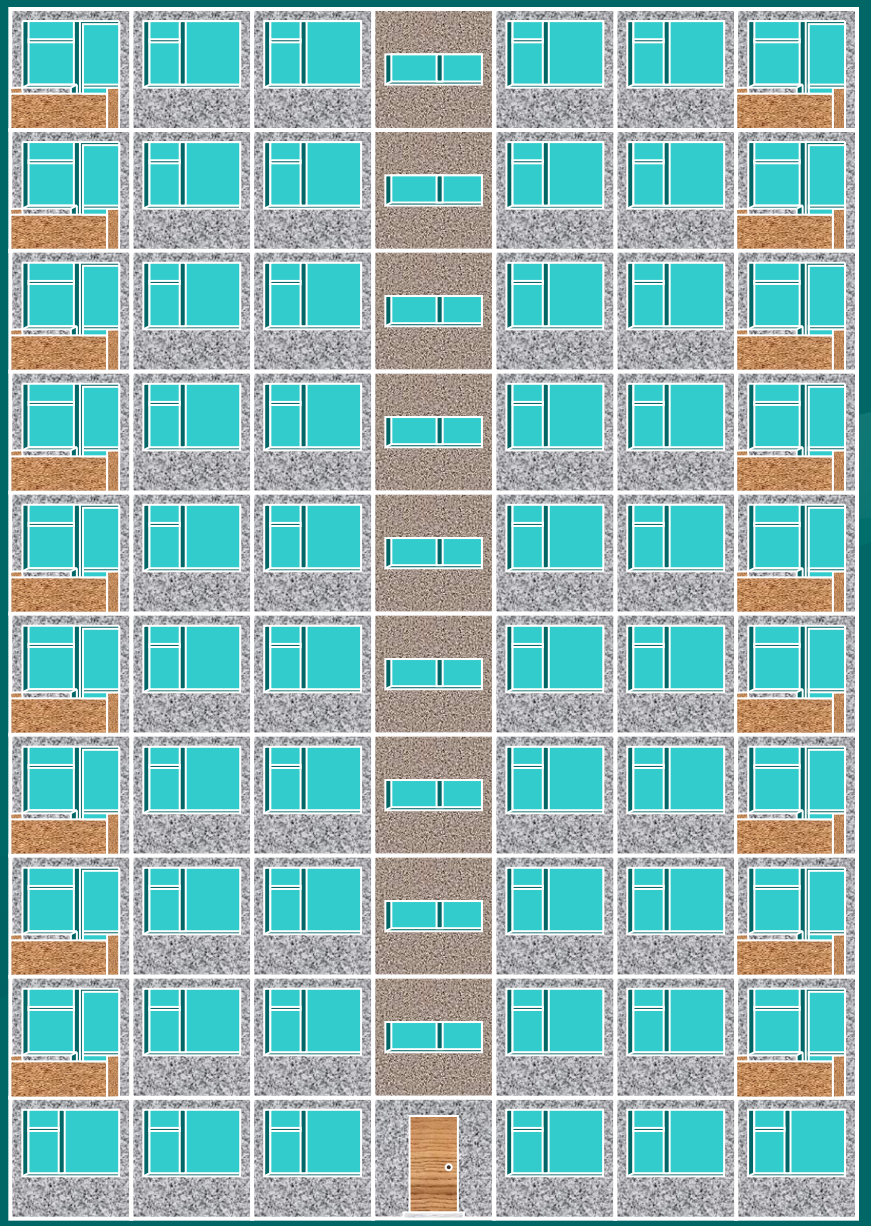
Библиотека фрагментов

Библиотека фрагментов

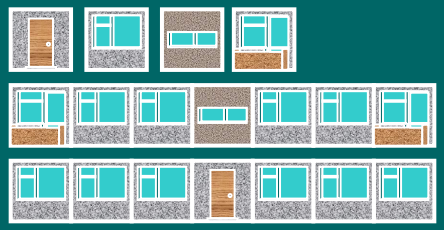


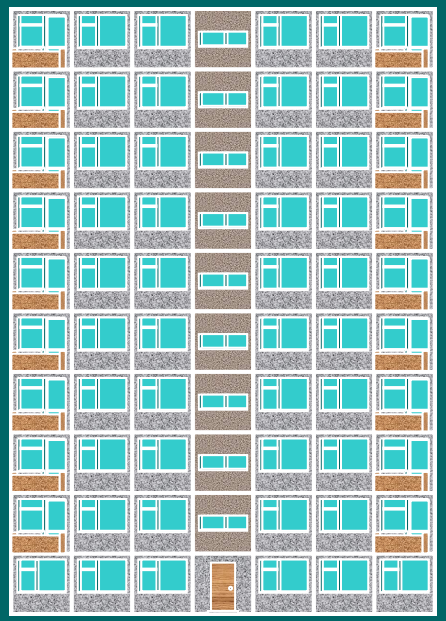
Библиотека фрагментов

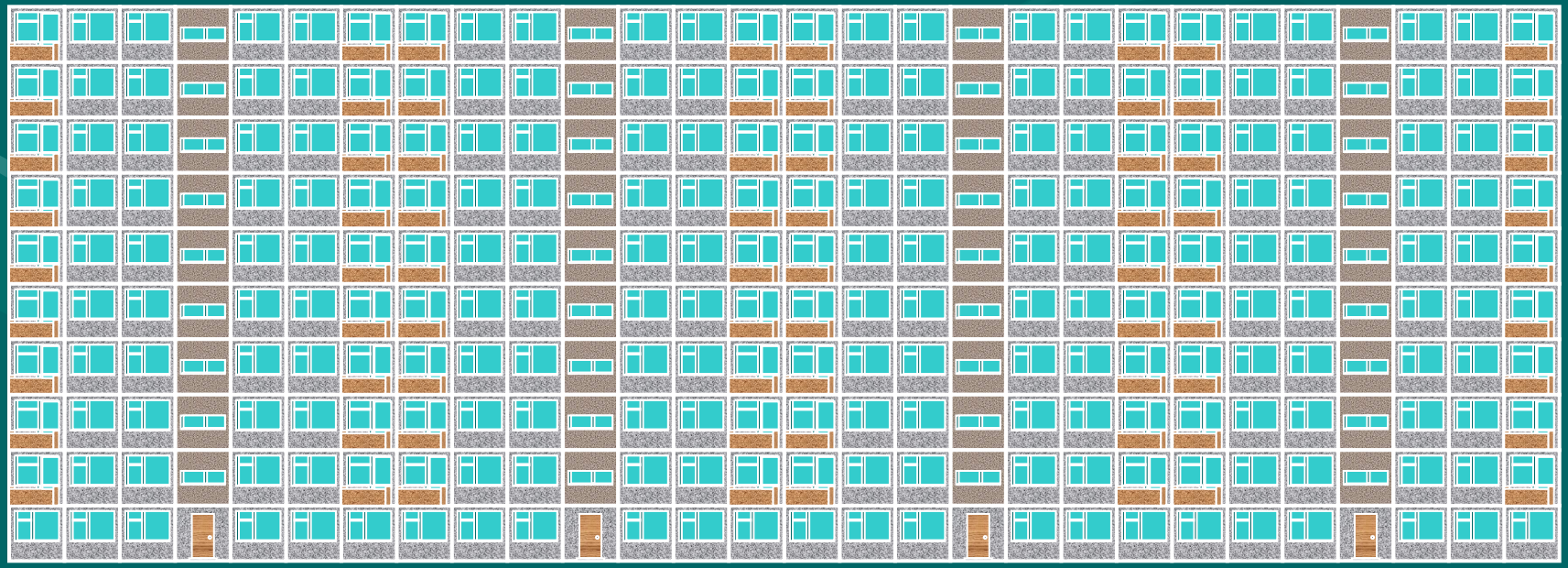




Библиотека фрагментов







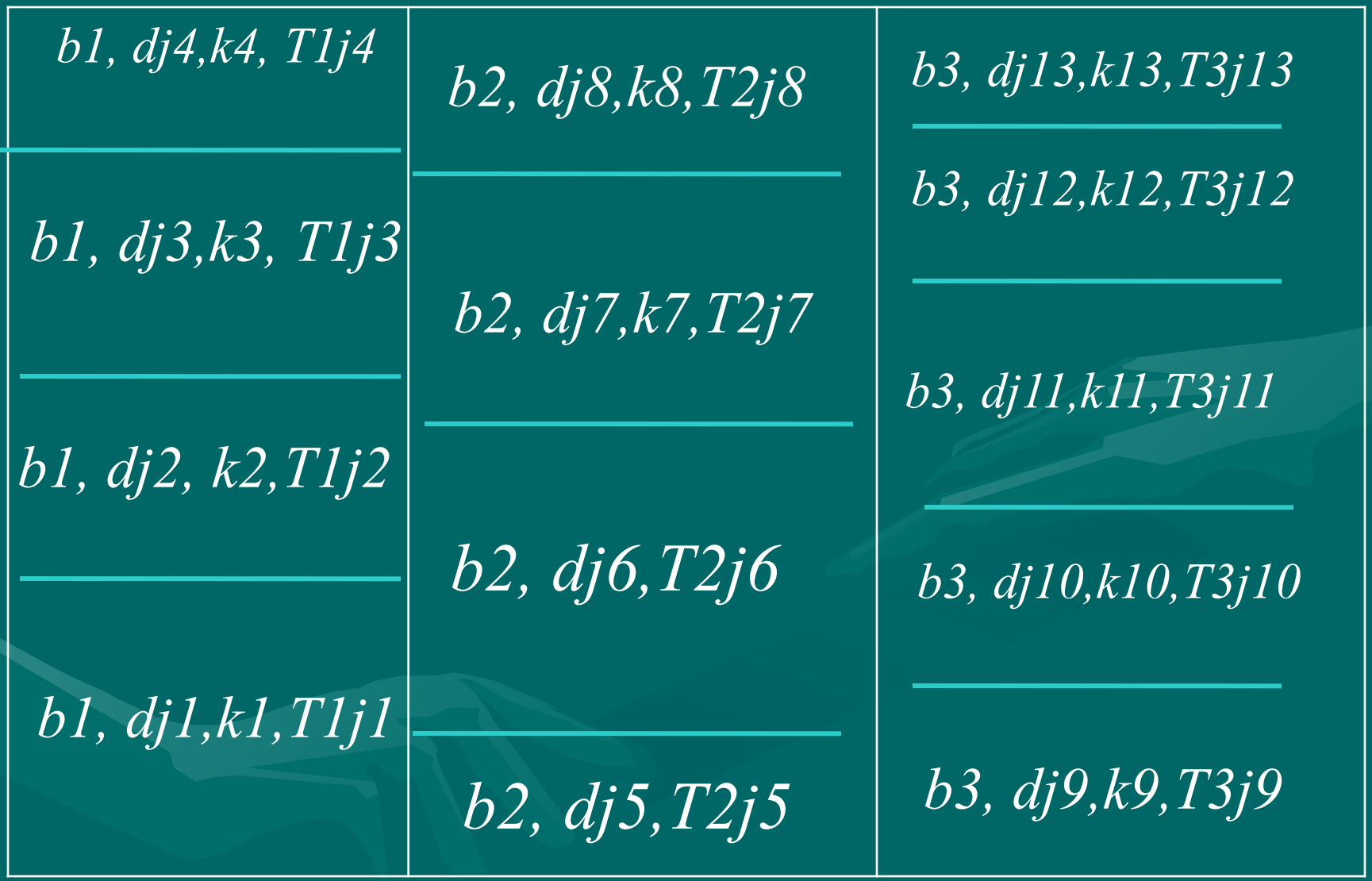
- Ещё один способ построения эвристического алгоритма и способ сокращения перебора
- **ДАНО**
  - Множество бригад *b1, b2, ... , br*
  - Множество домов *d1, d2, ... , dl*
  - Множество кранов *k1, k2, ... , kn, n>r*
  - Время работы t_{ij} бригады *bi* на сборке дома *dj*

РАСПИСАНИЕ - Таблица (*bi, dj, ki1, Tij*), где *Tij* – дата назначения бригады *bi* на сборку дома *dj*. Годится расписание с минимальным простоем бригад и кранов.

Самый дорогой ресурс – бригада, её простой недопустим

tk – время перебазирования крана *k* от дома *di* к дому

дни



b1 b2 b3 назначения

- Одновременно используются разные краны
- Время считается в днях.
- Дата $Tij = Tij + t_{ij}$ освобождения бригады bi после выполнения работы dj,
- Дата освобождения крана $Tij + tk_{ij}$. В разных расписаниях эти даты разное
- Оптимальное/субоптимальное расписание обычно не толерантно к невыполнению расписания.
- Алгоритм составления расписания должен быть в некоторой мере толерантен к сбоям в реализации расписания
- Учитывается готовность площадок и график готовности новых площадок
- Все бригады должны работать без простоев

Алгоритм формирования расписания

.Очередное назначение.

- Начинает строиться, когда появляется хотя бы одна готовая бригада *bi*, которая включается в назначение, время $Ti1j + tk_{ij}$
- Выбирается один из домов *dj*, площадка которого готова для начала строительства
- Выбирается кран *ki* из числа свободных
- Вычисляется время $Tij = Ti(j-1) + t_{ij}$ завершения строительства дома *bi*
- Вычисляется время $Tij + tk_{ij}$ освобождения крана *ki1* и с этого момента он помещается во множество свободных

Если готовых бригад, кранов или домов нет, время начала назначения отодвигается ($Tij + tk_{ij}$ увеличивается на время ожидания)

2. Затем делается первое назначение всех бригад.
3. Затем делается второе назначение для всех бригад. Бригады, которые на предшествующем шаге закончили работу раньше получают бóльшую работу. И так, пока все дома не построены, т.е. построено распределение.
4. Подсчитывается простой бригад и кранов.
5. Перебор. Далее меняется первое назначение и все последующие, вычисляется простой и так перебираются все распределения. Из них выбирается распределение с минимальным простоем. Это и есть оптимальное решение.
6. Фиксация назначения. При большом числе домов, которые надо построить, перебор становится невыполнимым. Для сокращения перебора делается фиксация частичного распределения.

Сокращение перебора, удовлетворение внесистемных требований. Добавление ограничений/требований

.Например, потребовать, чтобы некоторый конкретный дом был построен до 25 ноября.

.Некоторый специальный кран (например, для высотного строительства) должен быть свободен к 1 марта.

.Дома *bi* *u* *bj* должны начать строиться одновременно.

.Краны не должны пересекать реку.