МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ В ГЭК
И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ
Заведующий кафедрой
д.п.н., профессор
_____И.Г. Захарова
29 июня 2018 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

РАЗРАБОТКА МОДУЛЯ АВТОМАТИЗИРОВАННОГО ПЛАНИРОВАНИЯ И ОПЕРАТИВНОЙ КОРРЕКЦИИ ПЛАНА РАЗБУРИВАНИЯ ДЛЯ ИНТЕГРИРОВАННЫХ РЕШЕНИЙ КОМПАНИИ HALLIBURTON

02.04.03. Математическое обеспечение и администрирование информационных систем
Магистерская программа «Высокопроизводительные вычислительные системы»

Выполнил работу: Студент 2 курса очной формы обучения	*(Подпись)*	Калинин Алексей Андреевич
Научный руководитель: Зав. кафедрой, д.п.н., профессор	*(Подпись)*	Захарова Ирина Гелиевна
Консультант: Ст. преподаватель	*(Подпись)*	Аксенов Михаил Валерьевич
Рецензент: Доцент, к.т.н. ФГБОУ ВО «ТюмГУ»	*(Подпись)*	Донкова Ирина Адольфовна

Тюмень - 2018

Содержание

Введение

Одной из важнейших задач промышленного производства является планирование, исполнение и оперативная коррекция планов эксплуатации. Проработке, подтверждению и уточнению планов уделяется значительное внимание, так как нарушение планов означает проблемы в многочисленных сложных и взаимосвязанных процессах составляющих работу компании. Качество планирования, при этом, очень сильно зависит от компетентности составителя и сложности предметной области, так как в работе с материальными ресурсами необходимо предусматривать не только ограничения во времени или исполнителях, но и учитывать различные риски, которые неизбежны при работе реального предприятия. Необходимо отметить, что качественная проработка плана напрямую зависит от затраченного времени, но производству необходимы новые планы как можно скорее, так как работа вне плана может привести к значительным материальным потерям для компании. Это обуславливает необходимость сокращения лага между получением новых входных данных и принятием рационального плана.

С недавних пор компании нефтегазовой отрасли начали внедрять средства автоматизированного планирования на своих предприятиях для решения обозначенной проблемы, так как на данный момент планирование все еще является преимущественно ручным процессом, который выполняет специальный отдел, анализирующий требования бизнеса, законодательства и имеющиеся ограничения. На основании перечисленных входных данных отдел планирования составляет план по выделенному количеству буровых установок и бригад. Этот процесс повторяется в течение года несколько раз с появлением новой информации.

Постановка задачи

Целью данной работы является исследование и сравнение алгоритмов генерации и оптимизации расписаний для последующего включения наилучшего алгоритма в модуль автоматизированного планирования и оперативной коррекции плана разбуривания.

Для достижения поставленной цели были поставлены следующие задачи:

1) Изучение предметной области:

- Проблемы планирования:
    - Ограничения на ресурсы (оборудование, персонал, время).
- Процедура разбуривания и сопутствующая отчетность:
    - Технологический процесс;
    - Нормативные документы;
    - Регламент разбуривания;
    - Операционная отчетность.
- Изучение интегрированных решений Halliburton.

2) Исследование технологий для разработки модуля;

3) Исследование алгоритмов автоматизированного планирования:

- Планирование:
    - Последовательная и параллельная схемы составления расписания (SSGS, PSGS);
    - Первый подходящий (First Fit);
    - Размещение объекта из очереди (Allocate Entity From Queue);
    - Самая дешевая вставка (Cheapest Insertion);

- Метод ветвей и границ (Branch And Bound).

- Оптимизация:
  - Алгоритм имитации отжига (Simulated annealing);
  - Поиск с запретами (Tabu Search);
  - Поиск восхождением к вершине (Hill Climbing).

4) Разработка прототипа модуля планирования и оперативной коррекции плана;

5) Сравнение алгоритмов на соответствие функциональным и нефункциональным требованиям:

- Функциональные требования:
  - Генерация исходного расписания происходит в соответствии с требованиями;
  - Скорректированное расписание соответствует установленным требованиям.

- Нефункциональные требования:
  - Производительность.

6) Выбор алгоритмов, которые будут включены в конечную сборку модуля.

Глава 1. Обзор предметной области и алгоритмов решения

1.1. Математическая модель проблемы планирования

Математическая модель построена следующим образом:

$W=\{w_1, w_2,\dots ,w_q\}$ – множество скважин, где q – количество скважин;

$A^{type}=\{a_1^{type},a_2^{type}, \dots ,a_y^{type}\}$ – множество типов бригад, где y – количество типов бригад;

$A=\{a_1, a_2, \dots , a_u\}$ – множество бригад, где $a_{i'}\in a_{j'}^{type}$, $i'=\overline{1,u}$, $j'=\overline{1,y}$, u – количество бригад;

$M^{type}=\{m_1^{type}, m_2^{type}, \dots ,m_o^{type}\}$ – множество типов станков, где o – количество типов станков;

$M=\{m_1, m_2, \dots , m_p\}$ – множество станков, где $m_{i'^2}\in m_{j'^2}^{type}$, $i'^2=\overline{1,p}$, $j'^2=\overline{1,o}$ p – количество станков;

$R=\{\{a_{i'^3},m_{j'^3}\}, \{a_{i'^3+1},m_{j'^3+1}\}, \dots , \{a_{i'^3+c},m_{j'^3+c}\}\}$ – множество ресурсов, где $i'^3\in\overline{1,u}$, $j'^3\in\overline{1,o}$, c – количество ресурсов;

$R'=\{\{a_{i'^4}\cup m_{j'^4}\}, \{a_{i'^4+1}\cup m_{j'^4+1}\}, \dots , \{a_{i'^4+b}\cup m_{j'^4+b}\}\}$ – множество ресурсов требуемого типа (бригада и/или станок), где $i'^4\in\overline{1,u}$, $j'^4\in\overline{1,o}$, b – количество требуемых ресурсов.

$E^{type}=\{e_1^{type}, e_2^{type}, \dots ,e_h^{type}\}$ – множество типов работ, где h – количество типов работ;

$T=\{t_1, t_2, \dots , t_n\}$ – множество времени начала работ;

$D=\{d_1, d_2, \dots , d_n\}$ – множество времени продолжительности работ;

$E=\{\{e_{i'^5}^{type},t_1,d_1,r_{j'^5}'\}, \{e_{i'^5+1}^{type},t_2,d_2,r_{j'^5+1}'\}, \dots , \{e_{i'^5+v}^{type},t_n,d_n,r_{j'^5+p}'\}\}$ – множество работ, где $i'^5\in\overline{1,h}$, $j'^5\in\overline{1,b}$, v – количество типов работ, p – количество требуемых ресурсов, n – количество работ;

S – множество расписаний, где $s_i=\{\{r_1,\{e_{i,1}, e_{i+1,1}, \dots, e_{i+l,1}\}\}, \{r_2,\{e_{i,2}, e_{i+1,2} \dots, e_{i+l,2}\}\}, \dots, \{r_z,\{e_{i,z}, e_{i+1,z} \dots, e_{l,z}\}\}$. Причем $\forall \{e_{i,j}, e_{i+1,j}, \dots, e_{i+l,j}\} \exists \{r'_{i,j}, r'_{i+1,j} \dots r'_{i+l,j}\} \in r_j$, где $i \in \overline{1,n}$, $j \in \overline{1,z}$, l – количество работ для одного ресурса, z – количество используемых ресурсов.

В теории расписаний используется следующая формулировка проблема планирования. Существует некоторый проект, состоящий из множества работ $E = \{e_1, e_2, \dots, e_n\}$, каждая из которых характеризуется некоторым номером/индексом $i = \overline{1,n}$ и временем своего выполнения/продолжительностью d_i ($d_i \geq 0$). На работы может накладываться так называемое ограничение предшествования - технологическое ограничение, при котором невозможно начать выполнение работы ранее, чем завершатся предыдущие, связанные с ней, работы – предшественники. Данная работа является последователем по отношению к своим предшественникам. Множество, состоящее из всех предшественников работы e_i, обозначается как P_i. Взаимозависимости между работами не должны иметь циклический характер.

В проекте может существовать K возобновляемых ресурсов. Максимальный доступный объем каждого ресурса $k = \overline{1,K}$ ограничен константой X_k. Каждая работа e_i может требовать для своего выполнения некоторое количество x_{ik} k-ого ресурса, причем $x_{ik} \leq X_k$. Другими словами, x_{ik} не может превышать допустимый объем X_k k-ого ресурса. Для выполнения работы может требоваться несколько ресурсов. Требование работы e_i к k-ому ресурсу означает, что с началом выполнения работы e_i k-ый ресурс в количестве x_{ik} считается не доступным для выполнения других работ. После окончания выполнения работы данное количество ресурса высвобождается.

Работы, которые имеют нулевую продолжительность, не требуют для своего выполнения ресурсов (время начала выполнения работы совпадает со временем завершения). Если работа была начата, ее выполнение не прерывается (работа не может приостановиться и высвободить все потребляемые ресурсы).

В проекте существуют две дополнительные фиктивные работы с нулевой продолжительностью (e_0 и e_{n+1}), не требующие ресурсов. Данные работы соответствуют началу и завершению всего проекта и отражают временные рамки выполнения всего проекта. Работа e_0 – предшественник всех работ, ранее не имевших предшественников, а работа e_{n+1} - последователь всех работ, ранее не имевших последователей.

Время t_i начала и время f_i завершения работы e_i считаются неизвестными, причём $f_i = t_i + d_i$. Задача RCPSP ставит своей целью вычисление всех значений t_i и/или f_i таких, чтобы время выполнения всего проекта $F = \max_{\overline{1,n}} f_i$ было минимальным из всех возможных [1].

На планирование разбуривания накладываются следующие ограничения:

1. Объем бурения – объем, который бригадам предстоит выполнить с учетом допустимого оборудования.
2. Количество скважин – сколько скважин необходимо разбуривать.
3. Количество кустов – сколько скважин необходимо разбуривать.
4. Распределение/очередность скважин по кустам – порядок, в котором необходимо выполнять разбуривания на скважинах.
5. Продолжительность бурения – время, за которое будет осуществлено бурение.
6. Продолжительность передвижки внутри куста.
7. Время монтажа.
8. Время монтажа/мобилизации – время, затраченное на монтаж.
9. Время демонтажа/демобилизации – время, затраченное на демонтаж.
10. Продолжительность завоза/вывоза бурового станка – время, затраченное на логистику буровых станков.
11. Продолжительность завоза/вывоза буровой бригады/вышкомонтажной бригады – время, затраченное на логистику бригад.

1.2. Процедура разбуривания и сопутствующая отчетность

Производственный цикл сооружения скважин включает следующие элементы [5]:

1) подготовка строительной площадки;
2) транспортировка и монтаж вышки и бурового оборудования;
3) подготовка к бурению;
4) бурение скважины и ее крепление (разбуривание, тампонажные и др. работы);
5) испытание скважины на продуктивность;
6) демонтаж наземного оборудования, разборка вышки на блоки и прочие подготовительные работы для перетаскивания бурового оборудования на новый объект.

В строительстве скважин участвует несколько бригад, выполняющих работы по бурению, креплению, освоению и вышкостроению.

Бригады бурения проводят подготовительные работы к бурению скважины, испытание первого объекта (горизонта). Бригады вышкостроения занимаются перебазировкой вышки и бурового оборудования между скважинами. Бригады испытания (освоения) проводят испытание последующих (второго, третьего и т.д.) объектов в разведочных скважинах.

Производственный цикл сооружения скважин проходит в следующей очередности. Буровую бригаду после завершения работ на текущей скважине переводят на другой объект, где после двух–трех дней подготовительных работ к бурению, начинается проходка скважины. Если пробуренная скважина имеет несколько объектов испытания, то на скважину приходит бригада освоения и с помощью специального оборудования проводит испытание.

После завершения работ буровой бригадой осуществляется мобилизация бурового оборудования на следующую скважину. Сроки завершения строительно-

монтажных работ планируются так, чтобы не допускать простоя буровой бригады из-за неподготовленности оборудования к бурению.

Все скважины по своему назначению подразделяются на эксплуатационные, разведочные и прочие (наблюдательные, параметрические и т. д.).

Разведочные скважины служат для оценки запасов новых месторождений (залежей) нефти по категориям, а эксплуатационные предназначены для подъема нефти на поверхность.

Деятельность организаций по строительству скважин имеет следующие особенности:

1. Различные характеристики и параметры сооружаемых скважин, обусловленные влиянием природных факторов.
2. Разобщенность объектов строительства по обширной территории и связанная с этим подвижность орудий труда и исполнителей, перемещающихся вдоль фронта работ.
3. Рациональная специализация рабочих по бригадам бурения, испытания, вышкостроения.
4. Разная продолжительность строительства скважин при наличии типовой технологии.

Указанные особенности приводят к необходимости повышения уровня планирования в буровой организации с учетом непрерывной координации всех элементов строительного процесса, что должно найти отражение при составлении производственной программы.

Показатели производственного плана, выраженные натуральными измерителями, формируются в виде плана-графика бурения скважин.

На основании плана-графика разрабатываются задания всем подразделениям основного и вспомогательного производства, а также определяются услуги сторонних организаций (промыслово-геофизические

работы, транспорт) и потребность в материально-технических и топливно-энергетических ресурсах.

Показатели графика бурения скважин одновременно используются как основные исходные данные для разработки планов:

1. По численности и заработной плате;
2. По затратам и прибыли;
3. По капитальному строительству;
4. По снабжению материально-техническими ресурсами.

На практике разработку плана-графика строительства скважин осуществляют планово-экономические службы буровых организаций и согласовывают с геологической службой, отделами главного энергетика и главного механика.

Последовательность составления графика бурения скважин, принятая на практике, следующая:

1) Служба планирования вышестоящей организации устанавливает количество законченных бурением и сданных заказчику скважин. При этом количество эксплуатационных скважин определяется на основании плановых заданий по объему добычи нефти, а разведочных – из требования ежегодного прироста запасов нефти;
2) На основании данных показателей устанавливается ориентировочная очередность бурения скважин и в форме заявочного списка передается в плановый отдел буровой организации.
3) В плановом отделе формируется окончательный вариант плана-графика бурения скважин с указанием сроков выполнения работ по каждой скважине, а также буровой бригады и типа бурового оборудования.

1.3. Сетевая модель задачи планирования

Сетевая модель является моделью реализации комплекса работ представленная в виде ориентированного графа, отображающего порядок выполнения работ на некотором временном промежутке. Сетевая модель может содержать следующие характеристики, относящиеся к отдельным работам или ко всем: время, ресурсы и другие [1].

Сетевую модель для задачи RCPSP можно описать следующим образом. На множестве работ $E=\{e_1, e_2, \dots e_n\}$ задано отношение предшествования, обозначаемое «→». Вершины графа соответствуют работам. Вершины e_i и e_{i+1} соединены ребром, из e_i в e_{i+1}, когда $e_i \rightarrow e_{i+1}$. Для связности графа добавляются фиктивные работы начала и окончания всех работ, т.е. $e_0 \rightarrow e_i$, $e_{i+1} \rightarrow e_{n+1}$. Например, для 3 ресурсов и неограниченного количества работ будет получен следующий граф (Рис. 1.1). Для каждой работы соответственно выполняется условие предшествование, что $t_{i+1} \geq t_i + d_i$.

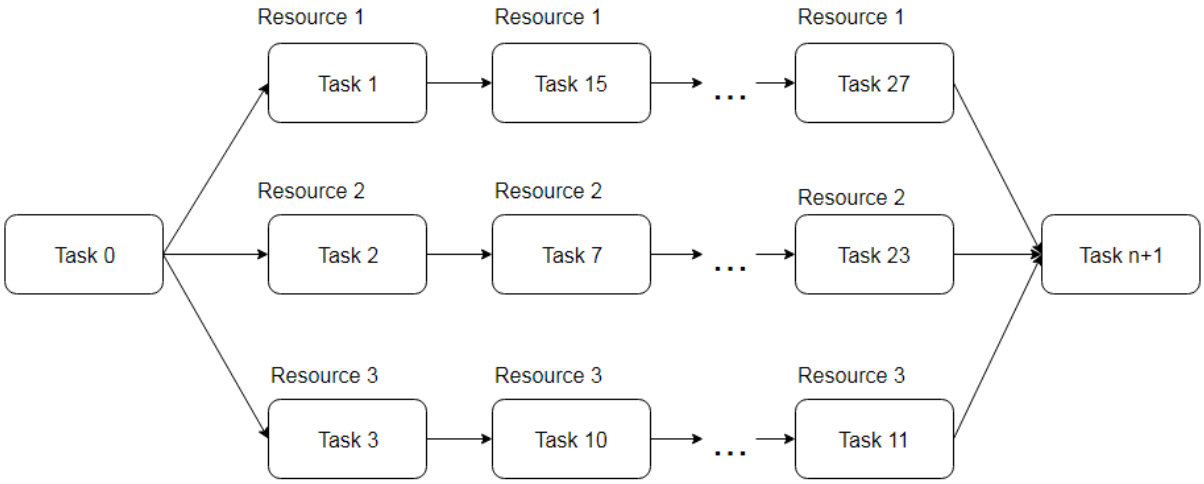


Рис. 1.1. Пример сетевой модели для плана разбуривания

1.4. Математическая постановка задачи планирования

Для решения проблемы планирования из теории расписаний были сформулированы следующие постановки задач [1]. Формулировка математической модели описано выше (1.1. Математическая модель проблемы планирования):

- Задача нахождения допустимого расписания:

  E – множество работ. Длительность выполнения $d_i > 0$, время поступления работы на выполнение $t'_i \geq 0$ и крайний срок $D^{end} > 0$. Необходимо построить допустимое расписание выполнения работ, при котором момент начала выполнения t_i такой, что $t_i \geq t_i'$, момент окончания выполнения $C_i = t_i + d_i \leq D^{end}$, $t_i < t_{i+1}$, $t_i + d_i \leq t_{i+1}$.

- Задача нахождения оптимального расписания:

  Имеется задача допустимого расписания. Пусть $D^{end} = +\infty$, $i = \overline{1,n}$; C_i – момент окончания выполнения работы i. Необходимо построить допустимое расписание, при котором значение C_i будет минимальным.

1.5. Функция оценки качества расписания

Оценка качества расписания является неотъемлемой частью процесса планирования, так как, не проведя оценку всех факторов, влияющих на расписание, трудно проверить соответствие плана поставленным целям. Поэтому для контроля качества расписания вводят функции оценки качества.

Для оценки расписания можно выделить два вида требований: допустимые и оптимальные. Допустимая требования содержит условия, которые не могут быть нарушены. Оптимальные требования могут быть нарушены. Для удобной оценки, требования разделяются на две соответствующие оценки: допустимая оценка и оптимальная оценка. Допустимая оценка, при правильности расписания, всегда должна равняться нулю. В противном случае расписание считается непригодным. Оптимальная оценка, напротив, может отличаться от нуля, однако должна стремиться к нулю.

Для модели, описанной выше (1.1. Математическая модель проблемы планирования), были построены следующие оценки:

1) Допустимые оценки:

- На одном временном промежутке бригады не повторяются:

$\forall s_i$, где $t_j = t_l \nexists e_j \neq e_l : a_v = a_z$;

- На одном временном промежутке станки не повторяются:

  $\forall s_i$, где $t_j = t_l \nexists e_j \neq e_l : m_v = m_z$;

- Ресурс соответствует данной работе:

  $\forall s_i \exists r_j' \in r_l$;

2) Оптимальные оценки:

- Время окончания работ минимально:

  $\forall s_i$, $(\sum_{j=1}^{n} (t_j + d_j)) \rightarrow \min$.

В дальнейшем функция оценки будет иметь следующие обозначение: $G(s_i) = \{g^{hard}(s_i), g^{soft}(s_i)\}$, где G – функция оценки качества расписания, s_i – расписание из множества S.

1.6. Методы составления расписания

Все методы, связанные с составлением расписания, построены на последовательной и параллельной схеме формирования расписания.

1.6.1. Последовательная схема составления расписания

$E = \{e_1, e_2, \ldots, e_n\}$ список работ и список ресурсов $R = \{r_1, r_2, \ldots, r_m\}$. На каждом i-ом шаге цикла планирования рассматривается множество работ E'_i, для которых в построенном, на предыдущих шагах, частичном расписании s_{i-1} выполнены ограничения предшествования, т.е. $e_j \in E'_i$, тогда и только тогда когда все работы из предшествующих e_j запланированы в s_{i-1}. С помощью некоторого эвристического правила среди работ E'_i выбирается одна планируемая работа e_j и вычисляется ближайшее время начала этой работы (в соответствии с ограничениями предшествования). Из множества ресурсов R выбирается такое подмножество R(t), которое содержит ресурсы, недоступные для планирования работы e_j в момент времени t.

Для каждого недоступного ресурса из множества R(t) определяется ближайший момент времени t'_{rk}, в который ресурс будет обладать достаточным объемом для планирования работы e_j. Минимальное время t_{min} из множества $\{t'_{rk},\}$, в которое работа может начаться, не нарушая ресурсных ограничений, является временем начала работы e_j и обозначается как t_j. Работа e_j вносится в частичное расписание s_i с временем начала $t_j=t_{min}$ и временем окончания $f_j = t_j + d_j$, где d_j – продолжительность работы e_j. После коррекции профилей доступности ресурсов, множество работ E'_i переопределяется, и цикл планирования продолжается, пока все работы не будут запланированы [6] (Рис. 1.2).

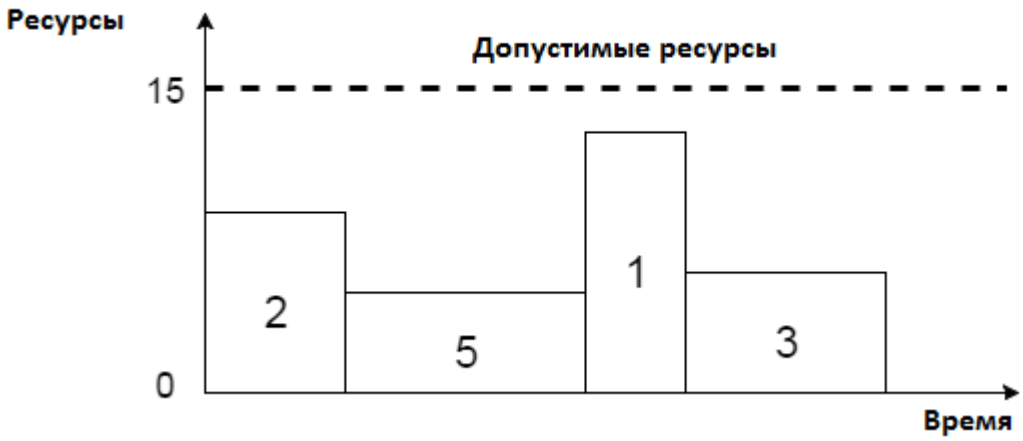


Рис. 1.2. Последовательная схема составления расписания

1.6.2. Параллельная схема составления расписания

$E=\{e_1,e_2,\ldots,e_n\}$ список работ и $R=\{r_1, r_2,\ldots, r_m\}$ список ресурсов. Использование параллельной схемы позволяет одновременно запланировать несколько работ, стартующих в один момент времени.

На каждом шаге цикла планирования отсчитывается левый край t_i нового частичного расписания s_i, таким образом определяется ближайший момент времени, в который может быть начато выполнение хотя бы одной работы из E'_i. Работы, которые нарушают какие-либо ограничения и не могут быть

запланированы в момент t_i удаляются. Оставшиеся работы добавляются в расписание s_i [6] (Рис. 1.3).

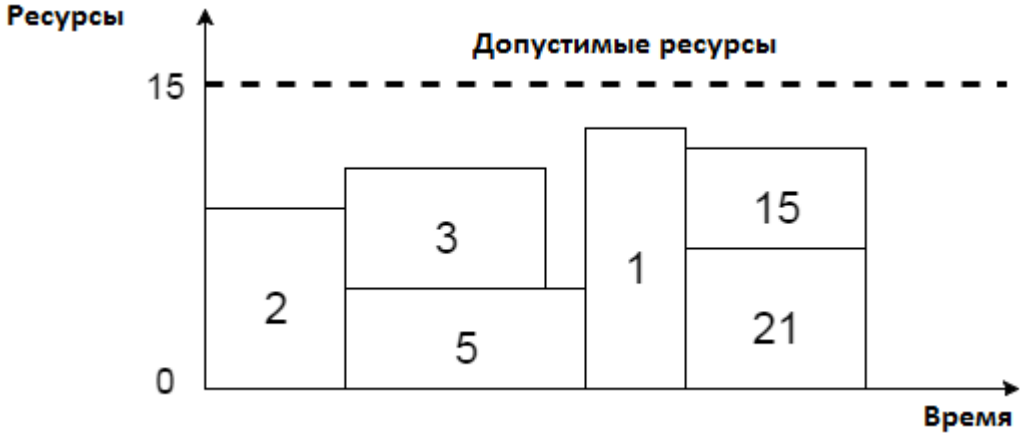


Рис. 1.3. Параллельная схема составления расписания

1.6.3. Первый подходящий

Алгоритм первый подходящий обходит все объекты планирования, инициализирую один объект за раз. Алгоритм состоит из следующих шагов [9] (Рис. 1.4):

1) Для расписания $s_0 = \{\emptyset\}$ выбирается такое расписание s_j, что $s_j = s_0 \cup \{\{r_j, \{e_{i,j}, e_{i+1,j}, \dots, e_{i+l,j}\}\}\}$ и $g(s_j) = \max$ на данный момент.

2) Для s_j аналогично выбирается следующее расписание $s_{j+1} = s_j \cup \{\{r_{j+1}, \{e_{i,j+1}, e_{i+1,j+1}, \dots, e_{i+l,j+1}\}\}$.

3) Повтор шага два, пока не будет получено расписание s_n, которое содержит все работы из множества работ E.

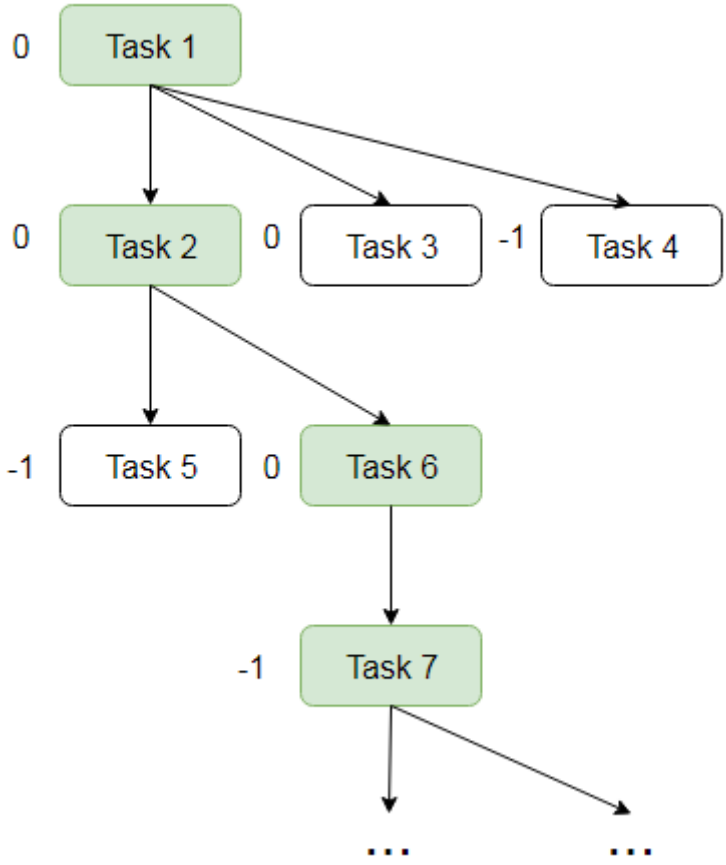


Рис. 1.4. Схема алгоритма первый подходящий

Вычислительная сложность $O(N^2)$.

1.6.4. Размещение объекта из очереди

Является универсальной формой таких алгоритмов, как первый подходящий, слабо подходящий и сильно подходящий. Алгоритм состоит из следующих шагов [9] (Рис. 1.5):

1) Ресурсы и работы помещаются в очередь.

2) Для расписания $s_0 = \{\emptyset\}$ выбирается такая комбинация ресурса и работ $\{r_j,\{e_{i,j}, e_{i+1,j}, \dots, e_{i+l,j}\}\}$ из очереди, что $s_j = s_0 \cup \{\{r_j,\{e_{i,j}, e_{i+1,j}, \dots, e_{i+l,j}\}\}\}$ и $g(s_j) = \max$.

3) Для s_j аналогично выбирается из очереди следующая комбинация $\{r_j,\{e_{i,j}, e_{i+1,j}, \dots, e_{i+l,j}\}\}$, что $s_{j+1} = s_j \cup \{\{r_{j+1},\{e_{i,j+1}, e_{i+1,j+1}, \dots, e_{i+l,j+1}\}\}$.

4) Повтор шага два, пока не будет получено расписание s_n, которое содержит все работы из множества работ E.

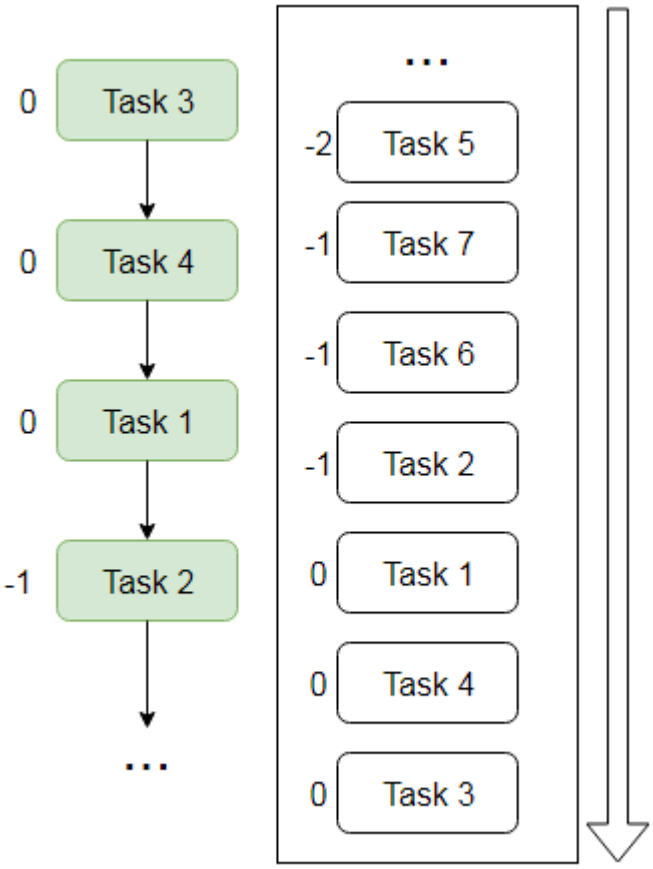


Рис. 1.5. Схема алгоритма размещения из очереди

1.6.5. Самая дешевая вставка

Данный алгоритм проходит по всем объектам планирования, инициализирую один объект планирования за раз. Объекту назначается наилучшее доступное значение (с учетом всех объектов и значений планирования). Алгоритм содержит следующие шаги [9] (Рис. 1.6):

1) Для расписания $s_0 = \{\emptyset\}$ выбирается такое расписание s_j, что $s_j = s_0 \cup \{\{r_j, \{e_{i,j}, e_{i+1,j}, \dots, e_{i+l,j}\}\}\}$ и $g(s_j) = \max$.

2) Для s_j аналогично выбирается следующее расписание $s_{j+1} = s_j \cup \{\{r_{j+1}, \{e_{i,j+1}, e_{i+1,j+1}, \dots, e_{i+l,j+1}\}\}$. Причем функция $g(s_{j+1}) = \max$ среди всех расписаний, содержащих s_j расписание.

3) Повтор шага два, пока не будет получено расписание s_n, которое содержит все работы из множества работ E.

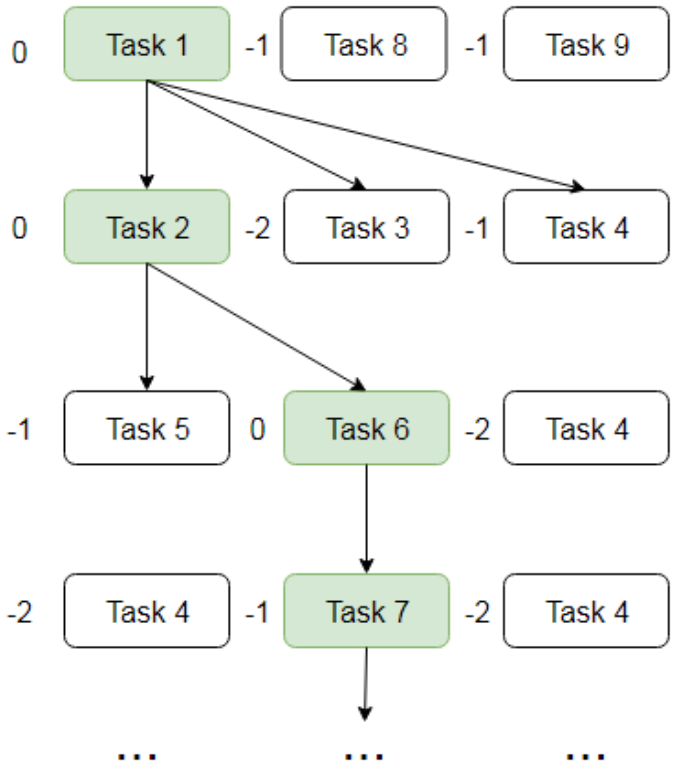


Рис. 1.6. Схема алгоритма самая дешевая вставка

Вычислительная сложность $O(N^2*(N+1)/2)$.

1.6.6. Метод ветвей и границ

Алгоритм для планирования генерирует дерево поиска решения, которое экспоненциально увеличивается по мере разрастания проблемы. Особенность алгоритма заключается в отсеивании на определенном шаге заранее не перспективных узлов. Принцип работы алгоритма следующий:

1) Для расписания $s_0 = \{\emptyset\}$ строятся возможные на данном шаге расписания $s_j = s_0 \cup \{\{r_j, \{e_{i,j}, e_{i+1,j}, \dots, e_{i+l,j}\}\}\}$ и выбираются все расписания $\{s_j, s_{j+1}, \dots, s_{j+k}\}$, у которых $\{g(s_j), g(s_{j+1}), \dots, g(s_{j+k})\} > \min$. Расписания с минимальной оценкой отбрасываются.

2) Для s_j аналогично выбираются расписания $s_{j+1} = s_j \cup \{\{r_{j+1}, \{e_{i,j+1}, e_{i+1,j+1}, \dots, e_{i+l,j+1}\}\}$.

3) Повтор шага два, пока не будет получено расписание s_n, у которого $g(s_n)=\max$, либо $n > c_{step}$, где c_{step} – количество шагов, при которых $g(s_n)$ остается неизменной.

Таким образом, генерируется $N^{(N-i)}$ узлов с начальными решениями. Затем каждый узел делится еще на $N^{(N-i)}$ узлов. Для каждого вычисляется оптимистичная оценка. Если данная оценка становится равной или ниже пессимистической оценки, то данный узел отсекается. Под пессимистической оценкой понимается минимальная оценка из просмотренной ветви [9] (Рис. 1.7).

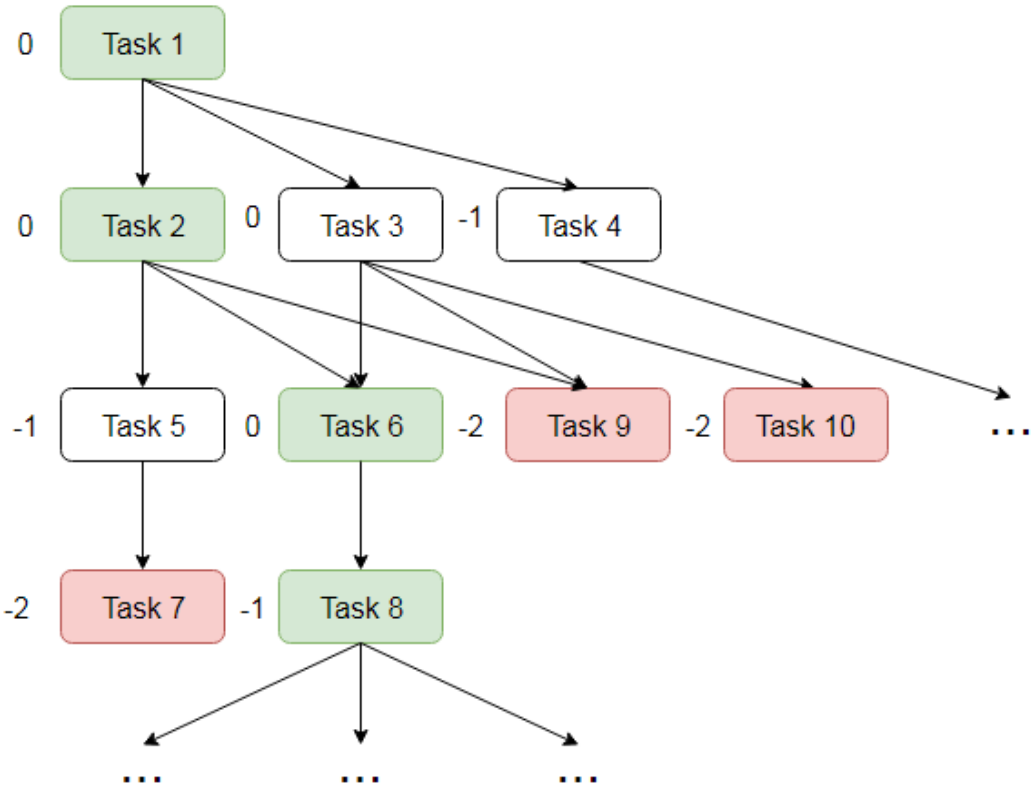


Рис. 1.7. Схема метода ветвей и границ

Вычислительная сложность $O(N^{N-i})$ где i – номер итерации.

1.7. Методы оптимизации расписания

1.7.1. Алгоритм имитации отжига

Алгоритм имитации отжига – алгоритм моделирующий процесс, протекающий в металлах при понижении температуры.

Поиска решения методом имитации отжига заключается в следующем [3, 4,9] (Рис. 1.8):

1) Генерируется новое расписание s_{j+1};

2) Выбор нового расписания текущим с вероятностью p=(ΔE,T), где $\Delta E = g(s_{j+1}) - g(s_j)$ – приращение энергии, T – температура, иначе повторяется процесс генерации нового расписания $s_k = s_j \cup \{\{r_k, \{e_{i,k}, e_{i+1,k}, \dots, e_{i+l,k}\}\}\}$. T→0.

3) Если T=0 или $n > c_{step}$, где c_{step} – количество шагов, при которых $g(s_n)$ остается неизменной.

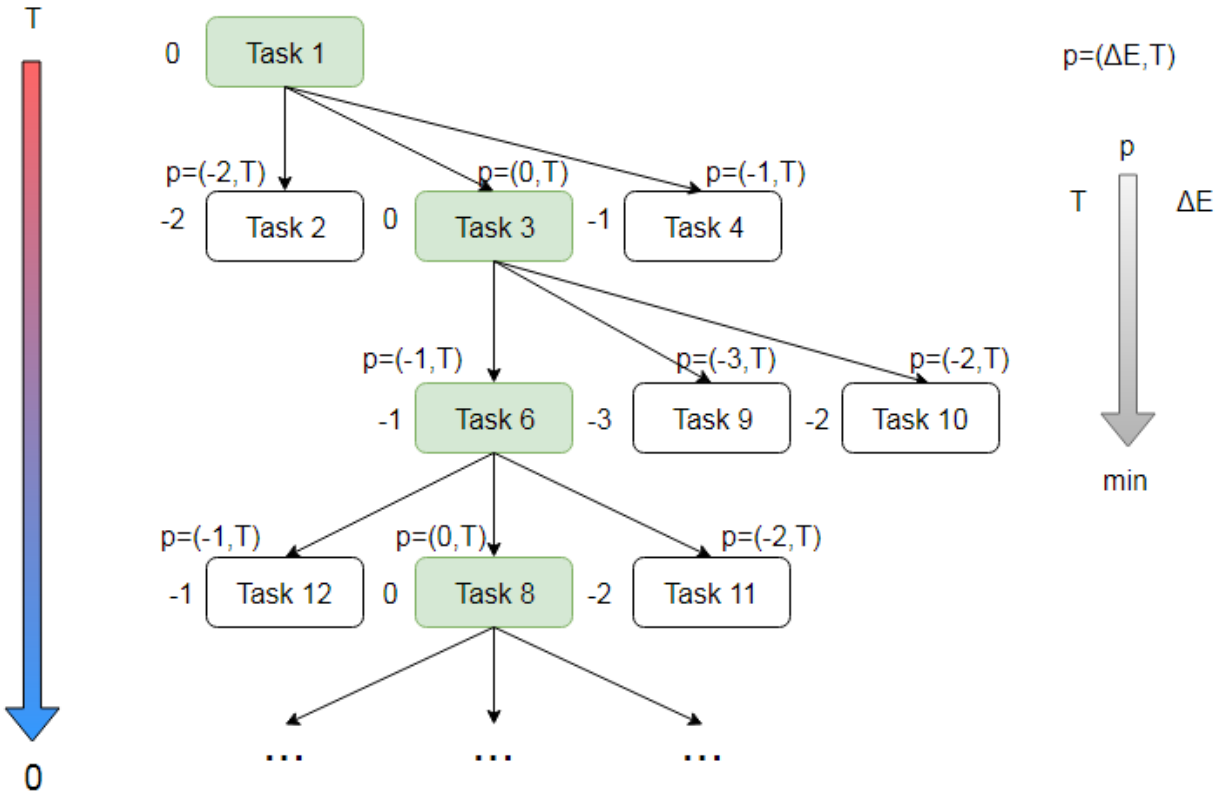


Рис. 1.8. Схема имитации отжига

Таким образом, если новое состояние дает лучшее значение оптимизируемой функции, то переход в данное состояние произойдет в любом случае. Поиск минимума целевой функции повторяется до тех пор, пока температура не уменьшится до некоторого заданного значения.

1.7.2. Поиск с запретами

Алгоритм, используемый для решения проблем оптимизации в теории графов и других областях. Суть алгоритма заключается в запрете на изменение той части решения, для которой на ближайших N шагах были введены изменения.

Схема алгоритма следующая [9] (Рис. 1.9):

1) Выбор начального расписания s_j. Очередь запретов $P=\{\emptyset\}$;

2) Выбор расписания $s_{j+1}=s_j \cup \{\{r_{j+1},\{e_{i,j+1}, e_{i+1,j+1}, \dots, e_{i+l,j+1}\}\}\}$ из множества $\{s_{j+1}, s_{j+2}, \dots, s_{j+n} | s_j \in s_{j+l}, l = \overline{1,n}\}$, где $g(s_{j+1})=\max$. Если имеется несколько расписаний с одинаковой оценкой, то выбирается одно случайное;

3) В очередь запретов заносится измененная часть решения. $P=P \cup \{r_{j+1},\{e_{i,j+1}, e_{i+1,j+1}, \dots, e_{i+l,j+1}\}\}$. Если количество элементов в очереди больше c_{count}, то убирается первый элемент из очереди $P=P \setminus \{r_{j+1},\{e_{i,j+1}, e_{i+1,j+1}, \dots, e_{i+l,j+1}\}\}$ и добавляется новый $P=P \cup \{r_{j+2},\{e_{i,j+2}, e_{i+1,j+2}, \dots, e_{i+l,j+2}\}\}$ где c_{count} – максимальное количество изменений в очереди.

4) Если $n > c_{step}$, где c_{step} – количество шагов, при которых $g(s_n)$ остается неизменной, то алгоритм окончен, иначе переход на шаг 2.

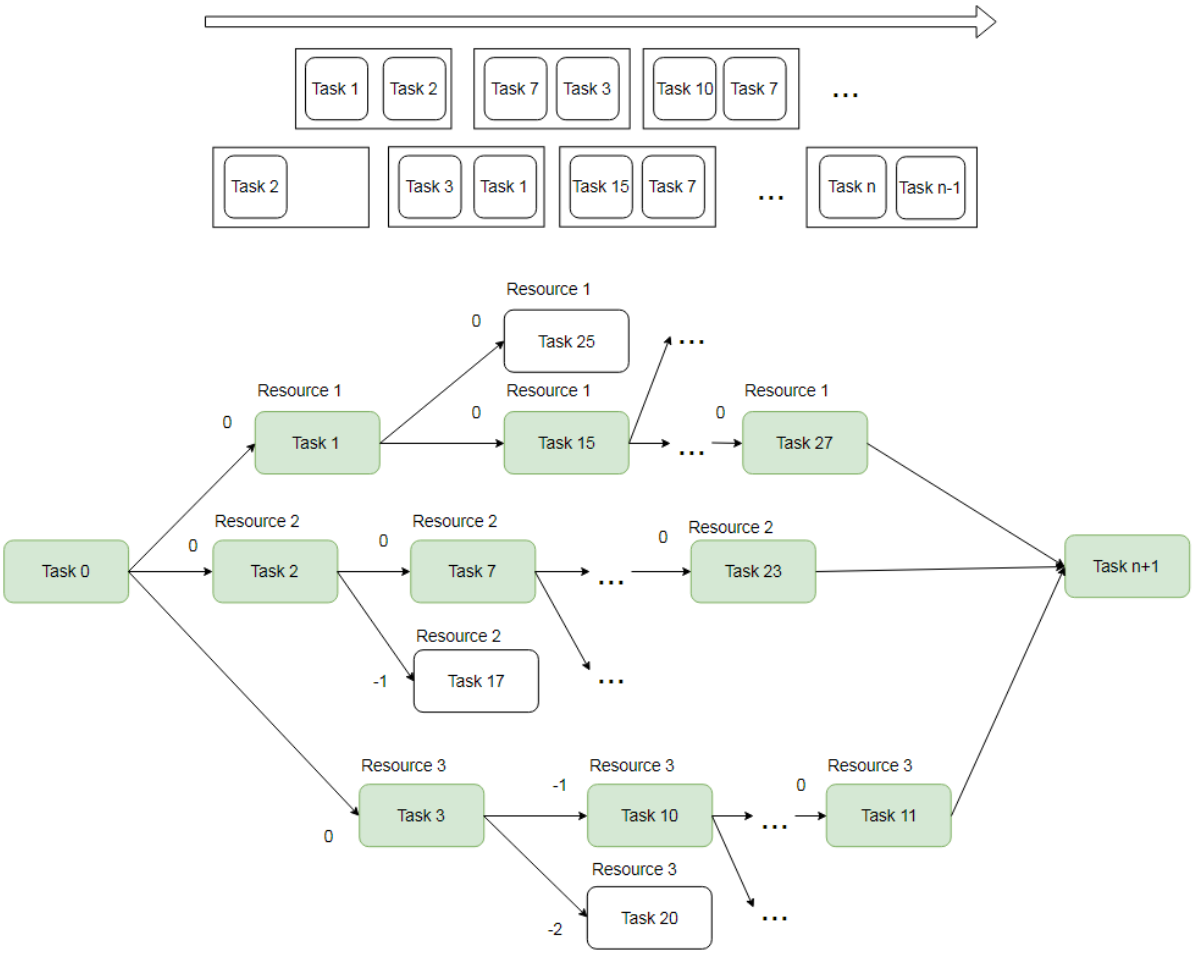


Рис. 1.9. Схема поиска с запретами

1.7.3. Поиск восхождением к вершине

Данный алгоритм является итерационным методом, который пошагово пытается получить улучшенное решение, изменяя один из элементов решения.

Шаги алгоритма [9] (Рис. 1.10):

1) Выбор начального расписания s_j;

2) Выбор расписания $s_{j+1}=s_j \cup \{\{r_{j+1},\{e_{i,j+1}, e_{i+1,j+1}, \dots, e_{i+l,j+1}\}\}\}$ из множества $\{s_{j+1}, s_{j+2}, \dots, s_{j+n} | s_j \in s_{j+l}, l=\overline{1,n}\}$, где $g(s_{j+1})=\max$. Если имеется несколько расписаний с одинаковой оценкой, то выбирается одно случайное;

3) Если $n > c_{step}$, где c_{step} – количество шагов, при которых $g(s_n)$ остается неизменной, то алгоритм окончен, иначе переход на шаг 2.

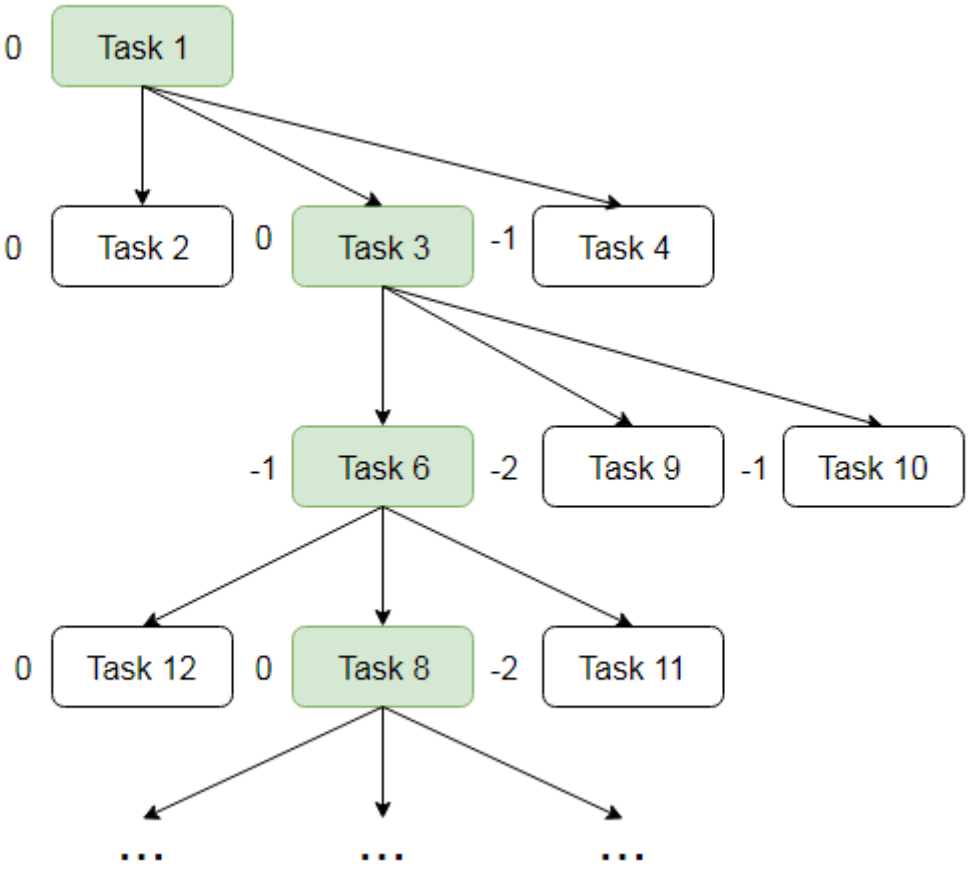


Рис. 1.10. Схема поиска восхождением к вершине

1.8. Интегрированные решения Halliburton

Компания Halliburton предлагает два решения в области бурения. (Рис. 1.11):

- Пакет настольных приложений EDT;
- Единое решение уровня предприятия с веб доступом.

EDT содержит программные продукты, охватывающие все задачи связанные с информационной поддержкой процесса бурения.

Витрина данных является платформой, которая содержит веб-интерфейсы, предоставляющие доступ к сервисам для работы с базами данных EDM, OpenWorks, PDM и другими.

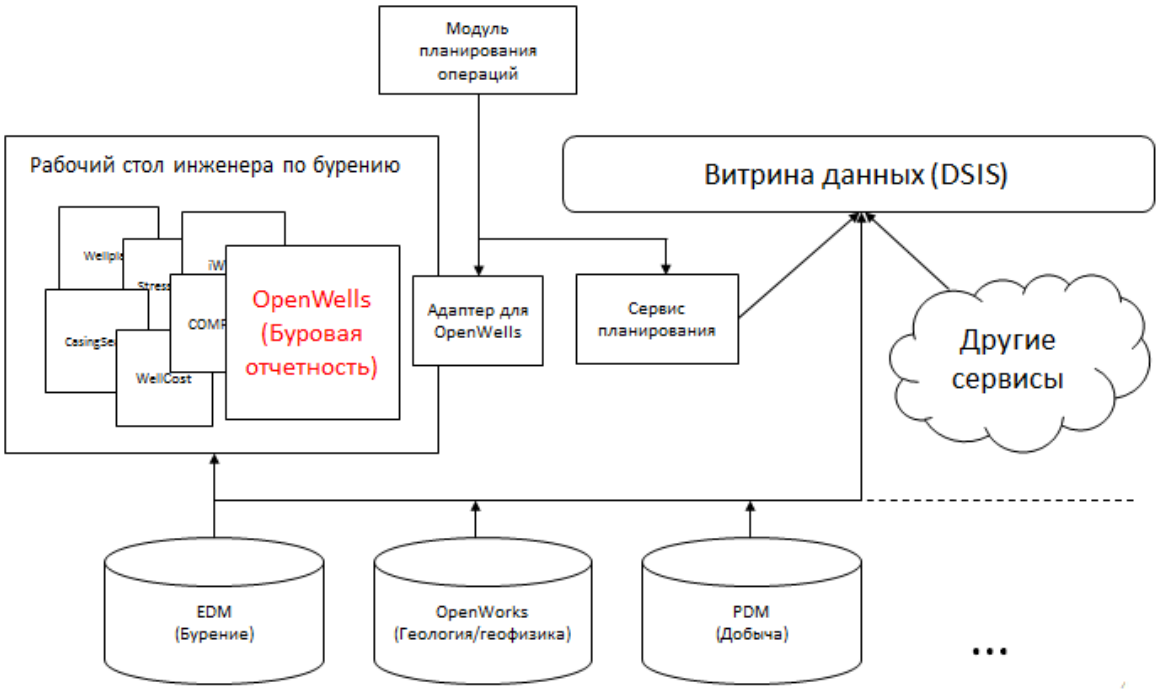


Рис. 1.11. Интегрированные решения Halliburton

Внедрения новых модулей в решения Halliburton можно сделать двумя способами. Первое – создание адаптера для продукта по буровой отчетности OpenWells. Второе – разработка и добавление сервиса планирования в витрину данных. Выбор способа внедрения напрямую зависит от требований и нужд компании.

1.9. Обзор технологий

Разработка приложение будет осуществляться на языке Java, с использованием Spring Framework. При разработке архитектуры приложения использовался язык моделирования ArchiMate. Для составления и отображения расписания используется функционал библиотеки OptaPlanner. Для расчета оценки качества расписаний используется язык управления правилами Drools.

1.9.1. ArchiMate

ArchiMate — это графический язык, содержащий набор понятий для описания архитектуры предприятия и фреймворк, представляющий логическую структуру для классификации этой информации. Основное предназначение –

высокоуровневое моделирования и анализ различных областей предприятия и взаимосвязей между ними (Рис. 1.12) [8].

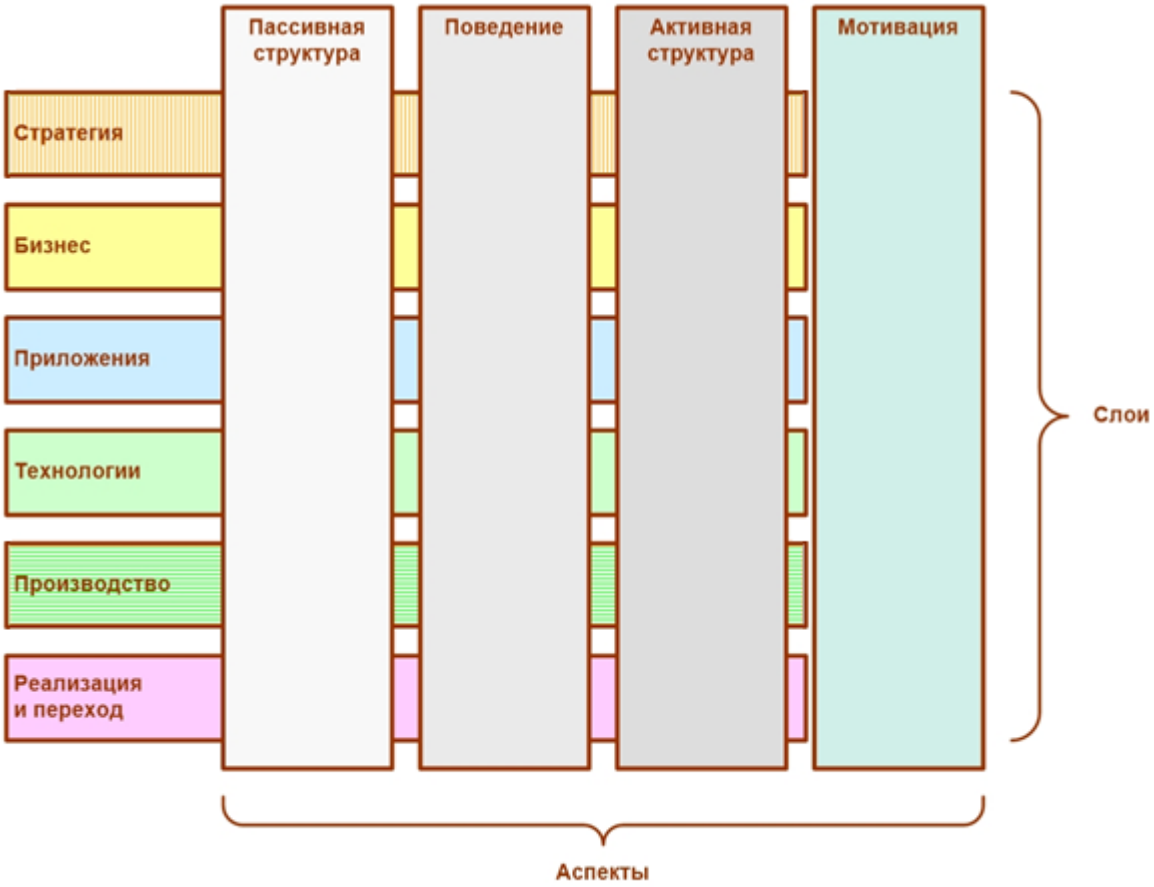


Рис. 1.12. Слои и аспекты ArchiMate

Архитектура решений в нотации ArchiMate чаще всего описывается в виде взаимодействия трех слоев – бизнеса, приложений и инфраструктуры. Такое представление позволяет учесть весь набор проблем от описания бизнес-требований и процессов до особенностей IT ландшафта, в котором будет эксплуатироваться целевая система.

1.9.2. Spring Framework

Spring Framework представляет собой набор готовых решений для использования всех основных Enterprise Java технологий — JDBC, ORM, JTA, Servlets/JSP, JMX и многих других. Основная цель Spring – предоставить разработчику готовые компоненты для решения типичных задач, чтобы большее

внимание сосредоточить на бизнес-логике приложения. Тем не менее, за счет гибкой (хоть и достаточно сложной) реализации, этот фреймворк позволяет расширить или переопределить поведение почти любого компонента, что позволяет адаптировать приложение к постоянно изменяющимся условиям бизнеса.

В Spring контекст приложения задаётся декларативно – т.е. с помощью внешнего XML-файла, а не внутри приложения. Этот подход обеспечивает очень большую гибкость приложений, создаваемых на основе фреймворка Spring [7].

Компоненты Spring:

1) IoC (Inversion of Control) контейнер;
2) AOP-фреймворк (включая интеграцию с AspectJ);
3) Data Access фреймворк;
4) Transaction management;
5) MVC-фреймворк;
6) Remote Access фреймворк;
7) Batch processing;
8) Фреймворк аутентификации и авторизации;
9) Remote Management;
10) Messaging-фреймворк;
11) Testing-фреймворк.

На схеме (Рис. 1.13) представлена связь выше перечисленных компонентов. Основными компонентами ядра, присутствующими в приложениях на Spring, являются IoC, AOP и Component/Service abstraction.

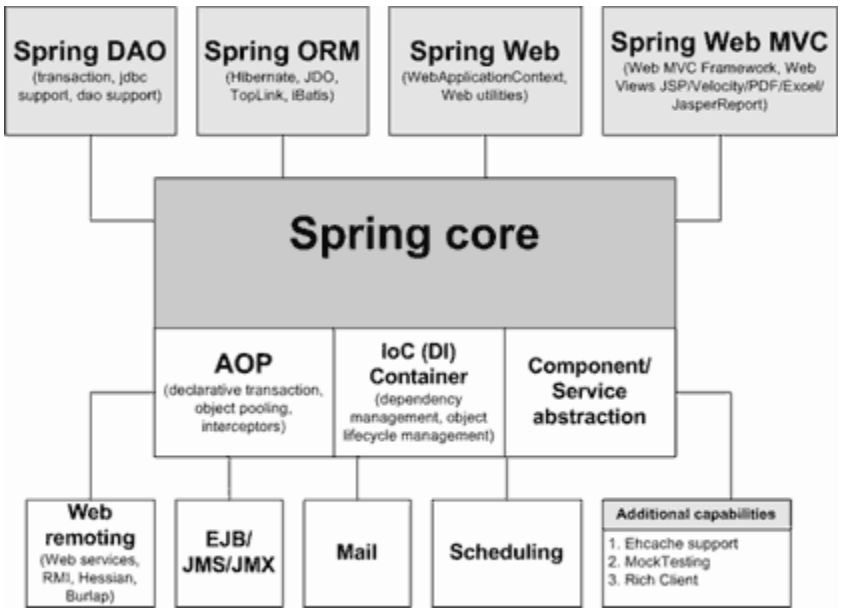


Рис. 1.13. Схема Spring Framework

Inversion of Control (инверсия управления) – паттерн проектирования, описывающий ситуацию, когда один объект реализует свой функционал через другой объект. Преимущество IoC в том, что данная модель позволяет отделить объекты от используемой ими реализации.

Aspect Oriented Programming (аспектно-ориентированное программирование) – парадигма программирования, основанная на разделении функциональности программы для лучшего разбиения на модули. В основе аспектно-ориентированного программирования лежит понятие crosscutting concerns (сквозная функциональность). Под сквозной функциональностью понимается функциональность, реализовать которую в отдельном компоненте языка программирования традиционными средствами процедурного или объектно-ориентированного программирования или очень сложно, или вообще невозможно, поскольку эта функциональность необходима в большей части модулей системы.

1.9.3. OptaPlanner

OptaPlanner – это библиотека в открытом доступе, предназначенная для решения задач оптимизации с ограниченными ресурсами и наложенными на задачу ограничениями. К таким задачам относится оптимизация целей

(минимизация затрат, максимизация эффективности работы), составление расписания работ. Данная библиотека обладает большим количеством примеров задач, для которых разработаны решения, а также отображение на экране.

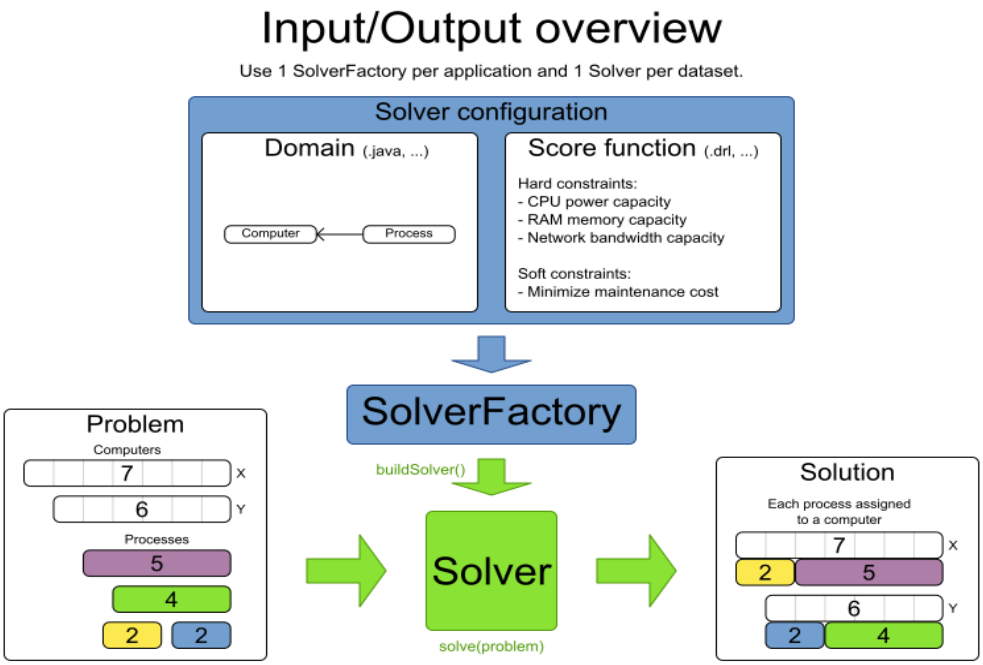


Рис. 1.14. Схема архитектуры решений в OptaPlanner

Концептуальна библиотека устроена следующим образом (Рис. 1.14):

1) Проблема (Problem) – набор исходных данных, для которых требуется найти решение. Проблема представляется специфический контрактом, который подходит только для решений, написанных под данный контракт.
2) Решение (Solution) – набор выходных данных, которые получены после решения проблемы.
3) Устройство, принимающее решения (УПР) (Solver) – модуль, отвечающий за получение решения для поставленной проблемы.
4) Фабрика УПР (SolverFactory) – содержит в всевозможные УПР, разработанные в библиотеке.
5) Конфигурация УПР (Solver configuration) – содержит описание структуры объектов проблемы (Domain), а также сценарий, в котором будут оцениваться ограничения для расчета функции оценки (Score function).

В библиотеке было заложено, что фабрика УПР распространяется на все приложение, при этом каждый набор данных соответствует одному УПР [9].

УПР содержит внутри несколько этапов получения решения(Рис. 1.15). Каждый этап (Phase) является полной или частичной реализацией алгоритма. К примеру, имеется УПР, которое сначала составляет допустимое расписание, а затем данное расписание оптимизируется. Получается, что на первом этапе происходит составление первичного расписания, которое на следующем этапе будет оптимизировано указанным алгоритмом.

Каждый этап состоит из шагов (Step). Шаг является итерацией алгоритма. К примеру, пусть у алгоритма имеется условие – пока оценка решения ниже 10. Алгоритм будет выполнять действия, пока условие не будет выполнено (оценка станет равной 10). Действие (Move) является простейшим элементом этапа, который выполняет функции изменения элементов, подпадающих под ограничения, в процессе постройки решения.

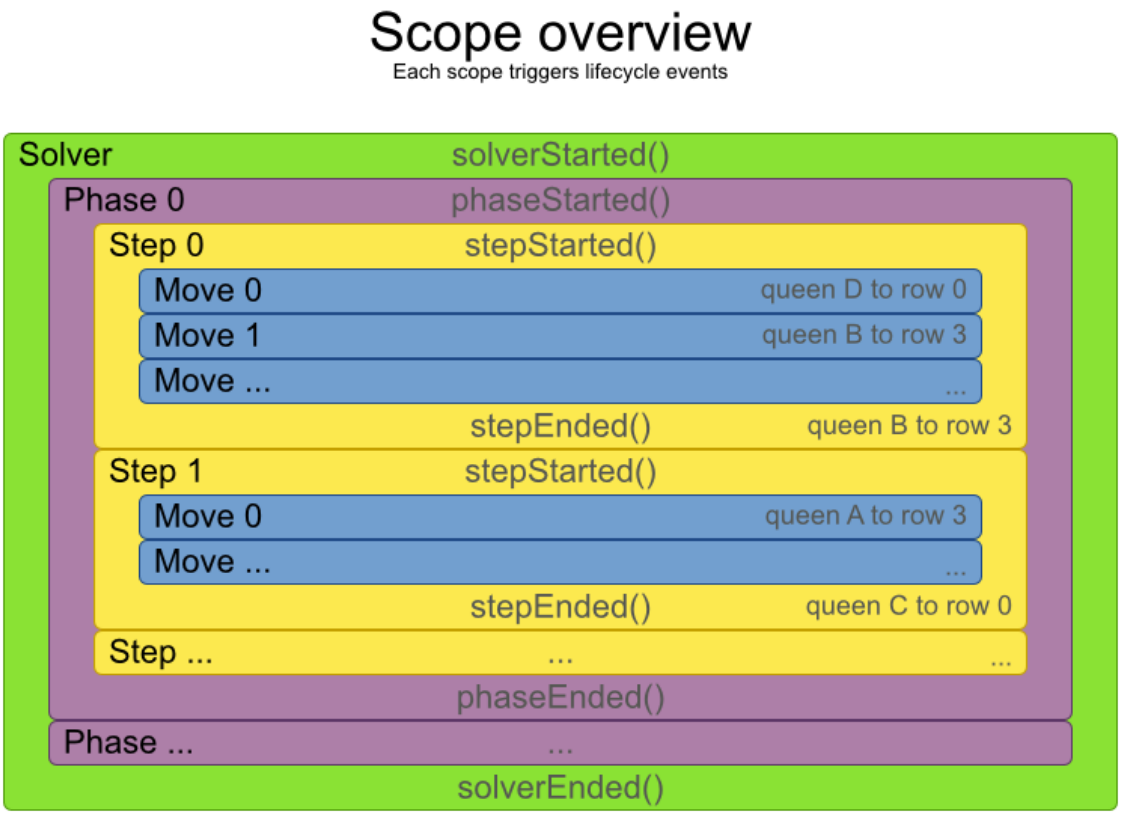


Рис. 1.15. Схема Solver

1.9.4. Drools

Drools является решением для управления бизнес правилами. Данная библиотека предоставляет функционал, позволяющий задавать правила для работы приложения через конфигурационный файл расширения *.drl. Drools позволяет описывать бизнес-правила приложения на простом и понятном языке без XML. Помимо этого Drools поддерживает вставку фрагментов кода на Java [10].

Одно из преимуществ Drools это обновление правил, без перекомпиляции приложения. Это позволяет экономить время, особенно в крупных проектах, где изменение могут привести к большим временным издержкам.

Правила имеют следующую структуру [11] (Рис. 1.16):

```
rule "Name"
    when
        conditions
    then
        actions
end
```

Рис. 1.16. Пример правила

Для правила описывается имя, затем после when задаются conditions (условия), при выполнении которых будут выполняться actions.

Схема ядра движка правил выглядит следующим образом (Рис. 1.17):

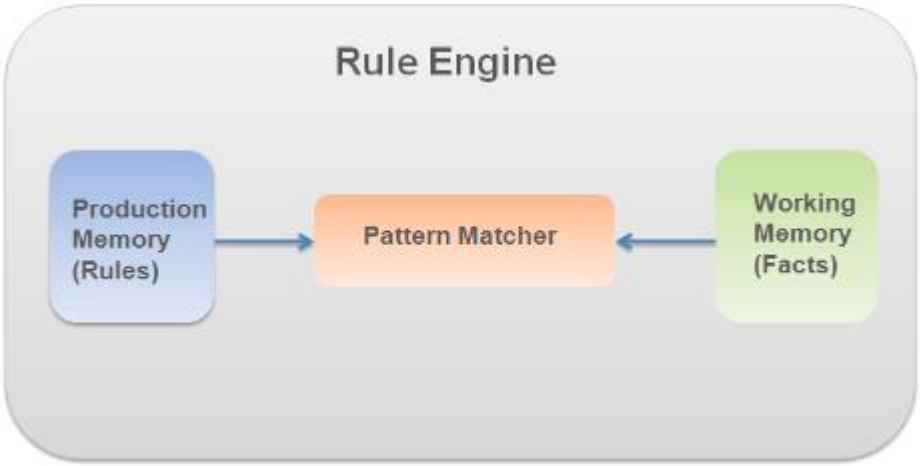


Рис. 1.17. Схема ядра движка

Правила, которые должны применить содержатся в производственной памяти (production memory). Движок правил (Rule Engine) проверяет, соответствуют ли действия правил производственной памяти тем действиям, что утверждаются в рабочей памяти. Если соответствуют, то данные правила выполняются (Рис. 1.18).

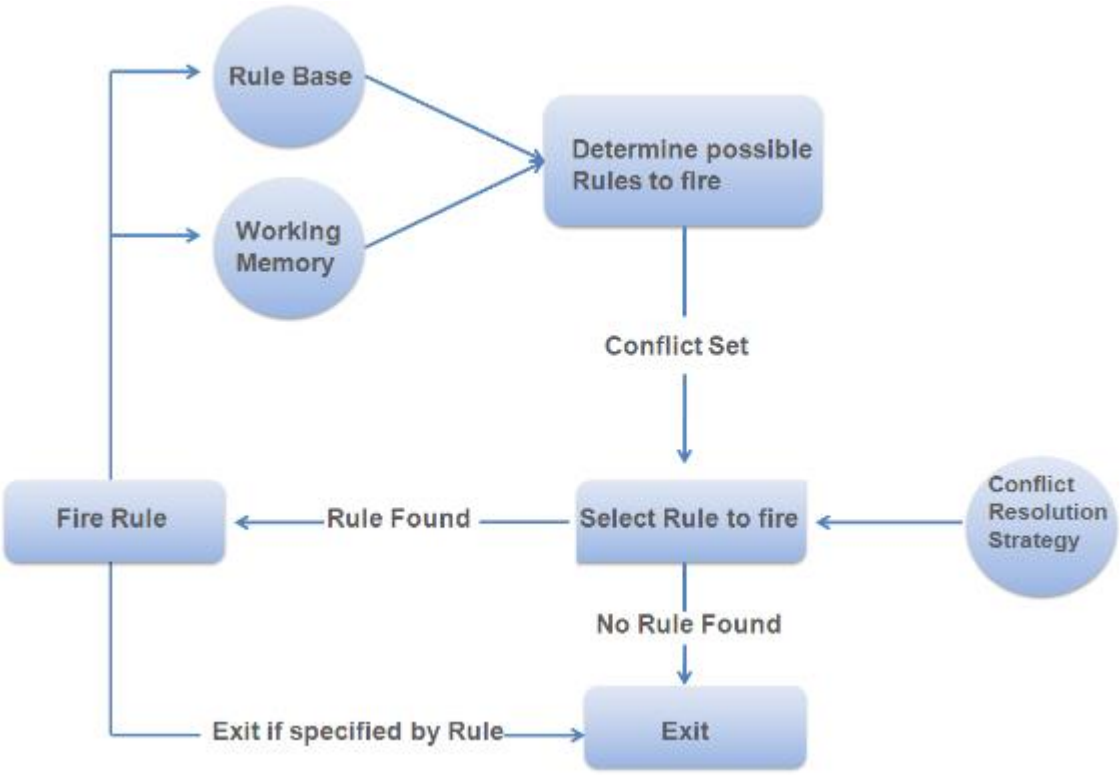


Рис. 1.18. Схема потока движка

```
user : UserBean()
eval ( user.getName() == James )
```

Рис. 1.19. Conditions

В переменную user идет объявление объекта класса UserBean. Затем при помощи java функции eval идет проверка, что имя данной переменной совпадает с указанным (Рис. 1.19). Объекты для каждого правила доступны только в рабочей памяти, а методы actions могут применяться только к объектам рабочей памяти.

Глава 2. Проектирование системы

2.1. Архитектура системы

Разработка архитектуры системы является важным этапом при разработке приложения. К сожалению, многие разработчики пренебрегают данным этапом, считая, что тщательное проработка функционирования приложения еще до его создания отнимает много времени и сил. Однако время, которое “было сэкономлено” на практике чаще всего приводит к еще большим временным затратам, нежели это отняло бы проектирование.

Для разработки данной системы была построена модель, описывающая три слоя системы (Приложение 1).

- Бизнес слой;
- Слой приложения;
- Инфраструктурный слой.

Бизнес слой описывает логику приложения со стороны пользователя. Имеется пользователь с ролью планировщик, который может выполнять два процесса: планирование и оптимизация плана. Данные процессы планировщик осуществляет при помощи пользовательского интерфейса (окно планирования), который имеет набор услуг: редактирование, планирование и оптимизация. Каждая услуга выполняется при помощи одной или нескольких функций, которые возвращают уже результат. Например, услуга работа с данными предоставляется посредствам трех функций: получение данных, просмотр данных и изменение данных. Получение данных, предоставляет исходные данные, состоящие из информации о бригадах, буровых установках и месторождении. Просмотр данных, предоставляет возможность увидеть эти данные. Изменение данных, позволяет получить изменённые пользователем данные.

Следующим следует слой приложения. Данные слой описывает логику взаимодействия приложения. К примеру, планировщик начинает процесс планирования. Для этого пользователь использует окно планирования, в котором

уже получены исходные данные. Затем планировщик обращается к услуге планирования. Планирование осуществляется при помощи функции создания плана, которая реализует свои возможности, используя интерфейс планирования, расположенный в слое приложения. Таким образом, происходит связь бизнес уровня с уровнем приложения. На данном уровне описана логика предоставления функций, используемых на бизнес уровне.

Последним из перечисленных слоев остается технологический слой. Данный слой описывает системное программное обеспечение, различные физические устройства. Например, это может быть сервер баз данных. При запуске приложения, происходит запрос на доступ к базе данных. После получения доступа, планировщик при обращении к окну планирования, через услугу работы с данными (в слое бизнеса), получит данные, которые будут предоставлены через интерфейс планирования (слой приложения) в функции отображения данных. Затем уже в функции отображения данных будет отправлен запрос к сервису базы данных, который обработает полученный запрос, запросит данные с сервера и вернет приложению, а приложение отобразит эти данные планировщику.

Таким образом, разработка архитектуры системы, позволяет увидеть систему в целом, на всех уровнях функционирования, что позволит сократить количество проблем и недочетов, при разработке.

2.2. Календарный план

1) Изучение предметной области (06.11.17 – 20.11.17);
2) Исследование алгоритмов планирования и оптимизации (20.11.17 – 07.06.18);
3) Выбор и настройка среды разработки (09.01.18 - 19.01.18);
4) Изучение принципов работы библиотеки OptaPlanners и Drools (19.03.18 – 26.06.18);
5) Разработка прототипа модуля планирования и коррекции (19.03.18 – 03.05.18);
6) Тестирование и доработка модуля (04.05.18 - 24.06.18);
7) Сравнение алгоритмов на соответствие функциональным и нефункциональным требованиям (25.05.18 - 26.06.18);
8) Отчет о проделанной работе (25.05.18 - 26.06.18).

2.3. Описание модуля

2.3.1. Описание моделей

Библиотека OplaPlanner использует такие понятия, как решение и проблема.

Проблема – это модель, описывающую структуру входных данных, на основе которых будет получено решение. Текущая модель проблемы выглядит следующим образом (Рис. 2.1):

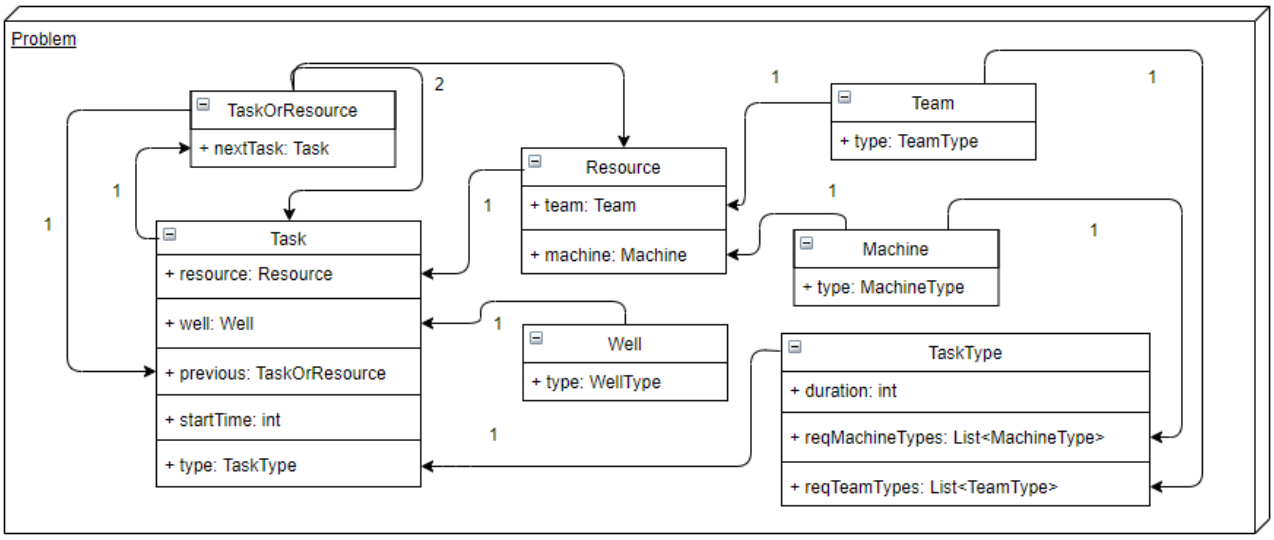


Рис. 2.1. Модель проблемы (1 – зависимость от реализующих классов, 2 – зависимость наследования)

Решение – модель, описывающая выходные данные, т.е. полученное необработанное расписание. Под необработанным расписанием понимается структура, которую требуется привести к виду, используемому в отчетности (Рис.2.2).

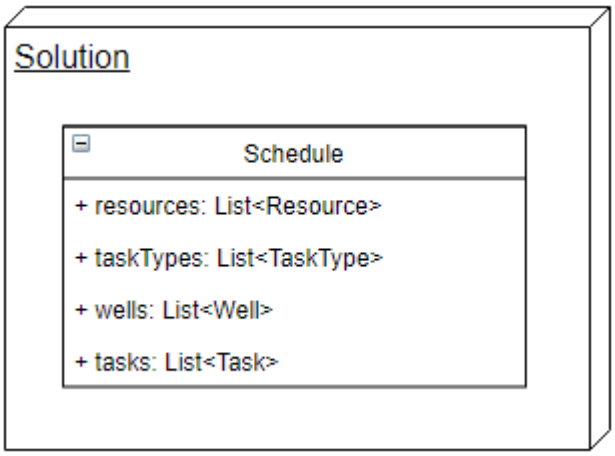


Рис. 2.2. Модель решения

Решение содержит список ресурсов доступных для работ, список типов работ, список скважин, на которых требуется выполнить работы, и список работ. Также каждый тип работ содержит необходимые типы бригад и станков для данных работ.

В библиотеке OptaPlanners используются собственные аннотации, позволяющие связать проблему и решение с тем, как будет происходить поиск решения.

При помощи аннотаций в классе Schedule отмечаются переменные, относящиеся к проблеме, либо к решению. Так resources, wells и taskTypes отмечены аннотацией @ProblemFactCollectionProperty. Все объекты отмеченные данной аннотацией, могут использоваться для расчета оценки качества решения в правилах Drools. Переменная tasks обозначается аннотацией @PlanningEntityCollectionProperty. После этого tasks становятся объектами, которые будут изменяться при формировании решения.

Класс Task основной класс данной модели, представляет работу, которую требуется выполнить. Работа содержит информацию о скважине, на которой будет выполнена работа и ресурсе, который будет использоваться. Ресурс является классом объединяющим команду и станок. Класс TaskOrResource является родительским классом для Task, который позволяет осуществить связку конкретного ресурса с работой, которая будет выполняться.

После того, как основные аннотации были распределены, требуется задать логику внутри модели. В первую очередь указывается объект, для которого будет происходить подбор элементов. В данном случае это ресурс. Обозначается следующей аннотацией (Рис. 2.3):

```
@AnchorShadowVariable(sourceVariableName = "previous")
private Resource resource;
```

Рис. 2.3. Аннотация переменной resource

Тем самым отмечается, что именно ресурс будет в качестве «якоря» – первого элемента графа решения, для которого уже будут подбираться работы. Связка осуществляется через переменную previous. Переменная previous содержит следующую аннотацию (Рис. 2.4):

```
@PlanningVariable(valueRangeProviderRefs = {"resourceRange", "taskRange"},
        graphType = PlanningVariableGraphType.CHAINED)
private TaskOrResource previous;
```

Рис. 2.4. Аннотация переменной previous

Аннотация @PlanningVariable отмечает как переменную, на которую можно ссылаться shadow variable (теневые переменные). В valueRangeProviderRefs указываются переменные, хранящие значения для переменной. В данном случае это список ресурсов и работ. Чтобы данная аннотация стала активной, требуется в классе Schedule для переменной resources и tasks указать соответственные аннотации: @ValueRangeProvider(id = "resourceRange") и @ValueRangeProvider(id = "taskRange"). После этого связка будет осуществлена. Также при помощи graphType отмечается, что привязка выполняется один к одному.

Далее для переменной nextTask объявляется следующая аннотация (Рис. 2.5):

```
@InverseRelationShadowVariable(sourceVariableName = "previous")
protected Task nextTask;
```

Рис. 2.5. Аннотация переменной nextTask

Данная аннотация связывает переменную nextTask с переменной previous. Для обновления теневой переменной требуется добавить аннотацию (Рис. 2.6):

```
@CustomShadowVariable(variableListenerClass = ScheduleStartTimeUpdatingVariableListener.class,
                      sources = {@PlanningVariableReference(variableName = "previous")})
private Integer startTime;
```

Рис. 2.6. Аннотация переменной startTime

Переменная variableListenerClass указывает на класс, в котором прописана логика обновления переменной previous (Приложение 2).

Таким образом после выше перечисленных действий, модель проблемы и решения связываются, и становятся распознаваемыми для УПР, которое может обрабатывать исходные данные и формировать на основе этих данных решение.

2.3.2. Описание конфигурации

В данном модуле содержится два конфигурационных файла:

- Конфигурационный файл УПР;
- Файл с правилами оценки качества решений.

Конфигурационный файл УПР содержит описание методов, доступных для поиска решения для данного модуля, классов, которым соответствует проблема и решение.

Вот пример конфигурационного файла (Рис. 2.7):

```
<?xml version="1.0" encoding="UTF-8"?>
<solver>
  <!-- <environmentMode>FULL_ASSERT</environmentMode> -->
  <solutionClass>com.examples.schedulegenerate.domain.Schedule</solutionClass>
  <entityClass>com.examples.schedulegenerate.domain.Task</entityClass>
  <entityClass>com.examples.schedulegenerate.domain.TaskOrResource</entityClass>

  <scoreDirectorFactory>
    <scoreDrl>com/examples/schedulegenerate/scheduleScoreRules.drl</scoreDrl>
  </scoreDirectorFactory>

  <constructionHeuristic>
      <constructionHeuristicType>ALLOCATE_ENTITY_FROM_QUEUE</constructionHeuristicType>
  </constructionHeuristic>

  <localSearch>
    <localSearchType>TABU_SEARCH</localSearchType>
    <termination>
        <unimprovedStepCountLimit>30</unimprovedStepCountLimit>
    </termination>
  </localSearch>
</solver>
```

Рис. 2.7. Конфигурационный файл

1) solver – текущая конфигурация УПР;

2) solutionClass – текущий класс решения;

3) entityClass – текущие класс (ы) проблемы;

4) scoreDirectorFactory – список файлов правил;

5) scoreDrl – файл с правилами;

6) constructionHeuristic – тег, в котором настраивается функция отвечающая за генерацию расписания;

7) constructionHeuristicType – конкретный тип функции;

8) localSearch – функции локального поиска (используются для оптимизации планирования);

9) localSearchType – конкретная оптимизационная функция;

10) termination – настройки прекращения работы УПР;

11) unimprovedStepCountLimit – ограничение по количеству шагов, при отсутствие изменений в оценке лучшего решения, для работы УПР.

Файл с правилами оценки содержит правила, по которым выстраивается оценка решений, при работе алгоритмов. В данном файле содержатся допустимые и оптимальные ограничения. В OptaPlanner такие ограничения называются hard и soft. Hard ограничения не могут быть нарушены и соблюдаются, начиная с генерации расписания. Soft ограничения могут нарушаться, но это приводит к снижению оценки. Благодаря данным оценкам и достигается качество построенных решений.

Файл начинается с подключения необходимых классов (Рис. 2.8):

```
package com.examples.schedulegenerate.solver;
    dialect "java"

import org.optaplanner.core.api.score.buildin.bendable.BendableScoreHolder;

import com.examples.schedulegenerate.domain.Task;
import com.examples.schedulegenerate.domain.TaskOrResource;
import com.examples.schedulegenerate.domain.Well;
import com.examples.schedulegenerate.domain.Schedule;
import com.examples.schedulegenerate.domain.Resource;

global BendableScoreHolder scoreHolder;
```

Рис. 2.8. Подключение зависимых классов

BendableScoreHolder предоставляет доступ к объекту через, который будет осуществляться изменение оценки решений. Остальные классы используются для получения доступа к объектам, на основе которых будут строится условия для правил. Далее следует блок hard правил (Рис. 2.9):

```
rule "Resource compare priority team"
    when
        Task($id : id, resource != null, $team : resource.getTeam(), $startTime : startTime)
        Task(id != $id, resource != null, resource.getTeam() == $team, startTime == $startTime)
    then
        scoreHolder.addHardConstraintMatch(kcontext, 0, -1);
end

rule "Resource compare priority machine"
    when
        Task($id : id, resource != null, $machine : resource.getMachine(), $startTime : startTime)
        Task(id != $id, resource != null, resource.getMachine() == $machine, startTime == $startTime)
    then
        scoreHolder.addHardConstraintMatch(kcontext, 0, -1);
end

rule "Resource compare required machine types with resource machine type"
    when
        Task($id: id, resource != null, resource.getMachine() != null, $machineType : resource.getMachine().getType())
        Task(id == $id,  type.getRequiredMachinesType() != null, type.getRequiredMachinesType().size() > 0, !type.getRequiredMachinesType().contains($machineType))
    then
        scoreHolder.addHardConstraintMatch(kcontext, 0, -1);
end

rule "Resource compare required team types with resource team type"
    when
        Task($id: id, resource != null, resource.getTeam() != null, $teamType : resource.getTeam().getType())
        Task(id == $id, type.getRequiredTeamsType() != null, type.getRequiredTeamsType().size() > 0, !type.getRequiredTeamsType().contains($teamType))
    then
        scoreHolder.addHardConstraintMatch(kcontext, 0, -1);
end
```

Рис. 2.9. Hard правила

Данные правила проверяют, чтобы ресурсы станок и бригада не использовались одновременно на разных работах. Эти правила относятся именно к hard, т.к. нарушение этих правил ломает логику бизнес-процесса (одна бригада или станок не может одновременно быть на двух и более разных объектах)

Следующими следуют правила soft блока (Рис. 2.10):

```
rule "Minimize time"
    when
        Task($id : id, well != null, resource != null, nextTask != null, $endTime : endTime)
    then
        scoreHolder.addSoftConstraintMatch(kcontext, 1, - ($endTime * $endTime));
end
```

Рис. 2.10. Soft правила

Данное правило используется для минимизации времени выполнения работ. Правило может быть нарушено, например, когда все работы выполняются последовательно. Поэтому данное правило относится к soft, т.к. нарушение правило не разрушает логику процесса.

Правило минимизации времени окончания работ использует следующую формулу в расчетах:

$$(-1)*(t_j + d_j)^2$$

Данная формула реализует квадратичную функцию справедливости. Функция справедливости позволяет сравнивать работы и выявлять «несправедливую оценку». Под несправедливой оценкой понимаются значения функции, которое при разных комбинациях работ может давать одни и те же значения. Например работы с оценками -3 и -7 будут совпадать по оценке с работами -4 и -6. При справедливой оценке данное совпадения будет невозможным.

Регуляция выполнения правил осуществляется при помощи методов addHardConstraintMatch и addSoftConstraintMatch, находящиеся в actions блоке, которые, при выполнении условий, снижают оценку.

В правилах, указанных выше, все значения оценок являются отрицательными. В логику библиотеки OptaPlanner заложено разделение значений оценок на максимизирующие и минимизирующие. Положительное значение оценки относится к максимизирующей. Отрицательное значение к минимизирующей. Минимизирующее значение подразумевает, что выполнение

правила, должно стремиться к минимуму. Максимизирующее значение, наоборот, должно стремиться к максимуму [9].

2.3.3. Описание интерфейса

Тестовая версия интерфейса выглядит, следующим образом (Рис. 2.11):

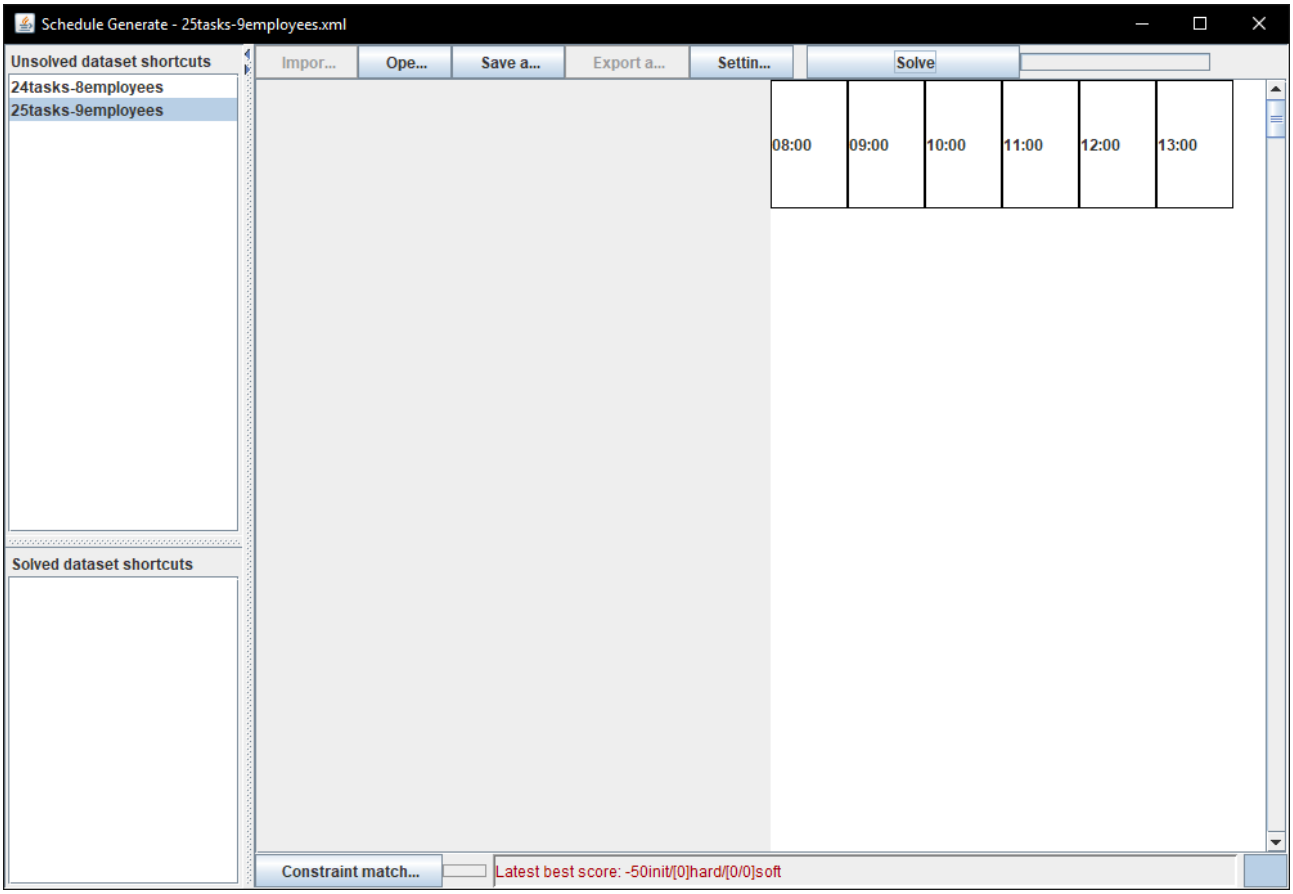


Рис. 2.11. Начальное окно

Интерфейс поддерживает загрузку исходных данных проблемы из xml файла. Исходные данные состоят из списка ресурсов, списка скважин, списка типов работ и списка работ. Список работ считается неинициализированным, т.е. для каждой работы класса Task поля предыдущей работы или ресурса previous и следующей работы nextTask равны null.

При нажатии на кнопку Solve начинается формирование и оптимизация расписания (Рис. 2.12):

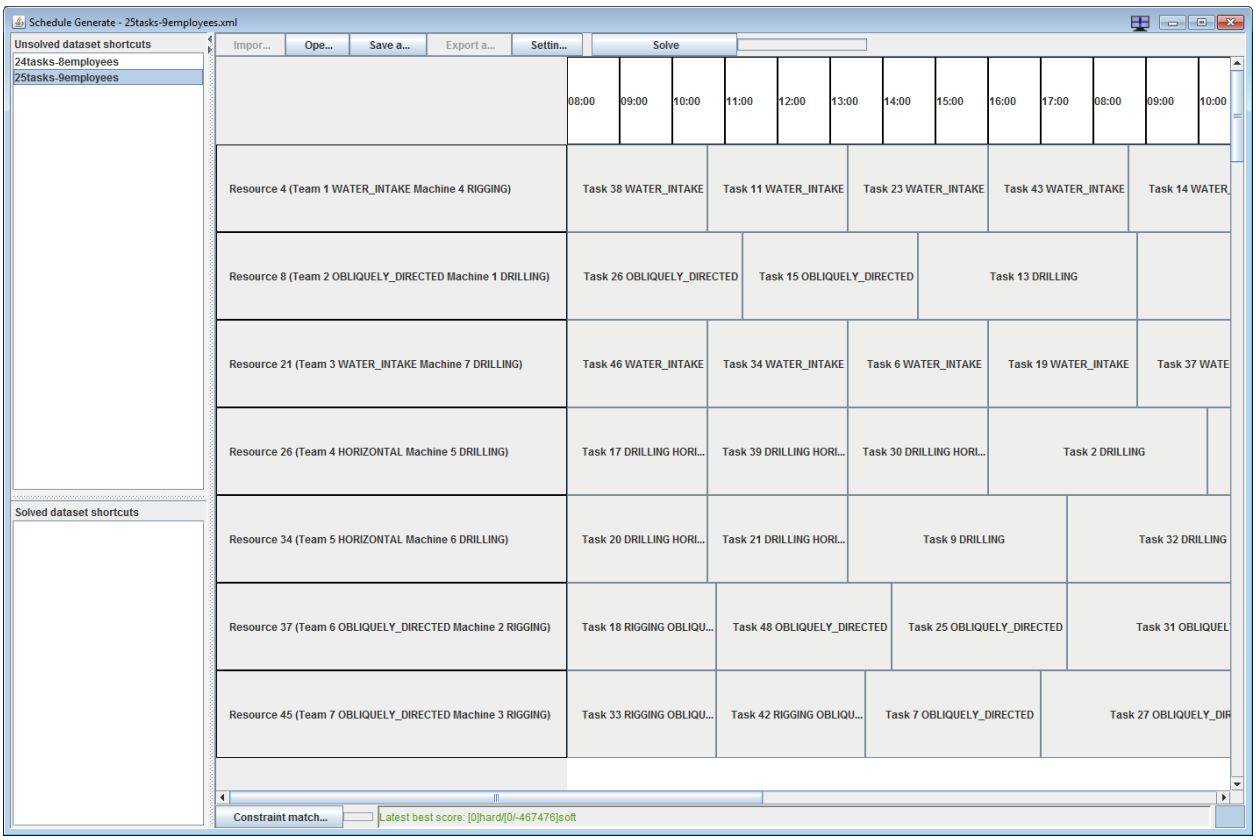


Рис. 2.12. Формирование расписания

Расписание представляет из себя список ресурсов, между которыми распределены работы. Resource содержит внутри комбинацию станков и бригад. Task содержат требуемые типы бригад и станков. Благодаря установленным hard правилам, при генерации расписания работы распределяются лишь на такие ресурсы, у которых не повторяются станки и бригады и совпадают требуемые типы.

После того как расписание сформировано, его можно выгрузить в виде xml файла и просмотреть содержимое. Eclipse позволяет просматривать xml файла в режиме Design, что позволяет в удобном виде наблюдать структуру файла (Рис. 2.13).

SGenToSchedule	
id	1
id	0
timeStart	0
durationDays	350
resources	
id	2
SGenToResource	
id	3
id	0
nextTask	
fullName	Resource 1
machine	
id	23
id	0
fullName	Machine 1
type	DRILLING
locked	false
team	
id	24
id	0
fullName	Team 1
type	HORIZONTAL
SGenToResource	

Рис. 2.13. Пример содержимого xml файла

При желании, во время выполнения, или остановив работу алгоритма кнопкой Solve, имеется возможность изменить местоположение работы (Рис. 2.14).

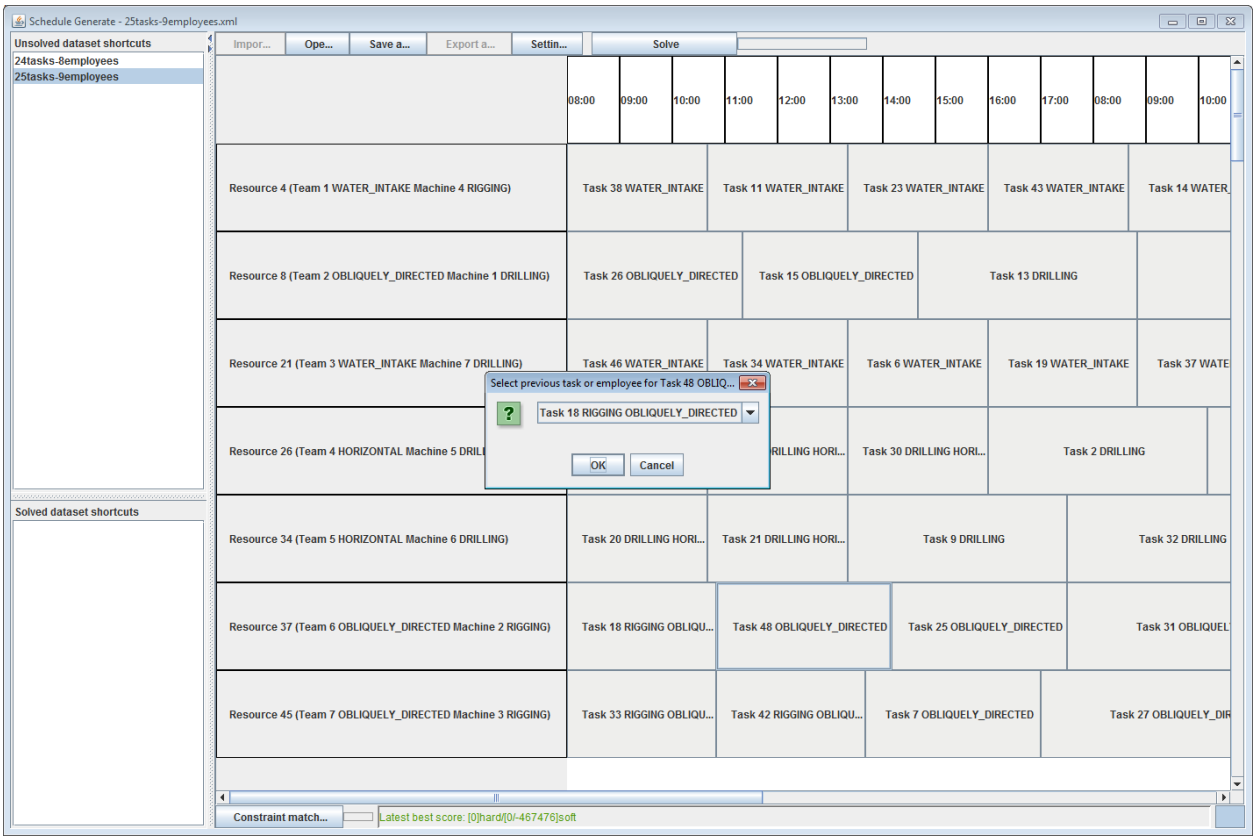


Рис. 2.14. Изменение позиции Task 48. Окно редактирования

Так, например, если взять Task 48, у которого предшествующий Task 18 и изменить предшествующий на Task 25, то Task 48 будет перемещен за Task 25 (Рис. 2.15).

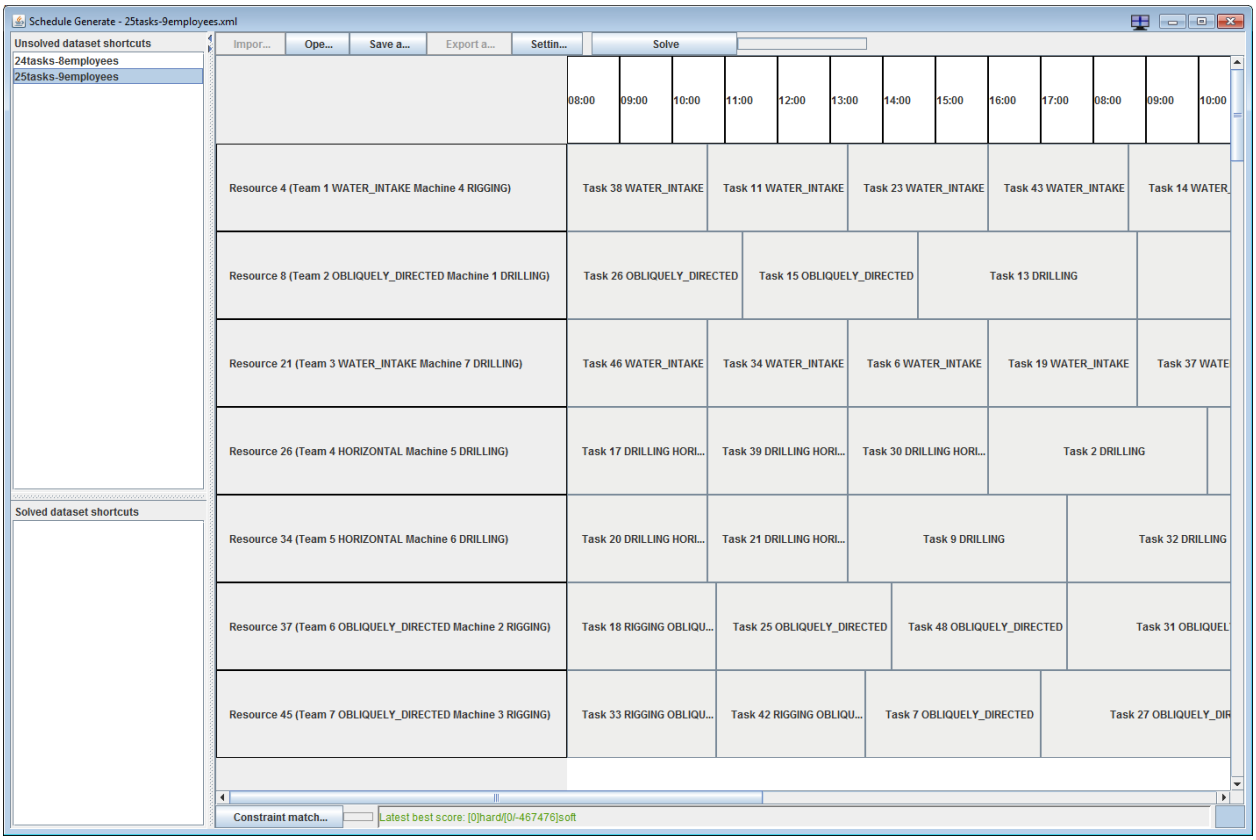


Рис. 2.15. Изменение позиции Task 48

2.4. Сравнение алгоритмов

Описание оборудования для тестирования:

- Процессор: Intel(R) Core(TM) i7 CPU M620 @2.67Ghz.
- Кол-во ядер: 2.
- RAM: 8.00 GB.
- Тип системы: 64-х разрядная.

В качестве модели использовалась модель описанная выше (Пункт 2.3.1. Описание моделей).

Исходные данные являются xml файлом, который содержит:

- 63 ресурса, в которых содержатся различные комбинации 7 бригад и станков;
- 10 скважин;
- 50 работ из 10 разных типов работ.

Для тестирования было произведено 100 замеров работы каждого алгоритма. Работа алгоритмов продолжалась до тех пор, пока, в течении 30 шагов, не прекращалось изменение оценки. Результаты планирования сравнивались с эталонным решением, которое имеет наилучшую оценку решения среди остальных (Рис. 2.16). Время решения 80.794 секунды, а оценка -467476. Полная версия решения содержится в приложениях (Приложение 4).

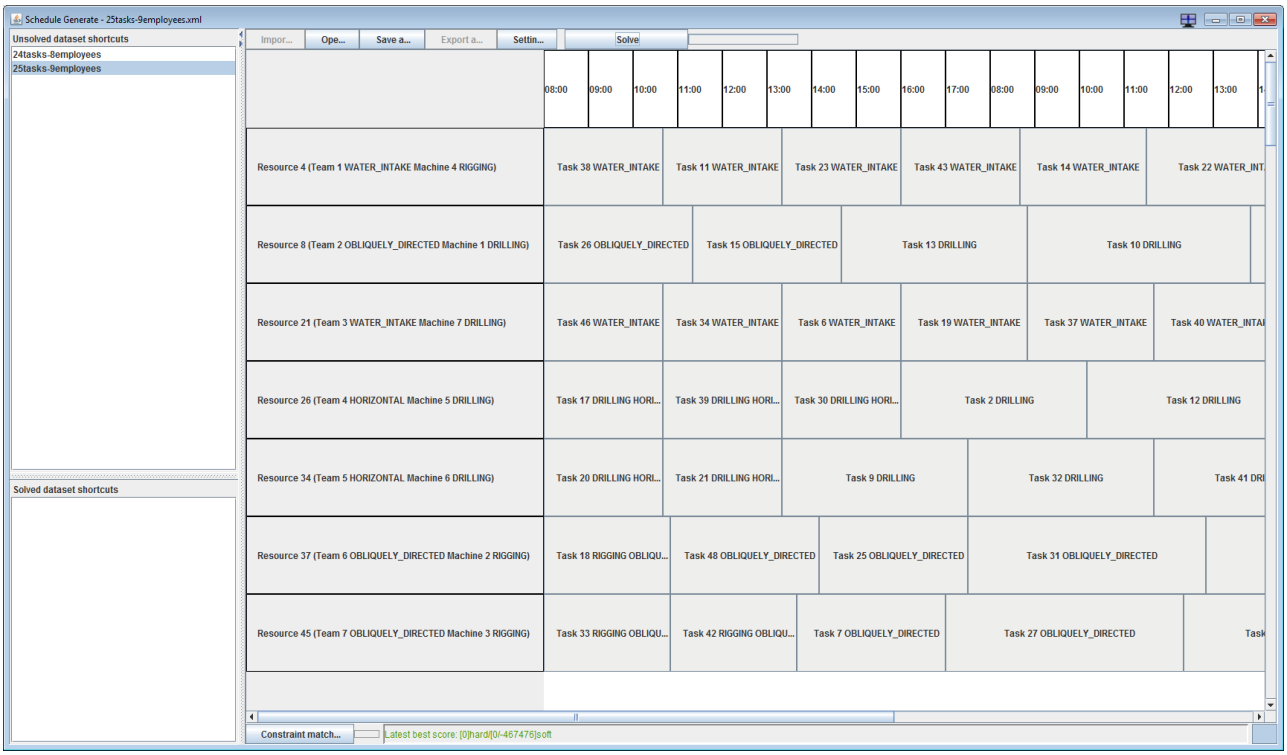


Рис. 2.16. Эталонное решение

Результаты тестирование следующие (Приложение 3):

- Лучшие результаты среди алгоритмов генерации расписания у алгоритма размещения объекта из очереди.
- Среди алгоритмов оптимизации лучшие результаты оказались у алгоритма имитации отжига и поиск восхождением к вершине.

Заключение

В данной работе были выполнены следующие задачи:

1) Изучена предметная область;
2) Исследованы технологии и алгоритмы для разработки модуля;
3) Разработан прототип модуля планирования;
4) Проведено сравнение алгоритмов с учетом указанных требований;
5) Выбраны алгоритмы, которые однозначно войдут в конечную сборку модуля.

Алгоритм размещения объекта из очереди демонстрирует лучшие результаты среди алгоритмов генерации по соотношению времени к качеству. Метод ветвей и границ совпадает по оценке качества решения, однако уступает по времени работы больше, чем в 3 раза.

Среди алгоритмов оптимизации стоит отметить алгоритма имитации отжига и поиск восхождением к вершине. Алгоритм имитации отжига показывает лучше качество, чем у поиска восхождением к вершине, однако уступает последнему по времени. В результате оба алгоритма войдут в конечную сборку. Поиск с запретами показал лучшую оценку, однако время выполнения оказалось больше, чем в 10 раз.

В дальнейшем планируется завершение конечной сборки модуля, а именно усовершенствование модели планирования, расширение функционала модуля и интеграция в приложение OpenWells.

Текущая модель планирования требует таких дополнений, как добавление кустов для скважин. Добавление возможности изменения списка типов скважин, бригад, станков и работ через конфигурации.

Модуль требует доработки интерфейса, понятного для пользователя. Разработку методов для представления расписания в виде план-графика разбуривания. Разделение функционала генерации и оптимизации расписания.

Реализацию адаптера для получения и корректировки исходных данных из баз данных.

Список литературы

1. Лазарев А. А., Гафаров Е. Р. Теория расписаний // Задачи и алгоритмы. -М.: Физический факультет МГУ. – 2011. – 222 с.

2. Аверченков В. Эволюционное моделирование и его применение: монография // В. Аверченков, П. Казаков. –2-е изд., стереотип. -М.: Флинта, 2011. – 200 с.

3. Лопатин, А. Метод отжига // Стохастическая оптимизация в информатике. – 2005. – Т. 1. – С. 133-149.

4. Савин, А.Н. Применение алгоритма оптимизации методом имитации отжига на системах параллельных и распределённых вычислений // Известия Саратовского университета. Новая серия. Серия Математика. Механика. Информатика. – 2012. – Т. 12. – №. 1. – С. 110-116.

5. Ладошкин А.И., Майорова И.А., Харитонова Е.А. Методы и модели оптимизации графика буровых работ // Вестник СамГУ. –2011. –№6, С. 172-180.

6. Fahmy A. M. Optimization Algorithms in Project Scheduling [Электронный ресурс] // Optimization Algorithms-Methods and Applications. – InTech, 2016 –URL: https://www.intechopen.com/books/optimization-algorithms-methods-and-applications/optimization-algorithms-in-project-scheduling (дата обращения: 25.06.2018).

7. Знакомимся со Spring Framework [Электронный ресурс] // KV.by High-Tech Club – 2009. – №. 29. –URL: https://www.kv.by/node/30203 (дата обращения: 25.06.2018)

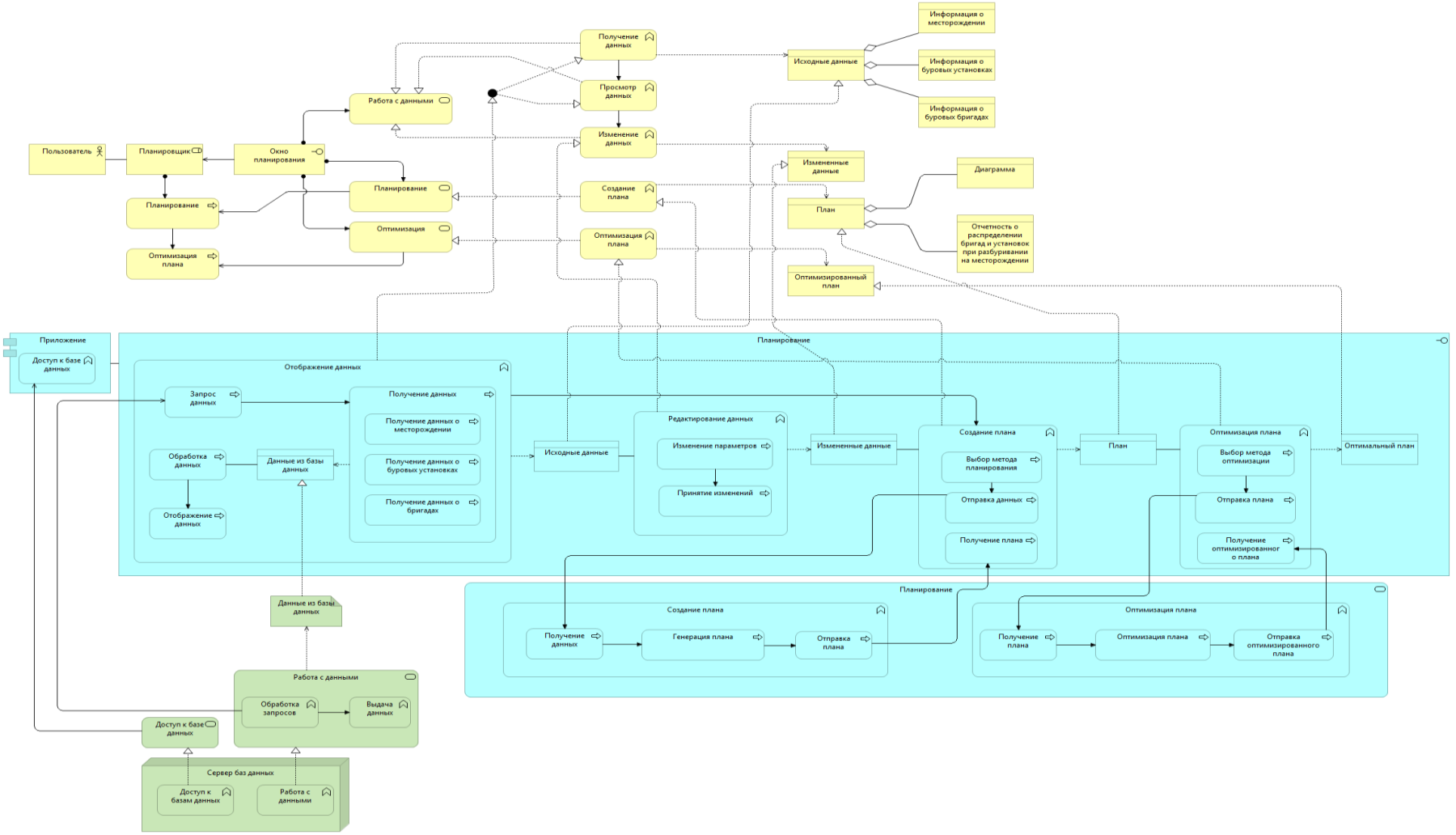
8. Josey A, Lankhorst M, Band I, Jonkers H, Quartel D. An Introduction to the ArchiMate® 3.0 Specification. [Электронный ресурс] // White Paper from The Open Group. –2016. –URL: https://www.itmg-int.com/itmg-int-wp-content/Archimate/An%20Introduction%20to%20Archimate%203.0.pdf (дата обращения: 25.06.2018)

9. OptaPlanner User Guide [Электроный ресурс]. –URL: https://docs.optaplanner.org/7.7.0.Final/optaplanner-docs/html_single/index.html (дата обращения: 25.06.2018)

10. Реализация бизнес-логики при помощи процессора правил Drools [Электроный ресурс] // Декларативный подход к программированию бизнес-логики приложений. Рикардо Оливьери. –URL: https://www.ibm.com/developerworks/ru/library/j-drools/index.html (дата обращения: 23.05.2018)
11. Introduction to rules engine [Электроный ресурс] // IBM Security Identity and Governance, Version 5.1.0. –URL: https://www.ibm.com/support/knowledgecenter/en/SSGHJR_5.1.0/com.ibm.isig.doc_5.1.0/CrossIdeas_Topics/RULES_ENGINE/RUD_DroolsRulesEngine.htm (дата обращения: 23.05.2018)

Приложение 1

Схема ArchiMate



Приложение 2

Код класса ScheduleStartTimeUpdatingVariableListener. Обновление теневой переменной

```
protected void updateStartTime(ScoreDirector scoreDirector, Task sourceTask) {
    TaskOrResource previous = sourceTask.getPreviousTaskOrResource();
    Task shadowTask = sourceTask;
    Integer previousEndTime = (previous == null ? null : previous.getEndTime());
    Integer startTime = calculateStartTime(shadowTask, previousEndTime);
    while (shadowTask != null && !Objects.equals(shadowTask.getStartTime(), startTime)) {
        scoreDirector.beforeVariableChanged(shadowTask, "startTime");
        shadowTask.setStartTime(startTime);
        scoreDirector.afterVariableChanged(shadowTask, "startTime");
        previousEndTime = shadowTask.getEndTime();
        shadowTask = shadowTask.getNextTask();
        startTime = calculateStartTime(shadowTask, previousEndTime);
    }
}

private Integer calculateStartTime(Task task, Integer previousEndTime) {
    if (task == null || previousEndTime == null) {
        return null;
    }
    return Math.max(task.getReadyTime(), previousEndTime);
}
```

Приложение 3

Таблица замеров работы алгоритмов

Алгоритм	Количество замеров	Среднее время (мс)	95-й процентиль для времени	Оценка	Отклонение оценки (%)
Первый подходящий	100	422	601	-478098	2,22
Размещение объекта из очереди	100	412	532	-471734	0,9
Самая дешевая вставка	100	5097	6039	-474765	1,54
Метод ветвей и границ	100	1413	1753	-471734	0,9
Поиск с запретами	100	8756	9827	-467476	0
Поиск восхождением к вершине	100	618	839	-467940	0,1
Алгоритм имитации отжига	100	701	923	-467702	0,05

Приложение 4

Эталонное решение