Алгоритмы оптимизации, основанные на методе проб и ошибок

1. Задачи условной оптимизации

1.1. Математическая формулировка задач условной оптимизации и их классификация

Задача условной оптимизации заключается в нахождении компонентов вектора $X = (x_1,\ldots,x_n)$ (оптимизируемых параметров) минимизирующих целевую функцию $f(X)$ при выполнении ограничений $g_i(X)$ и $X \in S$:

$$\begin{aligned} &\min f(X) \\ &g_i(X) \le 0,\ i=1,\ldots,m \qquad (1.1.) \\ &X \in S \end{aligned}$$

Если $X \notin S$, то как минимум одна из функций f, g_i в этом случае будет не определена на этом значении X.

По свойствам функций f, g_i и определению множества S можно ввести классификацию различных задач оптимизации [М. Мину. Математическое программирование. Теория и алгоритмы.- М.: Наука, 1990.]. Например, если функции f, g_i – линейные и $S \subset Z^n$, то задача (1.1) относится к классу задач целочисленного линейного программирования.

В отдельный класс выделим задачи комбинаторной оптимизации. Они возникают, например, при проектировании информационно-управляющих систем реального времени. В постановке любой задачи комбинаторной оптимизации присутствуют некоторые объекты, которые требуется определенным образом разместить, упорядочить или разбить на группы. В этом случае множество S состоит из решений X, описывающих всевозможные размещения/упорядочивания/разбиения на группы исходно заданных объектов. Математически множество S описывается набором ограничений, которые требуют согласованного изменения значений элементов X: изменение значения одного из элементов для сохранения корректности решения может требовать принятия конкретных значений ряда других элементов. Допустимые значения этих переменных в отличии от задач целочисленной оптимизации единственны.

Поясним сказанное выше для задачи построения многопроцессорных расписаний с минимизацией числа используемых процессоров. Задано множество взаимодействующих работ подлежащих планированию. На множестве работ определено отношение частичного порядка, которое задает ограничения на последовательность выполнения работ. Отношение частичного порядка является ацикличным и транзитивным. Для каждой работы задано время ее выполнения. Расписание определено, если для каждой работы

задан процессор, на котором она выполняется и порядковый номер выполнения работы на процессоре (работа не может начать выполняться, пока не выполнены все работы с меньшими номерами). Требуется построить расписание выполнения работ на минимально необходимом числе процессоров время выполнения которого не превосходит исходно заданный директивный срок. Для этой задачи значением функции f является число процессоров, функция g задает ограничение: время выполнения расписания не должно превышать директивный срок. Множество S определяется следующими ограничениями:

1. Каждая работа должна быть назначена на процессор для выполнения.
2. Каждая работа должна быть назначена лишь на один процессор.
3. Частичный порядок, заданный на множестве работ должен быть сохранен в расписании.
4. Расписание должно быть беступиковым. Условием беступиковости расписания является ацикличность отношения частичного порядка в расписании.

Если эти условия не выполняются, то мы не можем вычислить время выполнения расписания.

Если мы изменяем расписание путем перемещения работы с процессора A на процессор B, то номера работ на процессоре A больше перемещаемой должны уменьшиться на 1, а номера работ на процессоре B, которые должны выполняться после перемещаемой работы должны увеличиться на 1.

1.2. Понятие «сложных» задач условной оптимизации. Проблемы применения оптимальных и эвристических алгоритмов, использующих априорно известные свойства о целевой функции и функциях ограничений

Под сложными задачами условной оптимизации будем понимать задачи (1.1) для которых отсутствует априорная информация о функциях f,g_i и множестве S, которая может быть использована для организации поиска оптимального решения, или сложность получения этой информации неприемлема. Можно выделить следующие основные особенности сложных задач условной оптимизации:

1. Функций f,g_i могут быть операторами, заданными правилами/алгоритмами их вычисления, т.е. их аналитическая структура не может быть использована для организации поиска оптимального решения задачи (1.1).

2. Негладкий характер функций f,g_i.

3. Отсутствие информации о производных функций f,g_i или их производные не являются непрерывными.

4. Множество S может быть компонентно разнородным: $S = S^R \cup S^Z \cup S^F \cup S^{KS}$, т.е. различные компоненты вектора X могут принадлежать подмножествам множества действительных чисел (R), целых чисел (Z), функций (F) и комбинаторных структур (KS).

Указанные выше особенности делают проблематичным применение оптимальных и эвристических методов, использующих некоторые априорно известные свойства задачи для организации поиска оптимального решения. При больших размерности и/или размерах задачи (1.1) методы полного перебора допустимых решений также оказываются неприемлемыми по сложности.

Идея о целесообразности случайного поведения, при наличии неопределенности, т.е. отсутствии достаточной информации, которая может быть использована для организации поиска оптимального решения/поведения, впервые в четкой форме была сформулирована У.Р. Эшби в работе [У. Росс Эшби. Конструкция мозга.- М.:, ИЛ, 1962.] и реализована в известном гомеостате (гомеостат Эшби). Применительно к сложным задачам условной оптимизации, алгоритм поиска оптимального решения должен опираться на метод проб и ошибок. Только такой процесс позволяет извлечь информацию, необходимую для организации поиска решения. Метод проб и ошибок основан на понятии случайного эксперимента: случайного выбора значений компонентов вектора X, что дает возможность получить информацию о функциях f,g_i и множестве S, которая может быть использована для организации поиска решения сложной задачи условной оптимизации.

Наиболее широко применяемыми подходами к построению алгоритмов оптимизации опирающихся на метод проб и ошибок являются следующие:

- алгоритмы случайного поиска (ненаправленного, направленного, направленного с самообучением) [Л.А. Растригин. Статистические методы поиска.- М.: Наука, 1968.],
- алгоритмы имитации отжига [Уоссермен Ф. Нейрокомпьютерная техника. Теория и практика.- М.: Мир, 1992. – 240с.],
- генетические и эволюционные алгоритмы [Holland J.N. Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan: Univ. of Michigan Press, 1975.],
- муравьиные алгоритмы [Dorigo M. Optimization, Learning and Natural Algorithms. // PhD Thesis. Dipartimento di Elettronica, Politechnico Di Milano, Milano. 1992.], [Штовба С.Д. Муравьиные алгоритмы: теория и применение// Программирование. 2005. №4.].

1.3. Классические задачи комбинаторной оптимизации

1.3.1. Задача коммивояжера

Задачу коммивояжёра можно представить в виде нахождения пути на графе. Вершины графа соответствуют городам, а ребра между вершинами являются путями сообщения между этими городами. Граф является полносвязным, т.е. между любыми двумя вершинами есть ребро. Каждому ребру можно сопоставить вес, который можно понимать, например, как расстояние между городами, время или стоимость поездки. Маршрутом (гамильтоновым циклом) называется маршрут на таком графе, в который входит по одному разу каждая вершина графа. Задача заключается в отыскании кратчайшего маршрута. Длина маршрута определяется как сумма весов входящих в него ребер. Маршрут может быть описан циклической перестановкой номеров городов $J = (j_1, j_2, \dots j_n, j_{n+1})$, где j_i - номер города находящийся в i –ой позиции перестановки. Пространством поиска решений является множество перестановок n городов, таких, что все $j_1, j_2, \dots j_n$ разные и $j_1 = j_{n+1}$.

Существует несколько частных случаев общей постановки задачи, в частности:

- симметричная задача коммивояжера,
- асимметричная задача коммивояжера,
- метрическая задача коммивояжера.

В симметричной задаче коммивояжера задается неориентированный граф и все пары ребер между одними и теми же вершинами имеют одинаковый вес, т.е. для любого ребра *(i,j)* $w_{ij} = w_{ji}$.

В асимметричной задаче коммивояжера задается ориентированный граф и дуги между одними вершинами могут иметь разный вес, т.е. $w_{ij} \neq w_{ji}$.

Симметричная задача коммивояжера является метрической, если для весов ребер выполняется правило треугольника: $w_{ij} \leq w_{ik} + w_{kj}$. То есть прямой путь между любыми двумя вершинами не длиннее любого обходного пути.

В практических задачах, если между некоторыми вершинами не существует ребер, то они искусственно добавляются в граф и им присваиваются большие веса. Из-за большого веса такое ребро никогда не попадает в оптимальный маршрут и маршруты, близкие по длине к оптимальному маршруту.

Данная задача принадлежит к классу *NP*-трудных задач и часто используется как тестовая задача для сравнения качества работы различных алгоритмов опирающихся на метод проб и ошибок.

Математические формулировки задачи могут быть разными. Выбор математической формулировки определяется классом алгоритма, который предполагается использовать для решения задачи.

1.3.2. Задача о рюкзаке

Имеется рюкзак объемом b и набор n различных товаров. Для каждого товара i заданы: объем a_i и стоимость c_i. Требуется упаковать рюкзак так, чтобы суммарная стоимость упакованных товаров была максимальной и их суммарный объем не превышал объем рюкзака b.

Математически эту задачу можно записать следующим образом:

$$\max_X\left(\sum_{i=1}^{n} c_i \cdot x_i\right)$$

$$\sum_{i=1}^{n} a_i \cdot x_i \le b$$

$$\forall i = 1,\ldots,n : x_i = 0 \text{ или } 1$$

Если товар упакован в рюкзак, то $x_i=1$, если нет $x_i=0$.

Если все коэффициенты a_i целые числа и b целое число, то задача является *NP*-трудной. Если все коэффициенты a_i вещественные, то задача может быть решена жадным алгоритмом сложности $O(n \cdot \log n)$. Обе задачи о рюкзаке обладают свойством оптимальности для подзадач. Вынув товар i из оптимально загруженного рюкзака, получим решение задачи о рюкзаке с максимальным объемом $b-a_i$ и набором из $n-1$ товара. То есть, жадный алгоритм может быть построен как для непрерывной, так и дискретной задачи. Вычислим относительную стоимость всех товаров в расчете на единицу объема c_i/a_i. Оптимальный жадный алгоритм для непрерывной задачи заключается в следующем: сначала в рюкзак укладывается по максимуму товар с самой дорогой относительной стоимостью. Если товар закончился, а рюкзак не заполнен, то укладывается следующий по относительной стоимости товар, затем следующий, и так далее, пока не набран вес b. В [Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. М.: МЦНМО, 2000.] показано, что аналогичный жадный алгоритм может не получать оптимум в дискретной задаче о рюкзаке.

1.3.3. Задачи построения расписаний и их классификация

Опишем общую задачу построения расписания согласно [Теория расписаний и вычислительные машины. Под ред. Э.Г. Коффмана. М.: Наука, 1984. 334 с.]. Модель в рамках которой могут быть сформулированы многие задачи построения расписаний задается совокупностью моделей ресурсов, системы заданий (работ) и меры оценки расписаний.

Ресурсы. В большинстве моделей ресурсы состоят просто из набора процессоров P. В наиболее общей модели еще имеется набор дополнительных типов ресурсов R, некоторое подмножество которых используется на протяжении всего времени выполнения задания на некотором процессоре. Общее количество ресурса типа R_i задается целым положительным числом.

Система работ может быть определена через $T, \prec, [\tau_{ji}], \{R_j\}$, где:

- $T=\{T_1,T_2,\ldots,T_n\}$ - набор работ подлежащих планированию;
- $\prec$ - заданное на T отношение частичного порядка, которое определяет ограничение на последовательность выполнения работ. Если работы связаны отношением $T_i \prec T_j$,то работа T_i должна быть завершена раньше, чем начнется выполнение работы T_j.
- $[\tau_{ji}]$ – матрица времен выполнения работ на процессорах, элемент которой τ_{ji} задает время выполнения работы j на процессоре i;
- $R=\{R_1(T_j),\ldots, R_s(T_j)\}$ – набор, *i-ая* компонента которого задает количество ресурса типа R_i необходимое для выполнения задания T_j.

Расписание может задаваться одним из трех способов:

- Временная диаграмма – для каждой работы задано время начала выполнения $s'(T_j)$ и процессор на котором она выполняется.
- Привязка работ к процессорам и порядковый номер выполнения задания на процессоре. Работа не может начаться выполняться, если не выполнены все работы на процессоре с меньшими порядковыми номерами, завершены все ее предшественники, выполняющиеся на других процессорах и достаточно количества требуемых дополнительных типов ресурсов. Привязка - всюду определенная на множестве работ функция, которая задает распределение работ по процессорам. Порядок задает ограничения на последовательность выполнения работ и является отношением частичного порядка, удовлетворяющим условиям ацикличности и транзитивности. Отношение порядка на множестве, работ распределенных на один и тот же процессор, является отношением полного порядка.

- Привязка работ к процессорам и приоритет работы. Работа может начать выполняться, если завершены все ее предшественники (работа готова к выполнению), в очереди готовых работ нет работ с большим приоритетом, на процессоре не выполняется никакая другая работа и достаточно количества требуемых дополнительных типов ресурсов. Частным случаем являются списочные расписания: существует единый упорядоченный список работ и не задается привязка работ к процессорам.

Задачи построения расписаний можно разделить на классы в соответствии со следующими характеристиками:

1. Тип процессоров.
2. Тип отношения частичного порядка на множестве работ.
3. Времена выполнения работ.
4. Способ задания директивных интервалов работ.
5. Модель вычислений.
6. Мера оценки эффективности расписания.

Тип процессоров. В теории расписаний наиболее часто рассматриваются следующие случаи:

- процессоры одинаковые по производительности и по функциональным возможностям;
- процессоры разные по производительности и одинаковые по функциональным возможностям;
- процессоры разные по производительности и по функциональным возможностям.

Тип отношения частичного порядка на множестве работ. Наиболее часто рассматриваются следующие отношения порядка:

- пустое,
- лес,
- произвольное.

Времена выполнения работ могут быть:

- равными,
- различными,
- равными 1 или 2.

Директивные интервалы работ могут быть:

- f- общий директивный интервал для всей системы работ,
- $[s_j,f_j)$ – индивидуальные директивные интервалы для каждой работы,
- $[0,f_j)$ – индивидуальные директивные интервалы с общей левой границей,
- r_j – требуемая частота выполнения работы.

Модель вычислений характеризуется дисциплиной обслуживания работ и учитываемыми временными задержками при вычислении времени выполнения расписания. *Дисциплина обслуживания работ* может быть:

- без прерываний – работа не может быть прервана до полного завершения,
- с прерываниями – разрешается прерывать работу и запускать ее в последующем. При этом предполагается, что общее время требуемое для выполнения работы остается неизменным.

Временные задержки:

- учитываются временные задержки начала выполнения работ, определяемые лишь отношением частичного порядка в расписании и ограниченным числом процессоров;
- учитываются временные задержки начала выполнения работ, связанные с получением доступа к разделяемым ресурсам (каналам обмена, общей памяти….) и работой системного программного обеспечения. В этом случае получение точного значения времени выполнения расписания возможно лишь с использованием имитационных моделей.

Мера оценки эффективности расписания:

- время выполнения расписания,
- число используемых процессоров для выполнения множества работ за время не превышающее заданные директивный срок,
- максимальное число совместимых работ (для задач в которых задаются индивидуальные директивные сроки работ),
- критерии, основанные на использовании функций штрафа за нарушение директивных сроков работ (используются при построении расписаний для систем мягкого реального времени).

В таблице приведены известные из работ [1-17] функции штрафа. Функции штрафа определены для работ $i = <s'_i, s_i, f_i, t_i>$ и имеют вид $F_i = F_i(s', s, f, t)$, где

- s' – время старта работы;
- $[s, f)$ – директивный интервал работы;
- t – время выполнения работы.

В таблице:

- $f' = s' + t$ – время завершения работы;
- c_1, c_2 - константы, не зависящие от конкретной работы.

Название штрафной функции	**Математическое представление:** **F_i =**
Отставание [1,2,3,11,13,14,15]	$max\{0, f'_i - f_i\}$
Смещение [1,2,4,5,6,11,13,14]	$f'_i - f_i$
Завершение [2, 4,5,6,7,14]	f'_i
Дискретное запаздывание [1,4,15]	*1, если* $f'_i > f_i$, *иначе 0*
Отклонение [5]	$\lvert f_i' - f_i \rvert$
Опережение [5,13]	$max\{0, f_i - f_i'\}$
Непопадание [7]	*0, если* $(s_i', f_i') \subseteq (s_i, f_i)$, *иначе 1*
Простых элементов [8,9]	$U * (U + 1) / 2$, *где* $U = max(f_i' - f_i, 0) + max(s_i' - s_i, 0)$
Красильщика [11]	$f_i' + const_i$
Длительность [13]	$f_i' - s_i'$
«Жесткие сроки» [12]	$-c_1$, *если* $f_i' \le f_i$, *иначе* $c_2(f_i'/f_i - 1)$
Крепкие сроки [12]	$-c_1$, *если* $f_i' \le f_i$, *иначе* c_2
Мягкие сроки [12]	$-c_1$, *если* $f_i' \le f_i$, *иначе* $(c_2-c_1)f_i'/f_i - c_2$
Неубывающая кусочно-неперывная (общий вид) [15]	*0, если* $f_i' \le f_i$, *иначе* $E = \varphi$, *где* $\varphi > 0$, φ – *неубывающая кусочно-непрерывная функция*
Ступенчатая [15]	*0, если* $f_i' \le f_i$, *иначе* a_i, *где* $a_i > 0$, a_i – *некоторая константа для i-й работы*
Длительность прохождения (Flow time) [16, 17]	$f'_i - s_i$
Функция специального вида (1) [1]	$\varphi(f_i')$ + α_i, *где* $\varphi(f_i')$ *не убывает в полуинтрвале* $(0, f]$, $\alpha_1 \ge \alpha_2 \ge \ldots \ge \alpha_n$
Функция специального вида (2) [1]	$\alpha_i * \varphi(f_i')$, *где* $\varphi(f_i')$ *не убывает в полуинтрвале* $(0, f]$, $\alpha_1 \ge \alpha_2 \ge \ldots \ge \alpha_n$
Функция специального вида (3) [1]	$\varphi(f_i' + \alpha_i)$, *где* $\varphi(f_i')$ *не убывает в полуинтрвале* $(0, f]$, $\alpha_1 \ge \alpha_2 \ge \ldots \ge \alpha_n$
Функция специального вида (4) [1]	$a_i u_i(f_i')$, $u_i(f_i') = 0$, *если* $f_i' \le f$, *иначе* $u_i(f_i') = 1$; $a_i > 0$

Название штрафной функции	**Математическое представление:** **F_i =**
Произвольная неубывающая [1]	*Произвольная неубывающая функция*

В таблице приведены критерии оценки качества расписаний построенные на основе функций штрафа.

Название критерия	**Математическое представление**
Максимум [1,4,5,6,11,13,14,15]	**max** F_i, i = 1..n
Взвешенный максимум [2,14]	**max** w_iF_i, I = 1..n
Сумма [9,10,13,14,15,16]	$\sum_{i=1}^{n} F_i$
Взвешенная сумма [8,10,11]	$\sum_{i=1}^{n} w_i F_i$
Средняя сумма [13]	$\frac{1}{n}\sum_{i=1}^{n} F_i$
Взвешенная средняя сумма [14]	$\frac{1}{n}\sum_{i=1}^{n} w_i F_i$

1. Танаев В.С., Гордон В.С., Шафранский Я.М.. Теория расписаний. Одностадийные системы. Москва "Наука", главная редакция физико-математической литературы 1984.

2. Щепин. Е.В. Теория расписаний // Школа Яндекса по анализу данных, 2007 (http://www.mi.ras.ru/~scepin/1-sched.pdf)

3. Лазарев А.А., Гафаров Е.Ф. Теория расписаний. Минимизация суммарного запаздывания для одного прибора // Вычислительный центр им. А.А. Дородницына Российской академии наук, 2006 [PDF] (http://aspirantura.mipt.ru/zastchita/avtoreferats/fupm/f_3hygnb)

4. Кононов А.В. Задачи теории расписаний на одной машине с длительностями работ, пропорциональными произвольной функции // Дискретный анализ и исследование операций. Июль - сентябрь 1998. Серия 1. Том 5, № 3, с. 17-37.

5. Алексеева Е.В. Теория принятия решений. Лекция 10 [PDF] (http://math.nsc.ru/LBRT/k5/lec10.pdf)

6. Алексеева Е.В. Теория принятия решений. Лекция 11 [PDF] (http://math.nsc.ru/LBRT/k5/lec11.pdf)

7. Сафонова Т.Е.. О минимизации числа элементов работ, не попавших в свои директивные интервалы [PDF] (http://www.mathnet.ru/php/getFT.phtml?jrnid=znsl&paperid=3165&what=fullt).

8. Сафонова Т.Е. Реализация алгоритма построения оптимального расписания с началом в заданном интервале [PDF] (http://www.mathnet.ru/php/getFT.phtml?jrnid=znsl&paperid=3166&what=fullt)

9. Шахабзян. К.В. Упорядочение структурного множества работ, минимизирующее суммарный штраф [PDF] (http://www.mathnet.ru/php/getFT.phtml?jrnid=znsl&paperid=3169&what=fullt)

10. Шахабзян. К.В. О задачах теории расписаний типа n|1||□ci(t) [PDF] (http://www.mathnet.ru/php/getFT.phtml?jrnid=znsl&paperid=3328&what=fullt)

11. Лукьянова А.П., Лесин В.М. Теория расписаний [ZIP] (http://rain.ifmo.ru/cat/data/vis/schedule-theory/schedule-theory-2003/shedule.zip)

12. Павлова Е. Управление транзакциями в СУБД реального времени [HTML] (http://meta.math.spbu.ru/~katya/publications/programmirovanie2000)

13. Бодров В.И., Лазарева Т.Я., Мартемьянов Ю.Ф. Методы исследования операций при принятии решений. // Тамбов. Издательство ТГТУ. 2004. [PDF] (http://window.edu.ru/window_catalog/files/r37969/tstu2005-016.pdf)

14. Коффман Э.Г. Теория расписаний и вычислительные машины. Москва "Наука", главная редакция физико-математической литературы 1984.

15. Танаев В.С., Шкурба В.В. Введение в теорию расисаний. Издательство "Наука", главная редакция физико-математической литературы. Москва. 1975.

16. A.W. Krings. Survivable Systems and Networks, Lecture 24 2001 CS404/504. [PDF] (http://www2.cs.uidaho.edu/~krings/CS448/Notes.2004/2004-24-ssn.pdf)

17. Nikhil Bansal. Algorithms for Flow Time Scheduling. School of Computer Science, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, December 2003 [PDF] (http://www.research.ibm.com/people/n/nikhil/papers/thesis.pdf)

1.4 Задачи, возникающие при проектировании вычислительных систем реального времени

1.4.1. Задача определения минимально необходимого числа процессоров и построения расписания выполнения функциональных задач со временем выполнения не превышающим заданный директивный срок

Модель прикладной программы. При определении модели прикладной программы предполагаем, что выделение работ, подлежащих планированию, и параллелизм, допускаемый при выполнении программы заданы (выявлены) предварительно.

В качестве модели прикладной программы будем использовать частный случай инварианта поведения программы предложенного в работах [(Смелянский Р.Л. Модель функционирования распределенных вычислительных систем, // Вестн. Моск. Ун-та. сер 15, Вычисл. Матем. и Кибернетика. 1990, No. 3, стр. 3-21.), (Смелянский Р.Л. Об инварианте поведения программ // Вестн. МГУ, сер. 15, Вычислительная математика и Кибернетика, 1990., No. 4, С. 54-60.)]. Следуя этим работам, обозначим поведение программы *Bh(PR)* и определим *Bh(PR)* следующим образом: $Bh(PR) = < S, \{R \text{ — } (PR)\}, \{R \rightarrow (PR)\}>$, где S – множество всех возможных шагов процессов в допустимом диапазоне входных данных программы, $\{R \rightarrow (PR)\}$ - отношения,

определяющие частичный порядок на множестве шагов каждого процесса, $\{R — (PR)\}$ - отношения взаимодействия между процессами.

Шаги процесса определяются последовательностью взаимодействий с другими процессами. Назовем рабочим интервалом процесса внутренние действия процесса между двумя последовательными взаимодействиями с другими процессами. Каждый рабочий интервал процесса по существу является реализацией соответствующего шага процесса.

Для задачи синтеза архитектур будем использовать одну из историй поведения программы $H(PR) \in Bh(PR)$ (поведение программы для конкретного набора входных данных). Для $H(PR)$ отношение $\{R \rightarrow (PR)\}$ является отношением полного порядка, а множество S сужается до множества шагов, которые делают процессы для конкретного набора входных данных.

$H(PR)$ можно представить ациклическим ориентированным размеченным графом. Вершинам $P = \{p_i\}_{i=1}^{N_1} \cup \{p_i\}_{i=1}^{N_2} \cup \ldots \cup \{p_i\}_{i=1}^{N_K}$, соответствуют рабочие интервалы процессов, дугам $\prec = \{\prec_{ik} = (p_i, p_k)\}_{(i,k) \in (1 \ldots N)}$ - связи, определяющие взаимодействия между рабочими интервалами из множества P (определяются объединением отношений $\{R—(PR)\}$, $\{R \rightarrow (PR)\}$). Где $\{p_i\}_{i=1}^{N_j}$ - множество рабочих интервалов j-го процесса, N_j - число рабочих интервалов в j-ом процессе, K - число процессов в программе PR, $N = N_1 + N_2 + \ldots + N_K$ - мощность множества P. Чередование рабочих интервалов различных процессов, назначенных на один и тот же процессор, допустимо, если не нарушается частичный порядок, заданный $\prec$. Отношение $\prec_{ik}$ представляется следующим образом: если $p_i \prec_{ik} p_k$, то рабочий интервал p_i, необходимо выполнить до начала выполнения рабочего интервала p_k. На отношение $\prec$ накладываются условия ацикличности и транзитивности. Каждая вершина имеет свой уникальный номер и метки: принадлежности рабочего интервала к процессу и вычислительной сложности рабочего интервала. Вычислительная сложность рабочего интервала позволяет оценить время выполнения рабочего интервала на процессоре. Дуга определяется номерами смежных вершин и имеет метку, соответствующую объему данных обмена. Объем данных обмена для каждой связи из $\prec$ позволяет оценить затраты времени на выполнение внешнего взаимодействия.

Таким образом, модель прикладной программы определим:

1. Множеством рабочих интервалов процессов, составляющих программу PR:

$$P = \{p_i\}_{i=1}^{N_1} \cup \{p_i\}_{i=1}^{N_2} \cup \ldots \cup \{p_i\}_{i=1}^{N_K},$$

Нумерация рабочих интервалов является сквозной и удовлетворяет условиям полной топологической сортировки. Каждый рабочий интервал имеет метку принадлежности к процессу.

2. Частичным порядком на P:

   $\prec = \{\prec_{ik} = (p_i, p_k)\}_{(i,k) \in (1...N)}$;

3. Вычислительной сложностью рабочих интервалов:

   $\{w_i\}_{i=1}^{N}$;

4. Объемом данных обмена для каждой связи из $\prec$:

   $\{v_{ik}\}_{(i,k) \in (1...N)}$;.

Расписание выполнения программы определено, если заданы: 1)множества процессоров и рабочих интервалов, 2)привязка, 3)порядок. Привязка - всюду определенная на множестве рабочих интервалов функция, которая задает распределение рабочих интервалов по процессорам. Порядок задает ограничения на последовательность выполнения рабочих интервалов и является отношением частичного порядка, удовлетворяющим условиям ацикличности и транзитивности. Отношение порядка на множестве, рабочих интервалов распределенных на один и тот же процессор, является отношением полного порядка.

Модель расписания выполнения программы определим набором простых цепей и отношением частичного порядка $\prec_{HP}$ на множестве P: $HP = (\{SP_i\}_{i=(1...M)}, \prec_{HP})$.
Где $\{SP_i\}_{i=(1...M)}$ -набор простых цепей (ветвей параллельной программы). Они образуются рабочими интервалами процессов, распределенными на один и тот же процессор (M – число процессоров в ВС). Отношение частичного порядка $\prec_{HP}$ на множестве P для HP определим как объединение отношений: $\prec_{HP} = \prec_c \cup \prec_1 \cup \ldots \cup \prec_M$, $\prec_i$ - отношение полного порядка на SP_i, которое определяется порядковыми номерами рабочих интервалов в SP_i; $\prec_c$ - набор секущих ребер, которые определяются связями рабочих интервалов, распределенных на разные процессоры. Если рабочие интервалы p_i и p_j распределены на разные процессоры и в $\prec$ существует связь $\prec_{ij}$, то она определяет секущее ребро в HP. На отношение $\prec_{HP}$ накладываются условия ацикличности и транзитивности.

Модель расписания можно рассматривать как граф HP, вершины которого имеют дополнительную метку “номер списка”, а дуги определяются отношением $\prec_{HP}$ или как граф H, вершины которого доразмечены двойками: {“номер списка”, “порядковый номер вершины в соответствующем списке”}. В модели HP сохраняются нумерация вершин, дуг и их метки заданные в модели поведения программы H. В дальнейшем при рассмотрении

свойств расписаний в некоторых разделах будет использоваться ярусная форма представления расписания. Ярусной формой максимальной высоты будем называть такую ярусную форму, у которой на каждом ярусе находится не более одной вершины.

Ограничения на расписание. Расписание *HP* является корректным (сохраняет инвариант поведения программы), если выполнены следующие ограничения:

5. Каждый рабочий интервал должен быть назначен на процессор (в SP_i).
6. Каждый рабочий интервал должен быть назначен лишь на один процессор (в один SP_i).
7. Частичный порядок, заданный в *H* сохранен в *HP*:
   $\prec \subset \prec_{HP}^{T}$, $\prec_{HP}^{T}$ *– транзитивное замыкание отношения* $\prec_{HP}$.
8. Расписание *HP* должно быть беступиковым. Условием беступиковости является отсутствие контуров в графе *HP*: $\prec_{HP}$ *– ациклично*.
9. Все рабочие интервалы одного процесса должны быть назначены на один и тот же процессор (в один и тот же SP_i).

Ограничения 1-4 обеспечивают сохранение инварианта поведения программы и являются обязательными. Ограничение 5 запрещает возобновление работы процесса после прерывания на другом процессоре, т.е. определяет способ организации параллельных вычислений в ВС, и не всегда является обязательным. В дальнейшем будем говорить, что расписание корректно $HP \in HP_{1-5}^{*}$, если оно удовлетворяет ограничениям 1-5. Нижний индекс в HP_{1-5}^{*} указывает ограничения, налагаемые на расписание.

Задачу построения расписаний как задачу условной оптимизации можно сформулировать следующим образом.

Дано: H(PR)=(P, $\prec$*)*- модель программы, *T=f(HP,HW)*- функция вычисления времени выполнения расписания *HP* на архитектуре *HW* (целевая функция), T^{dir} - директивный срок выполнения программы.

Требуется построить: HP– расписание выполнения программы такое, что:

$$\min_{HP} M(HP, HW)$$
$$T = f(HP, HW) \le T^{dir}$$
$$HP \in HP^{*}_{1-5}$$

Здесь $M(HP, HW)$ - число процессоров, на котором выполняется расписание. Функция вычисления времени выполнения расписания может быть задана в аналитическом виде или в виде имитационной модели.

С материалами данного раздела можно ознакомиться в работах:

1. Смелянский Р.Л. Модель функционирования распределенных вычислительных систем// Вестн. Моск. Ун-та. сер 15, Вычисл. Матем. и Кибернетика. 1990, No. 3, С. 3-21.
2. Смелянский Р.Л. Об инварианте поведения программ// Вестн. МГУ, сер. 15, Вычислительная математика и Кибернетика, 1990., No. 4, С. 54-60.
3. Костенко В.А. Задача построения расписания при совместном проектировании аппаратных и программных средств// Программирование, 2002., №3. - С.64-80.
4. Калашников А.В., Костенко В.А. Параллельный алгоритм имитации отжига для построения многопроцессорных расписаний// Известия РАН. Теория и системы управления, 2008., N.3, С.133-142.
5. Калашников А.В., Костенко В.А. Итерационные алгоритмы построения расписаний, основанные на разбиении пространства решений на области// Вестн. Моск. ун-та. Сер. 15. Вычислительная математика и кибернетика. 2008., №. 3, С.56–60.

1.4.2. Задача построения статико-динамических расписаний

Статико-динамическое расписание представляет собой набор окон, каждое из которых характеризуется временем открытия, временем закрытия и списком работ, выполняющихся внутри окна. При этом порядок выполнения работ внутри окна определяется динамически и заранее неизвестен. Длина окна должна быть не меньше, чем суммарное время выполнения работ внутри него.

Системы реального времени накладывают дополнительные ограничения на расписание. Кроме времени выполнения, каждая работа характеризуется также директивным интервалом, в пределах которого возможно ее выполнение. При этом временной интервал окна должен лежать внутри каждого из директивных интервалов работ, выполняющихся в данном окне, чтобы гарантировать, что директивные интервалы работ не будут нарушены.

Примером систем реального времени, использующих статико-динамические расписания, являются системы, работающие по стандарту ARINC-653 [Arinc Specification 653. Airlines Electronic Engineering Committee. [PDF] (http://www.arinc.com).]. Одним из основных понятий стандарта является понятие раздела: в системе имеется определенный набор разделов, и каждая работа характеризуется номером раздела. Работы из одного раздела могут выполняться непосредственно одна за другой, без задержек; для перехода из одного раздела в другой необходимо переключение контекста, которое требует определенного времени. Все работы внутри окна должны принадлежать одному разделу, таким образом, работы внутри окна могут выполняться непосредственно друг за другом, без временных затрат на переключение контекста, которое может производиться только между закрытием одного окна и открытием следующего. Таким образом, для каждой работы задано время выполнения, номер раздела и директивный временной интервал выполнения.

Приведем формальную постановку задачи построения расписания для таких систем:

Пусть задано множество работ:

$SW=\{a_i=<s_i,f_i,t_i,p_i>|i\in[1..n]\}$, где

- $[s_i,f_i)$ – директивный интервал;
- t_i – время выполнения работы;
- p_i – номер раздела работы.

В дальнейшем будем предполагать, что для $\forall i\in[1..n]: s_i<f_i$ и $t_i\leq f_i-s_i$.

Кроме того, заданы два параметра: $\Delta 1$ и $\Delta 2$, определяющие, соответственно, временные затраты на переключение контекста между окнами и резерв свободного времени внутри каждого окна.

Требуется построить расписание, представляющее собой набор окон:

$SP=\{w_i=<S_i,F_i,SW_i>|i\in[1..m]\}$, где

- S_i – время открытия окна;
- F_i – время закрытия окна;
- $SW_i=\{a^i_j\}\subseteq SW$ – множество работ, выполняемых внутри окна.

Будем предполагать, что окна упорядочены в порядке возрастания времени открытия S_i.

При этом должны выполняться следующие ограничения:

1. $\forall(i,j)\in[1..m], i\neq j: SW_i\cap SW_j=\varnothing$ – множества работ, размещенных внутри разных окон, не пересекаются;

2. $\forall i \in [1..n], \forall j \in [1..m], a_i \in SW_j : s_i \leq S_j < F_j \leq f_i$ – временной интервал окна лежит внутри директивных интервалов работ, выполняющихся в окне;
3. $\forall (i,j) \in [1..n], \forall k \in [1..m], a_i, a_j \in SW_k : p_i = p_j$ – разделы работ, размещенных внутри одного окна, совпадают;
4. $\forall (i,j) \in [1..m], i < j : S_j \leq F_i + \Delta 1$ – окна не пересекаются и между любыми двумя соседними окнами есть промежуток не короче времени, необходимого на переключение контекста;
5. $\forall j \in [1..m]: \sum_{a_i \in SW_j} t_i + \Delta 2 \leq F_j - S_j$ – суммарное время выполнения всех работ из одного окна с учетом резерва времени не больше, чем длина окна.

Критерием оптимальности расписания является отношение количества размещенных работ к общему количеству исходно заданных работ.

С материалами данного раздела можно ознакомиться в работах:

1. Балаханов В.А., Костенко В.А. Способы сведения задачи построения статико-динамического однопроцессорного расписания для систем реального времени к задаче нахождения на графе маршрута // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007. – С.148-156.
2. Балаханов В.А., Кокарев В. Муравьиный алгоритм построения статико-динамических расписаний и исследование его эффективности// Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2008.

1.4.3. Задача построения расписания обменов по шине с централизованным управлением (на примере стандарта MIL-STD 1533B)

В данном подразделе приведем пример конкретной бортовой системы и протокола работы шины. Описание системы и протокола работы шины ограничим уровнем детализации необходимым лишь для понимания задачи построения расписаний обменов по шине с централизованным управлением.

Система состоит из единственного канала обмена (шины) и ограниченного набора оконечных устройств, подключенных к этому каналу, которые являются источниками/приемниками информации, передаваемой по шине. Одно из оконечных устройств назначается контроллером шины, который управляет обменом информации и осуществляет контроль состояния других оконечных устройств в соответствии со статическим расписанием. Эти оконечные устройства лишь выполняют адресованные им команды контроллера. Обмен информацией осуществляется асинхронно путем поочередной передачи информации по принципу "команда-ответ". Информация

передается в виде сообщений, которые могут состоять из командных слов, слов данных и ответных слов.

Рассмотрим пример работы шины в соответствии со стандартом MIL STD [Государственный стандарт СССР «Интерфейс магистральный последовательный системы электронных модулей» ГОСТ 26765.52-87.]. Стандарт допускает основные и групповые форматы сообщений. Форматы основных сообщений используются для передачи информации одному из оконечных устройств и предусматривают выдачу им ответного слова. Форматы групповых сообщений используются для передачи информации одновременно нескольким оконечным устройствам без выдачи ими ответных слов. Последняя группа форматов обеспечивает понижение загрузки шины, но при этом снижается надежность. Если требуется подтверждение факта приема оконечным устройством группового сообщения, то контроллер в этом случае (используя формат основного сообщения) может послать оконечному устройству команды “передать ответное слово” или “передать последнюю команду”, и факт приема группового сообщения может быть установлен контроллером путем анализа в ответном слове признака “принято групповое сообщение”. Каждый формат определяет количество и порядок следования командных слов, ответных слов и слов данных. Ниже приведены примеры форматов одиночного сообщения (рис.1.1.) и группового сообщения (рис.1.2.) (КС – командное слово, ОС – ответное слово, СД – слово данных, t_1, t_2 – паузы).

Основное сообщение (передача данных от оконечного устройства оконечному устройству).

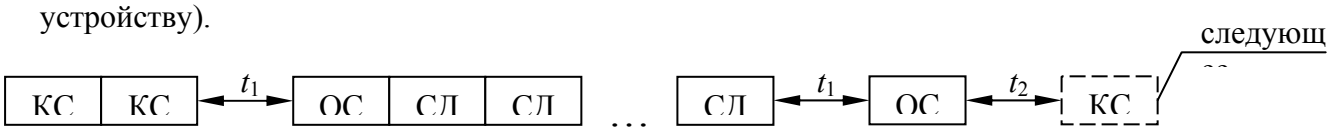

Рис.1.1. Пример формата одиночного сообщения.

Контроллер должен без паузы передать команду обмена данными с адресом оконечного устройства ***А*** на прием данных и команду обмена данными с адресом оконечного устройства ***Б*** на передачу данных. Оконечное устройство ***Б*** после установления факта достоверности принятой команды должно передать без пауз ответное слово и указанное в команде количество слов данных. Оконечное устройство ***А*** после установления факта достоверности адресованной ему информации должно передать ответное слово.

Групповое сообщение (рис.1.2.) (передача данных от оконечного устройства оконечным устройствам).

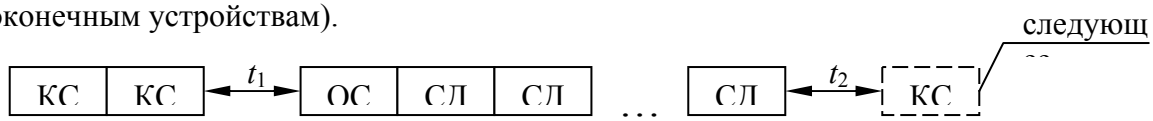

Рис.1.2. Пример формата группового сообщения.

Контроллер должен передать без паузы групповую команду обмена данными на прием данных и команду обмена данными с адресом одного оконечного устройства на передачу данных. Это оконечное устройство после установления факта достоверности принятого командного слова должно передать без пауз ответное слово и указанное в команде количество слов данных.

Шина в данной системе может рассматриваться как одноприборное устройство, обслуживающее исходно заданный набор работ без прерываний. Задача построения статических одноприборных расписаний без прерываний в общем виде может быть сформулирована следующим образом.

Дано:

- Множество работ, которые должны выполняться на системе $J = \{j\}_{j=1}^{N_j}$. Для каждой работы заданы $t_j > 0$ - время выполнения, $[s_j, f_j)$ - директивный интервал выполнения и выполняется условие $f_j - s_j \geq t_j$;
- Дополнительные ограничения $g_i(H, \bar{x}) \leq 0,\ i = 4 \ldots N_g$ на корректность расписания H, которые обусловлены технологическими особенностями шины и системного программного обеспечения;
- Вектор значений технологических требований $\bar{x} = (x_1 \ldots x_{N_x})$.

Расписание выполнения работ, которое представляет собой упорядоченное (по критерию время начала выполнения работы) множество $H = \left\{j_k, s_j^*\right\}_{k=1}^{N_H}$, $j \in J$, Здесь k - порядковый номер *j*-ой работы в расписании, s_j^* - время начала выполнения *j*-ой работы в расписании H, $f_j^* = s_j^* + t_j$ - время завершения выполнения *j*-ой работы. Множество корректных расписаний $H^{*'}$ (т.е. множество S в задаче (1.1.)) определим набором ограничений:

$$g_1 : (\forall j \in H) \Rightarrow \left((s_j^* \geq s_j) \wedge (f_j^* \leq f_j)\right)$$
$$g_2 : (\forall j \in H) \Rightarrow \left(f_j^* - s_j^* = t_j\right)$$
$$g_3 : (\forall (j,l) \in H, j \neq l) \Rightarrow \left(((s_j^* < s_l^*) \vee (s_j^* \geq f_l^*)) \wedge ((f_j^* \leq s_l^*) \vee (f_j^* > f_l^*))\right)$$

Ограничения g_1,g_2,g_3 являются обязательными для одноприборных расписаний без прерываний и соответственно означают:

1) интервал выполнения каждой работы располагается в рамках её директивного интервала;
2) не допустимы прерывания;
3) интервалы выполнения работ не пересекаются.

Задача:

$$\max_{H \in H^*} |H|$$

известна в теории расписаний как задача о выборе максимального числа совместимых заявок и является *NP*–трудной. Однако есть частные задачи этой задачи, которые являются полиномиально разрешимыми. Например, для частной задачи:

$$\max_{H \in H^*} |H|$$
$$\forall\ j\colon t_j = f_j - s_j$$

известен оптимальный жадный алгоритм сложности $O(n \cdot \log n)$ [Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. М.: МЦНМО, 2000.].

В задаче построения статических расписаний обменов по шине с централизованным управлением присутствуют дополнительные ограничения:

$$\max_{H \in H^*} |H|$$
$$g_i(H, \bar{x}) \le 0, i = 4..N_g,$$

Ограничения $g_i(H, \bar{x}) \le 0, i = 4 \ldots N_g$ определяются особенностями аппаратных и программных средств конкретной вычислительной системы реального времени. Приведем ограничения типичные для авиационных и навигационных систем для двух схем организации обменов: с подциклами и без подциклов.

В качестве исходных данных задается множество периодических сообщений $SM = \{M_i =< t_i, r_i > | \, i \in [1..n]\}$, где t_i – время передачи сообщения, r_i – частота передачи сообщения. Период передачи сообщения обозначим за $\tau_i = \frac{1}{r_i}$. Для каждого сообщения формируется множество соответствующих ему работ $J^i = \{ j =< t_j, s_j, f_j > | \, j \in [1..n_i]\}$, где $n_i = \left\lceil \frac{r_{scl}}{\tau_i} \right\rceil$ – количество экземпляров сообщения, которые надо передать в интервале планирования, r_{scl} – длительность интервала планирования, $s_j = (j-1) \cdot \tau_i$; $f_j = j \cdot \tau_i$ – соответственно левая и правая

границы директивного интервала сообщения *j*. Для некоторых сообщений могут быть заданы фазовые сдвиги, которые сужают директивный интервал сообщения, определяемый номером сообщения и периодом его передачи (рис.1.3.) Множество работ определяется как объединение множества работ соответствующих заданным сообщениям $J = \bigcup_{i=1}^{n} J^i$.

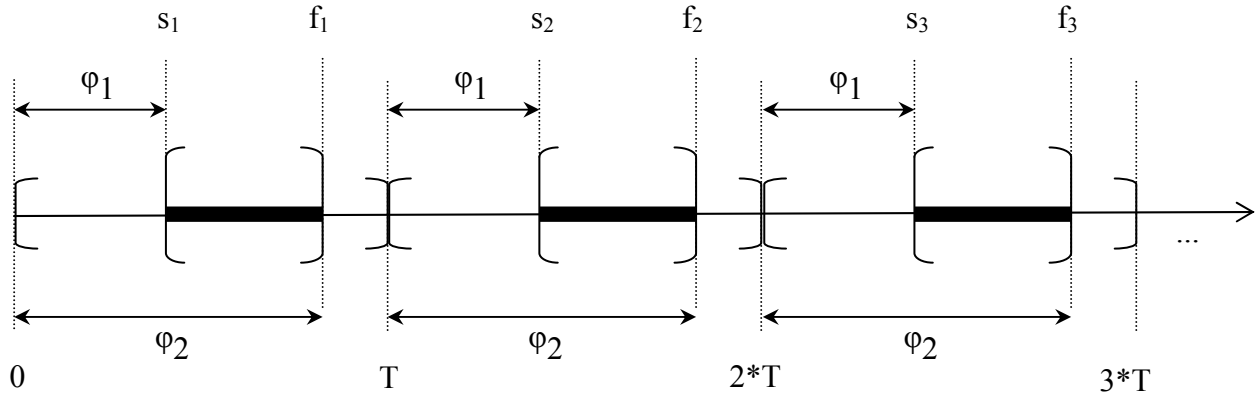

Рис. 1.3. Директивные интервалы работ

Для схемы с подциклами (рис.1.4.) (интервал планирования разбивается на отрезки равной длины - подциклы), возможны следующие дополнительные ограничения (g_i):

4) в каждом подцикле может находиться не более 1 цепочки работ и работы в цепочке следуют друг за другом без пауз;
5) время выполнения работ не должно пересекать границы подцикла;
6) время начала цепочки работ относительно начала соответствующего подцикла не должно быть меньше заданного значения;
7) в конце подцикла должен быть зарезервирован интервал времени, длительность которого не меньше, чем заданная доля длительности подцикла;
8) число работ в цепочке не должно превышать заданного значения;
9) сдвиг работы «вправо» по временной оси на время, не превышающее значение равное заданному проценту от интервала «время начала выполнения работы минус время начала цепочки» не должен приводить к нарушению директивного времени завершения работы или требования к минимальному резерву времени в конце подцикла.

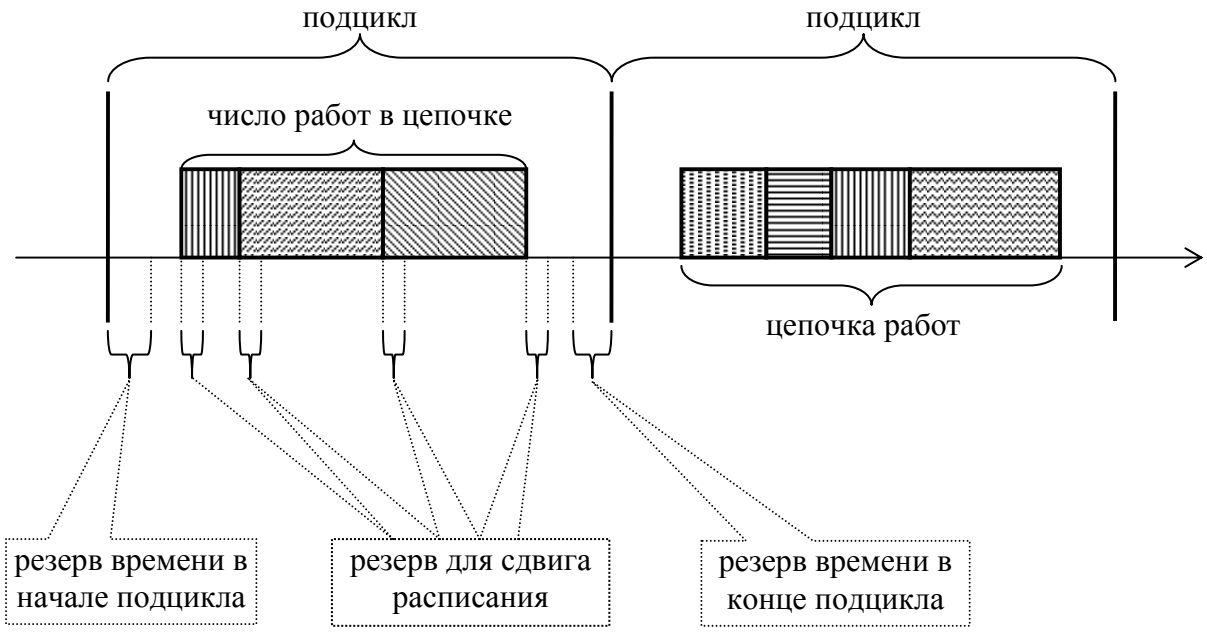

Рис. 1.4. Циклическая схема обмена данными

В схеме без подциклов возможны следующие дополнительные ограничения (g_i):

4) число работ в цепочке не должно превышать заданного значения;
5) суммарная длительность выполнения работ цепочки не должна превышать заданного значения;
6) интервал времени между последовательными цепочками должен быть не меньше заданного значения;
7) сдвиг работы «вправо» по временной оси на время, не превышающее значение равное заданному проценту от интервала «время начала выполнения работы минус время начала цепочки» не должен приводить к нарушению директивного времени завершения работы или требования к минимальному интервалу времени между последовательными цепочками.

С материалами данного раздела можно ознакомиться в работах:

С материалами данного раздела можно ознакомиться в работах:

1. Костенко В.А., Гурьянов Е.С. Алгоритм построения расписаний обменов по шине с централизованным управлением и исследование его эффективности// Программирование, 2005., N6. - С.67-76.
2. Балашов В.В. Алгоритмы формирования рекомендаций при планировании информационного обмена по каналу с централизованным управлением// Известия РАН. Теория и системы управления, 2007, N.6, с. 76-84.

3. Балашов В.В., Костенко В.А. Задачи планирования вычислений для одноприборных систем, входящих в состав вычислительных систем реального времени// Методы и средства обработки информации: Третья Всероссийская научная конференция. Труды конференции. - М.: Издательский отдел факультета ВМиК МГУ имени М.В. Ломоносова; МАКС Пресс, 2009. - С.193-203.

4. Костенко В.А. Алгоритмы построения расписаний для одноприборных систем, входящих в состав систем реального времени// Методы и средства обработки информации: Третья Всероссийская научная конференция. Труды конференции. - М.: Издательский отдел факультета ВМиК МГУ имени М.В. Ломоносова; МАКС Пресс, 2009. - С.245-258.

5. Р. Смелянский, В. Костенко, В. Балашов, В. Балаханов. Инструментальная система построения расписания обмена данными по каналу с централизованным управлением// Современные технологии автоматизации, 2011., N.3, С.78-84.

2. Алгоритмы случайного поиска

Основой методов случайного поиска служит итерационный процесс:

$$X^{k+1} = X^k + \alpha_k \cdot \frac{\xi}{\|\xi\|}, k = 0,1,\ldots,$$

где α_k – величина шага, $\xi = (\xi_1,\ldots,\xi_n)$ – некоторая реализация n-мерного случайного вектора ξ.

Ненаправленный случайный поиск (метод Монте-Карло) заключается в

$$X^{k+1} = \xi, \quad \xi \in S, \quad g_i(\xi) \le 0, \quad k = 0,1,\ldots, \quad i = 1,\ldots,m.$$

многократном случайном выборе допустимых вариантов решений и запоминании наилучшего из них:

Все алгоритмы направленного случайного поиска без самообучения делают шаг от текущего значения X^k оптимизируемых параметров. Известны следующие алгоритмы направленного случайного поиска без самообучения [Л.А. Растригин. Статистические методы поиска.- М.: Наука, 1968.]: алгоритм с парной пробой, алгоритм с возвратом при неудачном шаге, алгоритм с пересчетом при неудачном шаге, алгоритм с линейной экстраполяцией, алгоритм наилучшей пробы, алгоритм статистического градиента.

Алгоритмы случайного направленного поиска с самообучением заключаются в перестройке вероятностных характеристик поиска, т.е. в определенном целенаправленном воздействии на случайный вектор ξ. Он уже перестает быть равновероятным и в результате самообучения приобретает определенное преимущество в направлениях наилучших шагов. Это достигается введением вектора $P^k = (p_1^k, p_2^k, \ldots, p_n^k)$, где p_j^k - вероятность выбора положительного направления по j-ой координате на k-ом шаге. Алгоритм рекуррентно корректирует значение компонентов этого вектора на каждой итерации в зависимости от того, насколько удачным/неудачным (изменилось значение целевой функции) был сделанный шаг.

Разделы 2.1. и 2.2. будут читаться в соответствии с [Л.А. Растригин. Статистические методы поиска.- М.: Наука, 1968.].

2.1. Ненаправленный случайный поиск (метод Монте-Карло)

2.2. Алгоритмы направленного случайного поиска без самообучения

2.2.1. Алгоритм с парной пробой

2.2.2. Алгоритм с возвратом при неудачном шаге

2.2.3. Алгоритм с пересчетом при неудачном шаге

2.2.4. Алгоритм с линейной экстраполяцией

2.2.5. Алгоритм наилучшей пробы

2.2.6. Алгоритм статистического градиента

2.3. Алгоритмы случайного направленного поиска с самообучением. Различные способы коррекции вектора направления наилучших шагов

3. Алгоритмы имитации отжига

3.1. Концепция построения алгоритмов

Алгоритмы имитации отжига в процессе поиска оптимального решения с некоторой вероятностью допускают переход в состояние с более высоким значением целевой функции. Это свойство позволяет им выходить из локальных оптимумов. Принципы, положенные в основу работы алгоритмов, можно объяснить на следующей физической аналогии [Уоссермен Ф. Нейрокомпьютерная техника. Теория и практика.- М.: Мир, 1992. – 240с.]. На рис.3.1. изображен шарик в коробке, внутренняя поверхность которой соответствует ландшафту целевой функции. При сильном встряхивании коробки в горизонтальном направлении шарик может переместиться из любой точки в любую другую. При постепенном уменьшении силы встряхивания будет достигнуто условие, когда сила встряхивания достаточна для перемещения шарика из точки A в точку B, но недостаточна для того, чтобы шарик мог переместиться из B в A. При дальнейшем уменьшении силы встряхивания до нуля шарик остановится в точке B - точке глобального минимума. В алгоритмах имитации отжига аналогом силы встряхивания является вероятность перехода в состояние с более высоким значением целевой функции. В начале работы алгоритма эта вероятность должна быть достаточно велика, чтобы была возможность перехода от выбранного начального решения к любому другому решению. В процессе работы алгоритма вероятность перехода уменьшается в соответствии с выбранным законом.

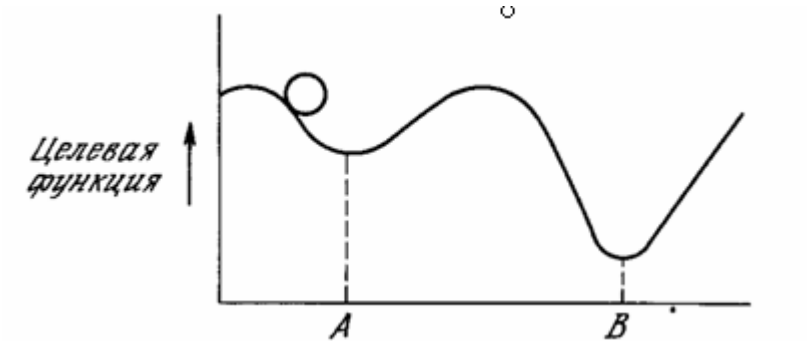

Рис. 3.1. Иллюстрация принципов работы алгоритмов имитации отжига.

3.2. Общая схема алгоритмов и проблемы построения алгоритмов для решения задач условной оптимизации

Для решения задач условной оптимизации схему имитации отжига можно представить следующим образом:

Ш а г 1. Задать начальное корректное решение $X^0 \in S$ и считать его текущим вариантом решения ($X=X^0$).

Ш а г 2. Установить начальную температуру T_0, приняв ее текущей ($T=T_0$).

Ш а г 3. Применить операции преобразования решения к текущему решению X и получить новый корректный вариант решения $X' \in S$.

Ш а г 4. Найти изменение функционала оценки качества решения $\Delta F = F(f(X'), g_i(X')) - F(f(X), g_i(X))$:

- если $\Delta F \le 0$ (решение улучшилось), то новый вариант решения считать текущим ($X=X'$);
- если $\Delta F > 0$ (решение ухудшилось), то принять с вероятностью $p = e^{\frac{-\Delta F}{T}}$ в качестве текущего решения новый вариант решения X'.

Ш а г 5. Повторить заданное число раз шаги 3 и 4 без изменения текущей температуры.

Ш а г 6. Если критерий останова выполнен, то завершение работы алгоритма.

Ш а г 7. Понизить текущую температуру в соответствии с выбранным законом понижения и перейти к шагу 3.

Для построения алгоритма имитации отжига для решения конкретной задачи условной оптимизации требуется решить следующие задачи:

1. Разработать способ представления решения X и операций преобразования текущего решения на шаге 3.
2. Разработать стратегию применения операций преобразования текущего решения на шаге 3: какую операцию применять, к какому элементу X, как изменять его значение.
3. Выбрать закон понижения температуры на шаге 7.
4. Определить функционал $F(f(X), g_i(X))$, используемый для оценки качества текущего решения на шаге 4.
5. Выбрать критерий останова алгоритма, используемый на шаге 6.

Если есть g_i не вошедшие в функционал $F(f(X), g_i(X))$, то на шаге 3 после применения операций преобразования решения, должно получаться решение,

удовлетворяющее этим ограничениям. Наиболее часто функционал $F(f(X), g_i(X))$ строится с использованием функций штрафа или барьерных функций [Мину].

Способ представления решения X и операции преобразования текущего решения должны удовлетворять условиям замкнутости (после применения любой операции к корректному решению получается корректное решение) и полноты (для двух любых корректных решений *X, X** существует конечная последовательность операций, приводящая *X* к *X**, такая, что все промежуточные решения корректны). Условия замкнутости и полноты системы операций преобразования решений являются достаточными условиями того, что алгоритм за конечное число итераций может перейти от начально-заданного решения к оптимальному решению.

Стратегия применения операций преобразования текущего решения и закон понижения температуры определяют вычислительную сложность и точность алгоритма.

При построении алгоритмов имитации отжига наиболее часто используются следующие законы понижения температуры:

- Закон Больцмана: $T = \frac{T_0}{\ln(1+x)}$.
- Закон Коши: $T = \frac{T_0}{1+x}$.
- $T = T_0 \frac{\ln(1+x)}{1+x}$.

Здесь T - текущая температура алгоритма, T_0 - начальная температура, x - номер итерации алгоритма. Начальная температура является параметром алгоритма и задается в качестве исходных данных.

Использование этих законов понижения понижение температуры в алгоритмах имитации отжига для обучения нейросетей прямого распространения (т.е. для решения квадратичных задач безусловной оптимизации) выявило следующие их недостатки [Уоссермен Ф. Нейрокомпьютерная техника. Теория и практика.- М.: Мир, 1992. – 240с.]. Так, использование закона понижения температуры Больцмана требует очень высоких начальных температур и чрезмерно большого числа итераций для получения приемлемого по точности решения. При относительно невысоких температурах алгоритм с данным режимом понижения температуры не находит приемлемых по точности решений. Использование закона понижения температуры Коши значительно уменьшается число итераций, производимых алгоритмом, однако при этом возможно “зависание” алгоритма в локальных оптимумах. Алгоритм с законом понижения температуры $T = T_0 \frac{\ln(1+x)}{1+x}$,

также подвержен, хотя и в меньшей степени, возможности “зависания” в локальных оптимумах. Однако число итераций хоть и существенно меньше, чем при использовании закона Больцмана, но значительно превышает число итераций алгоритма, использующего закон Коши при сохранении качества решений.

3.3. Модификация закона понижения температуры, позволяющая ускорить выход из локальных оптимумов

В работе [Костенко В.А., Калашников А.В. Исследование различных модификаций алгоритмов имитации отжига для решения задачи построения многопроцессорных расписаний// Дискретные модели в теории управляющих систем. Труды VII Международной конференции. М.: МАКС Пресс, 2006. - С.179-184.] предлагается временно повышать температуру при попадании алгоритма в локальный оптимум для повышения вероятности выхода из него. В качестве критерия попадания в локальный оптимум, используется количество итераций, в течение которых решение не улучшается. В работе предложен следующий закон изменения температуры:

$$T = \min(T_0, f(T_0, x) + \max(0, c_t(k - k_t))), \text{ где}$$

- $f(T_0, x)$ - стандартная функция понижения температуры, например функция Больцмана или Коши,
- k - количество итераций алгоритма без улучшения решения,
- k_t - количество итераций алгоритма без улучшения решений после которых начинается повышение температуры (этот параметр подбирается таким образом, чтобы k_t итераций хватало для нахождения локального оптимума с достаточной точностью),
- c_t - коэффициент, характеризующий скорость увеличения температуры.

3.4. Методы распараллеливания алгоритмов имитации отжига

Многие лучшие решения *NP*-трудных задач были получены алгоритмами имитации отжига. Все эти алгоритмы выполнялись на многопроцессорных вычислительных системах. В этом разделе рассмотрим известные методы распараллеливания алгоритмов имитации отжига.

3.4.1. Асинхронный параллельный алгоритм

Данный метод подробно рассмотрен в работе [Z. J. Czech. Parallel Simulated Annealing for the Delivery Problem. // Ninth Euromicro Workshop on Parallel and Distributed Processing. 2001. Mantova Italy. P 219-226.] для задачи коммивояжера. Параллельные процессы независимо выполняют алгоритм имитации отжига, стартуя с различных начальных приближений. По окончании

работы алгоритма процессы отсылают наилучшее решение координатору, который выбирает среди них оптимальное. На рисунке 3.2. схематично показана работа асинхронного параллельного алгоритма имитации отжига.

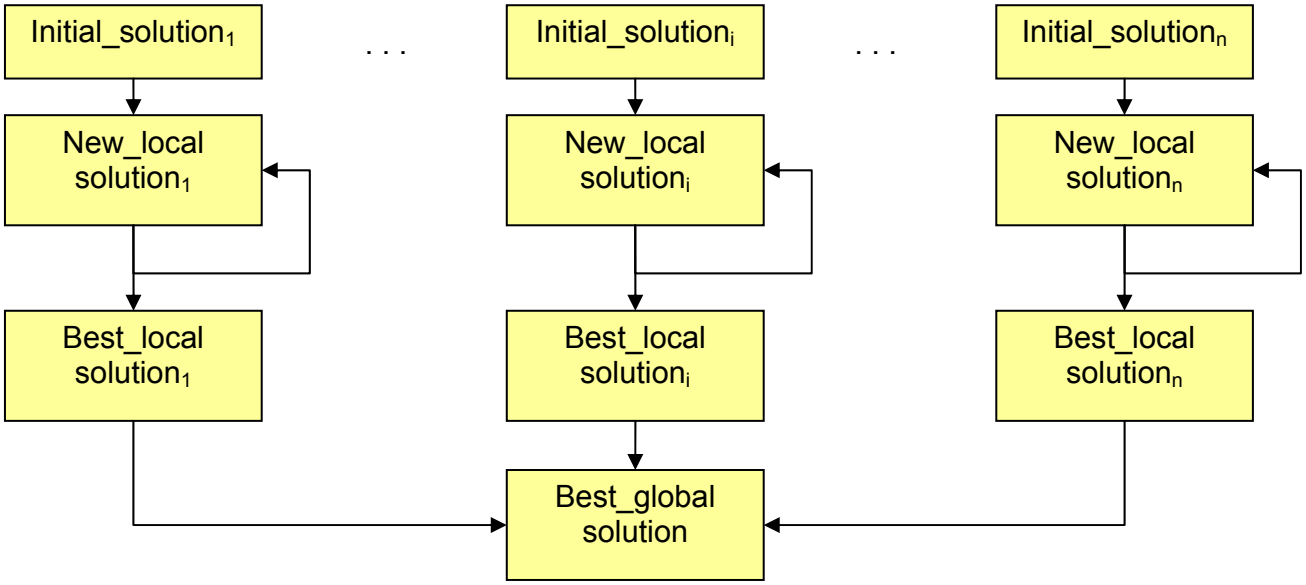

Рисунок 3.2. Схема параллельного асинхронного алгоритма.

Основными достоинствами данного метода являются простота распараллеливания на любое число процессоров и низкие требования к среде обмена. В работе [Z. J. Czech. Parallel Simulated Annealing for the Delivery Problem. // Ninth Euromicro Workshop on Parallel and Distributed Processing. 2001. Mantova Italy. P 219-226.] показано, что скорость поиска решений и качество получаемых решений практически не отличаются от классического последовательного алгоритма.

Техника распараллеливания с помощью асинхронного параллельного алгоритма является универсальной и может быть использована для любых приложений алгоритма имитации отжига.

3.4.2. Параллельный алгоритм с синхронизацией

В работах

1. Z. J. Czech. Parallel Simulated Annealing for the Delivery Problem. // Ninth Euromicro Workshop on Parallel and Distributed Processing. 2001. Mantova Italy. P 219-226.
2. Schmid, M., Schneider, R. Parallel Simulated Annealing Techniques for Scheduling and Mapping DSP-Applications onto Multi-DSP Platforms // In Proceedings of the International Conference on Signal Processing Applications & Technology. 1999. Orlando, U.S.A.: Miller Freeman, Inc. http://citeseer.ist.psu.edu/schmid99parallel.htm.
3. J.M.Varanelli. On the Acceleration of Simulated Annealing // PhD thesis, University of Virginia, USA. 1996. P 77-81.

4. Low Chinyao, Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines // Computers and operations research. 2005. 32. №8. P 2013-2025.

рассматриваются различные подходы к распараллеливанию алгоритма имитации отжига, объединенные общей идей. Параллельные процессы начинают поиск решения с одного исходного приближения. Через фиксированное число итераций процессы обмениваются своими решениями и корректируют свое текущее приближение. На рисунке 3.3. схематично показана работа асинхронного параллельного алгоритма имитации отжига.

Существуют различные способы обмена полученными решениями и стратегии принятия решения в качестве текущего.

Например:

- Процесс i посылает процессу $i+1$ своё текущее решение. Процесс $i+1$ принимает это решение в качестве текущего, если $F(X_i) < F(X_{i+1})$. Где $F(X_i)$ – значение целевой функции текущего решения i-ого процесса.
- Широковещательный обмен решениями. Принятие j-ым процессом решения от i-ого процесса в качестве текущего, если $F(X_i) < F(X_j)$ и $i < j$.
- Отправка текущих решений координатору, выбор из них произвольного (возможно с разными вероятностями, в зависимости от значений целевой функции) и принятие его в качестве текущего всеми процессами.

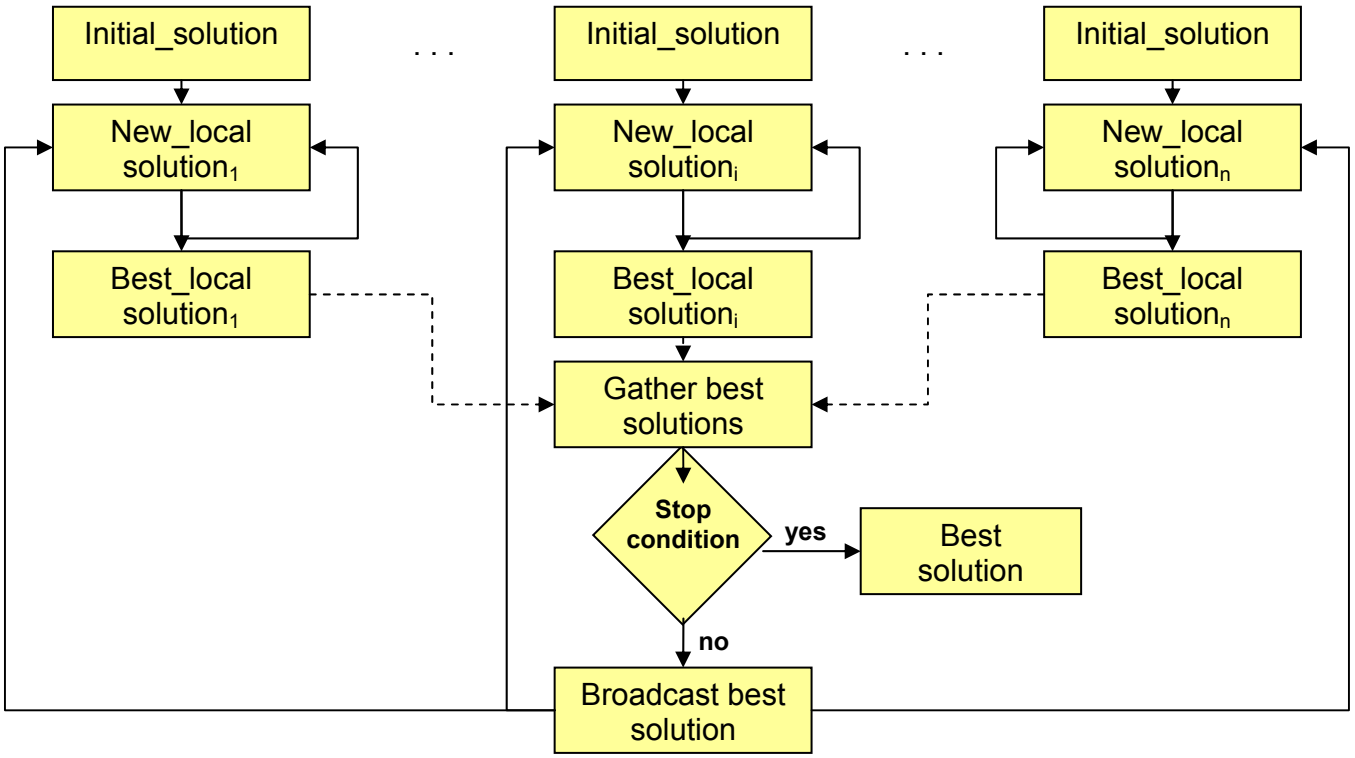

Рисунок 3.3. Схема параллельного алгоритма с синхронизацией

Похожая техника распараллеливания рассмотрена в работе [J.C.Gomez. A General Interface for Distributed and Sequential Simulated Annealing [HTML] (http://citeseer.ist.psu.edu/110625.html).]. На разных процессорах алгоритм имитации отжига работает при различных температурах и режимах понижения температуры. Процесс с номером $i+1$ отвечает за локализацию решения, полученного процессом с номером i, алгоритм имитации отжига процесса $i+1$ работает при меньшей температуре и более медленном режиме понижения температуры. Если процесс i получает решение X_i лучше, чем текущее решение X_{i+1} процесса $i+1$, то решение X_i принимается в качестве текущего для процесса $i+1$.

Среди достоинств данного подхода стоит отметить хорошую масштабируемость и универсальность, из недостатков высокие требования к среде обмена, обусловленные большим числом обменов (особенно на начальной стадии работы алгоритма, при высоких температурах) и несущественное уменьшение времени работы алгоритма с ростом числа процессоров.

3.4.3. Алгоритм, основанный на декомпозиции целевой функции

Большую часть времени алгоритм имитации отжига затрачивает на вычисление целевой функции. Если целевая функция является декомпозируемой $F(x_1, x_2, \dots, x_i, \dots, x_n) = F(x_1) + F(x_2) + \dots + F(x_i) + \dots + F(x_n)$, то возможно параллельное её вычисление на m ($m \le n$) процессорах.

В работах

1. R. Azencott, Ed., Simulated Annealing: Parallelization Techniques. New York: John Wiley and Sons. 1992. P 47-79.
2. S.A. Kravitz and R.A. Rutenbar. Placement by Simulated Annealing on a Multiprocessor // IEEE Trans. CADICS. 1987. №6. P 534-549.
3. Stefka Fidanova, Simulated Annealing for Grid Scheduling Problem //. IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing (JVA'06). 2006. P 41-45.

рассмотрены задачи, для которых удается существенно повысить скорость работы алгоритма имитации отжига благодаря использованию декомпозиции целевой функции. Одной из таких задач является задача размещения транзисторов на интегральной схеме. Если в качестве целевой функции рассматривать количество перекрывающихся элементов, то оно может быть посчитано, как сумма перекрывающихся элементов отдельных участков интегральной схемы.

Несомненным достоинством данного метода является практически линейный рост производительности с ростом числа процессоров, тем не менее, возможности по масштабированию ограничены видом целевой функции. Главным недостатком является

узкая направленность метода, возможность его применения только для задач с декомпозируемой целевой функцией.

3.4.4. Подходы, основанные на декомпозиции пространства решений

В некоторых случаях возможно разбить все пространство решений на области [D.Janaki Ram, T.H.Sreenivas, K.Ganapathy Subramaniam. Parallel Simulated Annealing Algorithms // Journal of parallel and distributed computing. 1996. №37. P 207-212.], [J.Allwright, D.Carpenter. A distributed implementation of simulated annealing for traveling salesman problem // Parallel Computing. 1989. 3. №9-10. P. 335-338.]. В таком случае возможна параллельная работа алгоритма имитации отжига в различных областях. Для каждой конкретной задачи разбиение на области осуществляется индивидуально с учётом индивидуальных особенностей пространства *S*. При использовании такого способа распараллеливания необходимо обеспечивать полное покрытие пространства решений непересекающимися областями. Следует позаботиться об изменении операций преобразования расписания таким образом, чтобы полученное на следующем шаге решение не выходило за рамки области. Итоговое решение получается как наилучшее среди решений, полученных во всех областях.

На рисунке 3.4. схематично показана работа параллельного алгоритма имитации отжига, основанного на декомпозиции пространства решений.

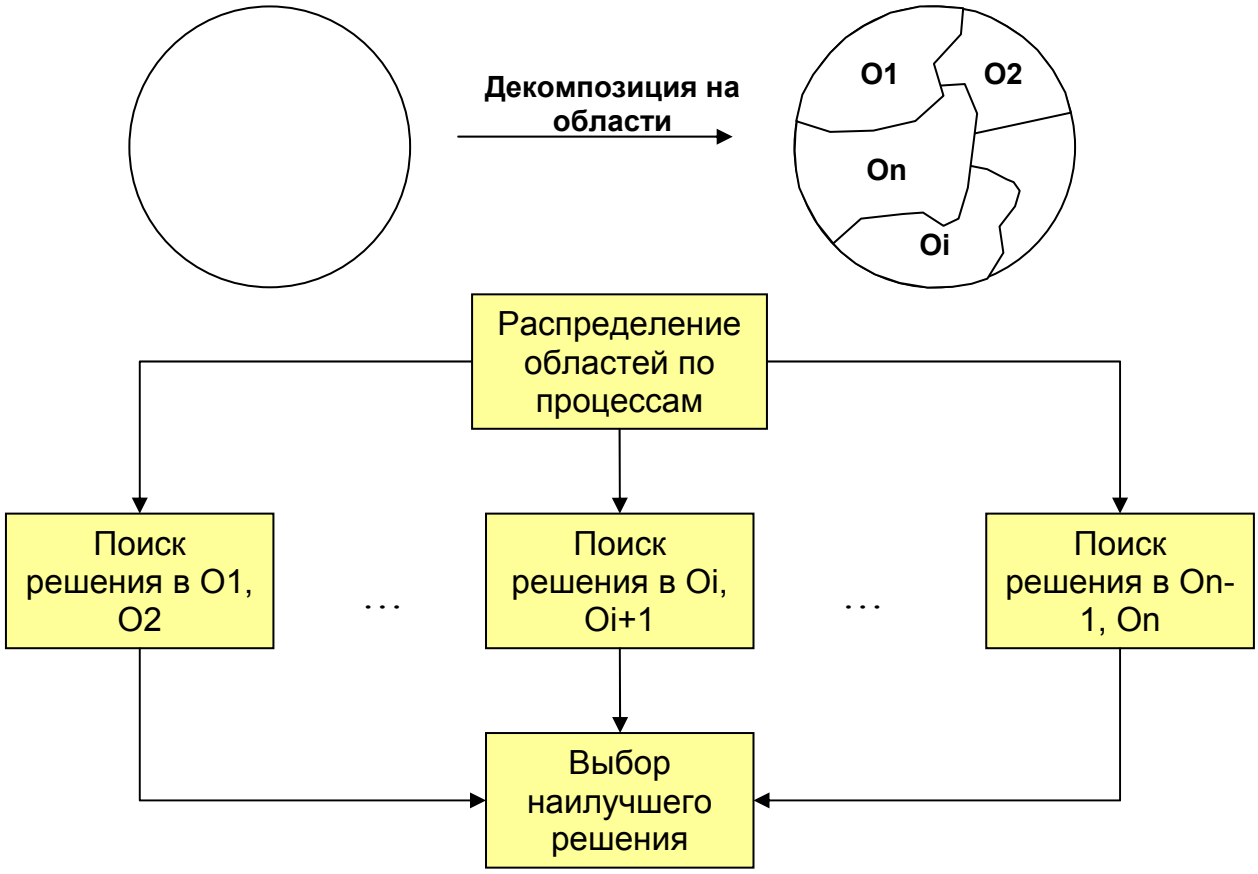

Рисунок 3.4. Схема параллельного алгоритма, использующего декомпозицию пространства решений.

К преимуществам данного подхода следует отнести: сужения пространства поиска решений, значительное уменьшение времени работы алгоритма и хорошие возможности по масштабированию (при увеличении количества процессоров необходимо лишь увеличить количество областей разбиения, или иначе распределять области между процессорами). Основным недостатком является узкая направленность подхода (для каждого приложения алгоритма имитации отжига надо разрабатывать свою схему разбиения на области и соответствующим образом менять операцию преобразования текущего решения).

3.5. Алгоритм имитации отжига для решения задачи построения статических многопроцессорных расписаний с минимальным временем выполнения на заданном числе процессоров

Напомним, что для построения алгоритма имитации отжига для решения конкретной задачи условной оптимизации требуется решить следующие задачи:

6. Разработать способ представления решения X и операций преобразования текущего решения.
7. Разработать стратегию применения операций преобразования текущего решения: какую операцию применять, к какому элементу X, как его изменять.
8. Выбрать закон понижения температуры.
9. Определить функционал $F(f(X), g_i(X))$, используемый для оценки качества текущего решения.
10. Выбрать критерий останова алгоритма.

В данном разделе рассмотрим решение этих проблем при построении алгоритма имитации отжига для решения задачи построения статических многопроцессорных расписаний с минимальным временем выполнения на заданном числе процессоров.

3.5.1. Математическая формулировка задачи

Задачу построения расписаний будем рассматривать в следующем варианте постановки (математические представления модели программы, расписания и условий его корректности приведены в разделе 1.4.1.)

Дано: H(PR)=(P,≺)- модель программы, *T=f(HP,HW)*- функция вычисления времени выполнения расписания *HP* на архитектуре *HW* (целевая функция).

Требуется построить: HP– расписание выполнения программы на заданном числе процессоров *M* такое, что:

$$\min_{HP} f(HP, HW)$$
$$HP \in HP^{*}_{1-5}$$

Функция вычисления времени выполнения расписания может быть задана в аналитическом виде или в виде имитационной модели.

3.5.2. Способы представления расписания и операций его преобразования

Бинарное представление расписания.

Расписание задается:

матрицей привязки $Y(HP)_{N\times M}$ и матрицей смежности $X(HP)_{N\times N}$ графа *HP*, где элементы матриц определяются:

$$y_{ij} = \begin{cases} 1, \textit{если } p_i \in SP_j \\ 0, \textit{если } p_i \notin SP_j \end{cases} \qquad x_{ij} = \begin{cases} 1, \textit{если } \prec_{ij} \in \prec_{HP} \\ 0, \textit{если } \prec_{ij} \notin \prec_{HP} \end{cases}$$

(первый индекс рабочий интервал-предшественник, второй индекс рабочий интервал-приемник). Где *M* – число процессоров в ВС, *N* – число рабочих интервалов в *H*.

Недостатком данного представления является большое число бинарных переменных $N^2+N\cdot M$.

Целочисленное представление расписания

1. Расписание задается матрицей $Y(HP)_{N\times M}$, где элемент матрицы определяется:

$$y_{ij} = \begin{cases} c, \textit{если } p_i \in SP_j \\ 0, \textit{если } p_i \notin SP_j \end{cases},$$

c – порядковый номер рабочего интервала p_i в SP_j.

При данном способе представления расписания число целочисленных переменных равно $N\cdot M$.

2. Расписание задается: вектором привязки $Y(HP)_K$ и вектором порядка $X(HP)_N$, где *i*-й элемент вектора $Y(HP)_K$ равен номеру списка в который назначены рабочие интервалы *i*-го процесса, а *i*-й элемент вектора $X(HP)_N$ равен порядковому номеру рабочего интервала

в соответствующем списке. При данном способе представления расписания число целочисленных переменных равно $K+N$.

Операции преобразования расписания

Можно выделить следующие варианты отличия расписаний HP и HP' друг от друга:

- расписания HP и HP' отличаются лишь порядком рабочих интервалов как минимум в одном SP_j;
- расписания HP и HP' отличаются привязкой рабочих интервалов.

Введем соответствующие операции преобразования расписаний, позволяющие устранить указанные варианты отличия: $O=\{O1,O2\}$ [Костенко В.А. Задача построения расписания при совместном проектировании аппаратных и программных средств// Программирование, 2002., №3. - С.64-80.]. Операции $O1,O2$ определим для целочисленного непосредственного представления расписания. Операции будем определять при предположениях: каждый процесс имеет не более одного рабочего интервала или при условии, что на расписание не накладывается ограничение 5. После доказательства теоремы о замкнутости и полноте системы операций $O=\{O1,O2\}$ и получении условий их применимости покажем возможность снятия указанных предположений.

Операция изменения порядка рабочих интервалов в одном списке изменяет порядковый номер рабочего интервала p_i в списке SP_m (порядковый номер рабочего интервала становится равным c) и корректирует порядковые номера соответствующих рабочих интервалов в данном списке (NS_m – число рабочих интервалов в списке SP_m):

$$O1(p_i, SP_m, c) \equiv \begin{cases} \left\{ c1 = y_{im},\ y_{im} = c \in (1,\ldots,c1-1); \forall j \middle|\ y_{jm} \neq 0 \vee c \le y_{jm} < c1 : y_{jm} = y_{jm} + 1 \right\} - \\ - \textit{уменьшение порядкового номера} \\ \left\{ c1 = y_{im},\ y_{im} = c \in (c1+1,\ldots,NS_m); \forall j \middle|\ y_{jm} \neq 0 \vee c1 < y_{jm} \le c : y_{jm} = y_{jm} - 1 \right\} - \\ \textit{увеличение порядкового номера} \end{cases}$$

Операция изменения привязки рабочих интервалов переносит рабочий интервал p_i из списка SP_m в список SP_k (порядковый номер рабочего интервала становится равным c) и корректирует порядковые номера соответствующих рабочих интервалов в списках SP_m и SP_k:

$$O2(p_i, SP_m \to SP_k, c) \equiv \{ y_{ik} = c \in (1 \ldots NS_k + 1); \forall j \middle|\ y_{jk} \neq 0 \vee c \le y_{jk} : y_{jk} = y_{jk} + 1,\ NS_k = NS_k + 1; \\ \forall j \middle|\ y_{jm} \neq 0 \vee y_{im} < y_{jm} : y_{jm} = y_{jm} - 1,\ NS_m = NS_m - 1,\ y_{im} = 0 \}$$

Указанные интервалы параметра операций *c* могут привести к нарушению ограничений 3,4. Покажем замкнутость и полноту системы операций *O={O1,O2}* и получим условия их применимости (выбор значения параметра *c*) не нарушающие ограничений 3,4.

Теорема 1. Если *HP* и *HP´* - произвольные корректные расписания ($HP, HP' \in HP^{*}_{1-4}$), то существует конечная цепочка команд $\{O_i\}_{i=1}^{K}, O_i \in \{O1, O2\}$, переводящая расписание *HP* в *HP´*, такая, что все *K* промежуточных расписаний являются корректными и $K \le 2N$.

Доказательство. Введем понятие канонического расписания HP^0: все рабочие интервалы находятся в SP_1 и порядковые номера рабочих интервалов в SP_1 равны номерам рабочих интервалов в графе *H*. HP^0 является допустимым, поскольку нумерация рабочих интервалов в графе *H* удовлетворяет условиям полной топологической сортировки. Полноту системы операций *{O1,O2}* докажем, показав, что существует стратегия (последовательность выбора рабочих интервалов, операция для каждого интервала из последовательности и значение параметра *c*) с числом шагов $K \le N$ позволяющая перевести произвольное расписание *HP* в расписание HP^0, такая, что все промежуточные расписания HP^i (полученные после выполнения отдельной операции O_i) являются допустимыми и любая операция из цепочки $\{O_i\}_{i=1}^{K}, O_i \in \{O1, O2\}$ является обратимой.

Применение операций *O1,O2* не может привести к нарушению ограничений 1,2 по определению операций при любой стратегии.

Расписания *HP,* HP^0 и HP^i будем представлять в ярусной форме максимальной высоты. Покажем, что следующая стратегия позволяет перевести *HP* в HP^0, не нарушая ограничений 3-4 для промежуточных расписаний HP^i:

1) выбор рабочих интервалов осуществляется в соответствии с их номерами в *H*;
2) если очередной рабочий интервал $p_i^s \in SP_1$, то применяем *O1*; если $p_i^s \notin SP_1$, то применяем *O2* (нижний индекс – номер рабочего интервала в *H*, верхний – номер яруса на котором расположен рабочий интервал);
3) для рабочего интервала p_i^s параметр *c=i*.

Рассмотрим применение данной стратегии к рабочему интервалу с номером *i=1*. Если рабочий интервал находится в SP_1 на первом ярусе, то его перенос не требуется. Если рабочий интервал находится в SP_1 на ярусе отличном от первого, то увеличиваем на *1* номера ярусов с первого яруса по ярус, предшествующий ярусу на котором находился первый рабочий интервал, и применяем к нему операцию *O1*. Полученное при этом расписание HP^1 удовлетворяет ограничению 3 в силу того, что нумерация рабочих

интервалов в H удовлетворяет условиям полной топологической сортировки, и является ярусным, следовательно, удовлетворяет ограничению 4. Если рабочий интервал не находится в SP_1, то применяем к нему операцию $O2$, осуществляющим его перенос в SP_1 на первый ярус. Если рабочий интервал находился на ярусе отличном от первого, то аналогично, как и при применении операции $O1$, корректируем номера ярусов. Полученное при этом расписание HP^1 удовлетворяет ограничению 3 в силу того, что нумерация рабочих интервалов в H удовлетворяет условиям полной топологической сортировки, и является ярусным, следовательно, удовлетворяет ограничению 4. Поскольку, HP допустимое расписание, то операции $O1/O2$ являются обратимыми (параметр c в обратной операции принимает старый номер рабочего интервала).

Пусть расписание HP^{i-1} является корректным. На предыдущих шагах все предшественники i-го рабочего интервала перенесены на ярусы лежащие выше i-го яруса в SP_1. Переносим i-й рабочий интервал в SP_1 на i-й ярус, используя операцию $O1/O2$. Если i-й рабочий интервал находился не на i-м ярусе, то перед применением операции увеличиваем на 1 номера ярусов с i-го яруса по ярус, предшествующий ярусу на котором находился i-й рабочий интервал. Полученное расписание HP^i, удовлетворяет ограничениям 3 и 4, так как все предшественники i-го рабочего интервала находятся на ярусах выше i-го яруса, последователи на ярусах ниже i-го яруса, и граф HP^i представлен в ярусной форме максимальной высоты. Поскольку, HP^{i-1} и HP^i корректные расписания, то операции $O1/O2$ являются обратимыми (параметр c в обратной операции принимает старый номер рабочего интервала).

После обхода всех вершин в соответствии с используемой стратегией получим расписание HP^0. Если операции $O1/O2$ применялись для всех рабочих интервалов то $K=N$. Поскольку, каждая операция из цепочки $\{O_i\}_{i=1}^{K}, O_i \in \{O1, O2\}$ обратима, то существует стратегия перехода от HP^0 к произвольному HP, такая, что все K промежуточных расписаний являются корректными и $K \le N$. Следовательно, существует стратегия перехода от произвольного корректного варианта расписания HP к произвольному корректному варианту расписания HP', такая, что все промежуточные расписания корректны и $K \le 2N$.

Следствие 1. Существует стратегия перехода от произвольного корректного расписания к оптимальному расписанию, такая, что длина цепочки команд $\{O_i\}_{i=1}^{K}, O_i \in \{O1, O2\}$, переводящей произвольное корректное расписание в оптимальное, не превосходит значения $2N$ ($K \le 2N$) и все K промежуточных расписаний являются корректными.

Условия применимости операций не нарушающие ограничений на HP

Получим интервал значений параметра *c* при применении операции *O1/O2* к рабочему интервалу p_i^s. Расписания *HP* будем представлять в ярусной форме максимальной высоты. Обозначим через $IN_i = \{p_k^l :\prec_{ki} \neq 0\}$ - множество непосредственных предшественников рабочего интервала p_i^s (всегда выполняется *k<i* и *l<s*), $OUT_i = \{p_k^l :\prec_{ik} \neq 0\}$ - множество непосредственных последователей рабочего интервала p_i^s (всегда выполняется *k>i* и *l>s*). Операция $lin = \max_{IN_i}(l)$ - получает максимальный номер яруса, на котором расположен один из непосредственных предшественников рабочего интервала p_i^s, для рабочих интервалов, не имеющих предшественников *lin=0* (нулевой ярус всегда пуст). Операция $lout = \min_{OUT_i}(l)$ - получает минимальный номер яруса, на котором расположен один из непосредственных последователей рабочего интервала p_i^s, для рабочих интервалов, не имеющих последователей *lout=N* (*N* – число ярусов в *HP*, для ярусной формы максимальной высоты число ярусов всегда равно числу рабочих интервалов). Тогда, рабочий интервал p_i^s может быть размещен в любой из списков SP_j на любой из ярусов из интервала *lin<s<lout*. Если выбранный ярус занят, то осуществляется коррекция ярусной формы путем соответствующего сдвига ярусов.

Пусть рабочий интервал p_i^s переносится в SP_j или изменяется его порядковый номер в этом списке. Разобьем множество SP_j на три подмножества: $PIN_j = \{p_k^l : l \geq lin,\ p_k^l \in SP_j\}$ - множество рабочих интервалов из списка SP_j, находящихся на ярусах, расположенных выше яруса *(lin-1)*; $PI_j = \{p_k^l : lin < l < lout,\ p_k^l \in SP_j\}$ - множество рабочих интервалов из списка SP_j, находящихся на ярусах $]lin, lout[$; $POUT_j = \{p_k^l : l \leq lout,\ p_k^l \in SP_j\}$ - множество рабочих интервалов из списка SP_j, находящихся на ярусах, расположенных ниже яруса *(lout+1)*. Параметр *c* при размещении рабочего интервала p_i^s в SP_j может принимать следующие значения, не нарушающие ограничения на *HP* (индекс *j* для *PIN, PI, POUT* будем опускать):

	PIN	*PI*	*POUT*	*C*
1	*PIN=∅*	*PI=∅*	*POUT=∅*	*c=1*
2	*PIN=∅*	*PI=∅*	*POUT≠∅*	*c=1*

3	$PIN=\varnothing$	$PI\neq\varnothing$	$POUT=\varnothing$	$1 \le c \le \max\limits_{PI}(y_{jk})+1$
4	$PIN=\varnothing$	$PI\neq\varnothing$	$POUT\neq\varnothing$	$1 \le c \le \max\limits_{PI}(y_{jk})+1$
5	$PIN\neq\varnothing$	$PI=\varnothing$	$POUT\neq\varnothing$	$c = \min\limits_{PIN}(y_{jk})+1$
6	$PIN\neq\varnothing$	$PI=\varnothing$	$POUT=\varnothing$	$c = \min\limits_{PIN}(y_{jk})+1$
7	$PIN\neq\varnothing$	$PI\neq\varnothing$	$POUT=\varnothing$	$\min\limits_{PI}(y_{jk}) \le c \le \max\limits_{PI}(y_{jk})+1$
8	$PIN\neq\varnothing$	$PI\neq\varnothing$	$POUT\neq\varnothing$	$\min\limits_{PI}(y_{jk}) \le c \le \max\limits_{PI}(y_{jk})+1$

Указанные выше интервалы значений параметра *c* не нарушающие ограничения на *HP* могут быть расширены, если ярусная форма максимальной высоты будет приведена к такому виду, что все рабочие интервалы SP_j, которые не находятся в отношении транзитивного порядка с непосредственным предшественником рабочего интервала p_i^s (находящимся на ярусе *lin*) и расположены на ярусах с меньшими номерами чем *lin*, будут перенесены на ярусы с номерами большими чем *lin*. Аналогичное преобразование ярусной формы может быть осуществлено и для непосредственного последователя рабочего интервала p_i^s, находящимся на ярусе *lout*.

Пусть непосредственным предшественником рабочего интервала p_i^s является рабочий интервал p_m^{lin}, находящийся на ярусе *lin*, и рабочий интервал p_i^s переносится в SP_j или изменяется его порядковый номер в этом списке. Пусть lin^* ярус, на котором находится транзитивный предшественник рабочего интервала p_m^{lin}, принадлежащий SP_j и с наибольшим порядковым номером в SP_j. Рабочие интервалы SP_j, находящихся между ярусами lin^* и *lin* могут быть перераспределены по ярусам, таким образом, что они будут находиться на ярусах с большими номерами, чем *lin* (при этом осуществляется соответствующее перераспределение по ярусам рабочих интервалов и других списков). Аналогичное преобразование ярусной формы осуществляется и для последователей.

Возможность снятия условий: каждый процесс имеет не более одного рабочего интервала или на расписание не накладывается ограничение 5

Поскольку, при доказательстве теоремы 1 и получении условий применимости операций *O1/O2* использована ярусная форма максимальной высоты (на каждом ярусе находится лишь один рабочий интервал), то полученные результаты легко могут быть обобщены на случаи когда: каждый процесс может иметь более одного рабочего

интервала или на расписание накладывается ограничение 5. При перемещении рабочего интервала из одного списка в другой, перемещаются также и все рабочие интервалы процесса, которому принадлежит перемещаемый рабочий интервал, но при этом остаются на прежних ярусах. В результате получаем ярусную форму нового расписания, и, следовательно, полученное расписание удовлетворяет ограничениям 3-5.

3.5.3. Стратегии применения операций преобразования текущего решения

Стратегия уменьшения задержек. Эта стратегия основана на следующем утверждении. Если время начала выполнения каждого рабочего интервала равно длине критического пути в графе *H* от истоков до рабочего интервала, то расписание будет оптимальным. Длина критического пути является минимально возможным временем начала выполнения рабочего интервала и равна сумме времен выполнения рабочих интервалов соответствующих вершинам критического пути. Разница между минимально возможным временем начала выполнения рабочего интервала и временем, когда рабочий интервал начал выполняться в расписании, является задержкой рабочего интервала. При применении операций *O1,O2* делается попытка уменьшить задержки рабочих интервалов.

Стратегия заполнения простоев. Эта стратегия основана на эмпирической гипотезе: чем меньше времени в сумме простаивают процессоры, тем лучше расписание. При применении операций *O1,O2* делается попытка уменьшить суммарный простой процессоров.

Смешанная стратегия. Смешанная стратегия объединяет две предыдущих. Во временной диаграмме выполнения расписания выделяются интервалы простоя процессоров и вычисляются задержки рабочих интервалов. При применении операций *O1,O2* делается попытка уменьшить задержку рабочих интервалов путем перемещения их в интервалы простоя процессоров.

Возможны следующие схемы реализации стратегий:

- случайный выбор операций и их параметров из допустимого диапазона;
- детерминированное применение операций на основе эвристических критериев;
- комбинированные схемы, сочетающие первые две.

Более подробно со стратегиями применения операций и схемами их реализации можно ознакомиться в следующих работах:

1. Калашников А.В., Костенко В.А. Параллельный алгоритм имитации отжига для построения многопроцессорных расписаний// Известия РАН. Теория и системы управления, 2008., N.3, С.133-142.

2. Калашников А.В., Костенко В.А. Итерационные алгоритмы построения расписаний, основанные на разбиении пространства решений на области// Вестн. Моск. ун-та. Сер. 15. Вычислительная математика и кибернетика. 2008., №. 3, С.56–60.
3. Д.А.Зорин, В.А.Костенко. Алгоритм синтеза архитектуры вычислительной системы реального времени с учетом требований к надежности// Известия РАН. Теория и системы управления, 2012., № 2, С.124–131.

3.5.4. Стандартные операции

В качестве закона понижения температуры используется закон описанный в разделе 3.3.

В качестве функционала $F(f(X), g_i(X))$ используется функция *T=f(HP,HW).*

В качестве критерия останова используется исходно заданное количество итераций без улучшения значения функции *T=f(HP,HW)*.

3.6. Параллельный алгоритм имитации отжига для построения статических многопроцессорных расписаний

В данном разделе рассмотрим построение параллельного алгоритма имитации отжига, основанного на декомпозиции пространства расписаний на области. Для этого множество всех возможных решений HP^*_{1-5} задачи построения расписаний представляется совокупностью областей $HP_1, HP_2, \ldots, HP_k$. Такое разбиение должно удовлетворять следующим условиям:

1) $HP_1 \cup HP_2 \cup \ldots \cup HP_k = HP^*_{1-5}$;

2) $HP_i \cap HP_j = \varnothing, \forall i, j : i \neq j$;

3) введённые в HP^*_{1-5} операции преобразования должны быть замкнутыми на областях $HP_1, HP_2, \ldots, HP_k$ и сохранять на них свойство полноты.

Определенное таким образом разбиение пространства всех решений на области позволяет искать решение в каждой из них независимо от других. Используя априорные нижние оценки времени выполнения расписания в каждой области, можно отсекать те, которые заведомо не содержат оптимального решения. Поиск решения в любой области можно осуществлять на разных узлах вычислительной системы. При этом в связи с возможным отсечением областей, в том числе в процессе поиска решений, распределение областей по узлам вычислительной системы должно обеспечивать однородную загрузку всех узлов в ходе всего времени работы алгоритма.

Таким образом, построение параллельного алгоритма требует решения следующих подзадач:

1) разбиение исходного пространства корректных расписаний на несколько непересекающихся областей, дающих в объединении все пространство;

2) выбор начального корректного расписания в каждой из областей;

3) модификация операций преобразования расписаний таким образом, чтобы модифицированные операции были замкнуты в каждой из областей и сохраняли все свойства базовых операций;

4) выбор способа распределения областей по узлам вычислительной системы и схемы отсечения областей.

3.6.1. Метрика в пространстве расписаний

Для разбиения пространства расписаний на непересекающиеся области необходимо ввести метрику на этом пространстве. Метрика в пространстве HP^* корректных расписаний будет строиться как количество элементарных операций $O=\{O_1, O_2\}$ преобразования расписания.

Пусть $S = \{s_1,\ldots,s_K\}$ - произвольная цепочка операций O_1 и O_2. Цепочка S допустима для расписания $HP \in HP^*$, если операции из этой цепочки могут быть применены к HP в заданном порядке, причем так, что все получаемые промежуточные расписания являются корректными. Обозначим: $\Omega(HP)$ - множество всех допустимых цепочек операций преобразования для расписания HP; $HP_1=S(HP)$ – расписание, полученное последовательным применением операций из цепочки S к расписанию HP. Длиной $L(S)$ цепочки S будем называть количество операций в этой цепочке.

Метрика в пространстве HP^* вводится следующим образом. Пусть HP_1, $HP_2 \in HP^*$ - два произвольных корректных расписания. Расстоянием $\rho(HP_1,HP_2)$ между расписаниями HP_1 и HP_2 будем называть число, равное длине минимальной допустимой цепочки операций O_1 и O_2, которая переводит расписание HP_1 в расписание HP_2: $\rho(HP_1,HP_2) = \min_{S\in\Omega(HP_1),S(HP_1)=HP_2} L(S)$.

Покажем корректность введенной метрики. Для этого надо доказать, что:

1) функция ρ определена всюду на HP^*;

2) для ρ выполнены аксиомы метрики.

Выполнение первого требования, очевидно, следует из теоремы 1 о полноте и замкнутости системы операций $O=\{O_1, O_2\}$ (см. раздел 3.6.2.). Покажем, что для функции ρ выполняются все свойства метрики:

Свойство 1. $\forall HP_1, HP_2 \in HP^*$: $\rho(HP_1,HP_2) \geq 0$ *и* $\rho(HP_1,HP_2) = 0 \Leftrightarrow HP_1=HP_2$.

Доказательство. Первая часть утверждения вытекает из того, что длина цепочки операций не может быть отрицательной. В совпадающих расписаниях каждый процесс имеет одинаковые значения привязки и порядка, поэтому применение любой операции делает эти расписания несовпадающими, откуда следует вторая часть утверждения.

Свойство 2. $\forall HP_1,HP_2 \in HP^*$: $\rho(HP_1,HP_2) = \rho(HP_2,HP_1)$

Доказательство. Пусть S - цепочка минимальной длины, переводящая расписание HP_1 в расписание HP_2. Тогда $\rho(HP_1,HP_2)=L(S)$. Из теоремы 1 (см. раздел 3) следует, что существует обратная допустимая последовательность $S_1 \in \Omega(HP_2)$, $S_1(HP_2)= HP_1$ такая, что $L(S_1) = L(S)$ и поэтому $\rho(HP_2,HP_1) \leq L(S_1)=L(S)= \rho(HP_1,HP_2)$. Теперь докажем, что строгое неравенство: $\rho(HP_2,HP_1)< \rho(HP_1,HP_2)$ не может выполняться. Если бы это неравенство выполнялось, то это означало бы существование цепочки операций $S^* \in \Omega(HP_2)$, $S^*(HP_2)=HP_1$, где $L(S^*) = \rho(HP_2,HP_1) < L(S)$. В этом случае должна существовать и обратная к S^* цепочка $S^*_1 \in \Omega(HP_1)$, $S^*_1(HP_1)= HP_2$, причем $L(S^*_1)=L(S^*)$. Это, в свою очередь, означает, что $L(S^*_1)<L(S)= \rho(HP_1,HP_2)$, что противоречит предположению о минимальности цепочки S. Таким образом, неравенство $\rho(HP_1,HP_2)< \rho(HP_2,HP_1)$ не может иметь места, и всегда верно равенство $\rho(HP_1,HP_2) = \rho(HP_2,HP_1)$.

Свойство 3. $\forall HP_1,HP_2,HP_3 \in HP^*$: $\rho(HP_1,HP_3) \leq \rho(HP_1,HP_2) + \rho(HP_2,HP_3)$ (неравенство треугольника).

Доказательство. Пусть S_1 - цепочка минимальной длины, переводящая расписание HP_1 в расписание HP_2, и S_2 - цепочка минимальной длины, переводящая расписание HP_2 в расписание HP_3. По определению метрики $L(S_1)= \rho(HP_1,HP_2)$ и $L(S_2)= \rho(HP_2,HP_3)$. Построим цепочку $S = S_1 + S_2$. Очевидно, что $S(HP_1)=HP_3$ и цепочка S допустима. При этом $L(S)=L(S_1)+L(S_2)$. Искомое неравенство вытекает из неравенства $\rho(HP_1,HP_3) \leq L(S)$.

Выполнение свойств 1-3 позволяет заключить, что функция $\rho(HP_1,HP_2)$ действительно является метрикой в пространстве HP^* корректных расписаний.

3.6.2. Разбиение исходного пространства решений на области

Разбиение исходного пространства решений на области достигается введением дополнительной разметки на граф модели поведения прикладной программы *H* для каждой области, которая расширяет ограничения на поведение программы. В качестве исходной области берется все пространство расписаний, разбиваемое на три непересекающиеся подобласти. Для этого фиксируются два произвольных рабочих интервала, не связанных транзитивным отношением порядка:

- в первой области эти рабочие интервалы распределены на разные процессоры;
- во второй рабочие интервалы обязаны выполняться на одном процессоре, причем второй рабочий интервал выполняется после первого;
- в третьей области рабочие интервалы обязаны выполняться на одном процессоре, причем первый рабочий интервал следует после второго.

На рис. 3.5. схематично изображен принцип разбиения пространства расписаний на области. На первом этапе выбираются два рабочих интервала (2 и 4), не связанные отношением порядка, затем пространство решений разделяется на три области, как показано на рисунке. Каждой из областей HP_i, образующих пространство HP^*_{1-5}, можно поставить в соответствие граф $H_i = (P, \prec \cup \prec', J, K)$. P - множество вершин, соответствующих рабочим интервалам. Дугам $\prec \cup \prec'$ отвечают связи, определяющие взаимодействия между рабочими интервалами. Все связи $\prec$ присутствующие в H сохраняются в H_i. Отношение $\prec'$ задает дуги, устанавливающие дополнительные ограничения на порядок выполнения рабочих интервалов в каждой области. Отношение $\prec \cup \prec'$ транзитивно и ациклично. Отношения J и K соответствуют ограничениям на привязку рабочих интервалов:

- два рабочих интервала p_i и p_j связаны отношением J, $(p_i, p_j) \in J$, если они должны выполняться на одном процессоре;
- два рабочих интервала p_i и p_j связаны отношением K, $(p_i, p_j) \in K$, когда они распределены на разные процессоры.

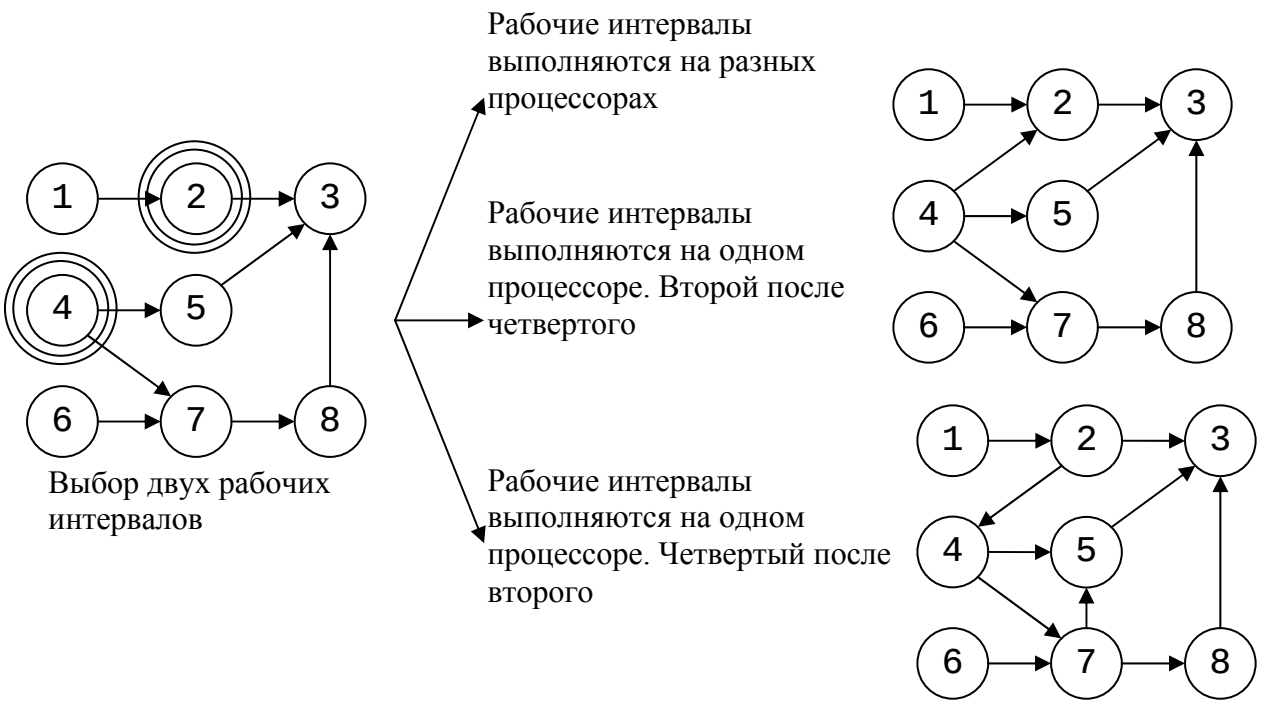

Рис 3.5. Принцип разбиения пространства расписаний на области.

.

Отношения J и K симметричны и $J \cap K = \varnothing$, а J транзитивно. Все метки, присутствующие в графе H, сохраняются в графе H_i. Расписание HP является корректным в области HP_i, если выполнены условия:

1) каждый рабочий интервал должен быть назначен на процессор;

2) любой рабочий интервал назначен лишь на один процессор;

3) частичный порядок $\prec \cup \prec'$, заданный в HP_i сохранен в HP;

4) расписание HP должно быть беступиковым. Условием беступиковости является отсутствие циклов в графе HP при неограниченном размере буферов обмена;

5) отношение J должно быть сохранено в HP: любые два рабочих интервала, связанные отношением J, должны быть назначены на один и тот же процессор;

6) отношение K сохраняется в HP: любые два рабочих интервала, связанные отношением K, должны быть назначены на разные процессоры.

Алгоритм разбиения исходного пространства решений на области. Введём следующие обозначения: P^p_{pred} и $\prec^p_{pred}$ – множества вершин и дуг графа H соответствующие рабочим интервалам, которые предшествуют рабочему интервалу p, в том числе и транзитивно, P^p_{anc} и $\prec^p_{anc}$ – множества вершин и дуг графа, отвечающие рабочим интервалам, которые следуют за рабочим интервалом p, в том числе и

транзитивно, $P_{\prec}^{p}$ - множество рабочих интервалов, не связанных с рабочим интервалом *p* отношением порядка $\prec \cup \prec'$, в том числе и транзитивно. Для разбиения пространства на подобласти используется алгоритм, состоящий из следующих шагов.

Ш а г 1. Задать количество областей разбиения.

Ш а г 2. В список графов поместить граф, соответствующий всему пространству расписаний: $H = (P, \prec, \varnothing, \varnothing)$.

Ш а г 3. Выбрать первый граф $\tilde{H} = (P, \tilde{\prec}, \tilde{J}, \tilde{K})$ из списка и построить три графа для трех непересекающихся областей следующим образом.

Ш а г 3.1. $\hat{P} = P$, где $\hat{P}$ - временное множество рабочих интервалов. В начале каждой итерации оно совпадает с множеством *P* всех рабочих интервалов.

Ш а г 3.2. Выбрать из $\hat{P}$ рабочий интервал p_i, для которого критический пусть в графе $(P_{pred}^{p_i}, \prec_{pred}^{p_i}, \tilde{J}, \tilde{K})$ минимален и удалить p_i из множества $\hat{P}$. Если $\hat{P}$ пусто, закончить разбиение, сообщив о невозможности его продолжения.

Ш а г 3.3. Для выбранного рабочего интервала p_i построить множество $P_{\prec}^{p_i}$. Если оно пусто, перейти к шагу 3.2 и рассмотреть следующий рабочий интервал из $\hat{P}$. Если $P_{\prec}^{p_i}$ не пусто, то использовать из него рабочий интервал p_j, соответствующий минимальному критическому пути в графе $(P_{anc}^{p_j}, \prec_{anc}^{p_j}, \tilde{J}, \tilde{K})$.

Ш а г 3.4. Построить графы H_1, H_2, H_3 для областей HP_1, HP_2, HP_3:

$$H_1 = (P, \tilde{\prec} \cup (p_i, p_j), \tilde{J} \cup (p_i, p_j) \cup (p_j, p_i), \tilde{K})$$

$$H_2 = (P, \tilde{\prec} \cup (p_j, p_i), \tilde{J} \cup (p_i, p_j) \cup (p_j, p_i), \tilde{K})$$

$$H_3 = (P, \tilde{\prec}, \tilde{J}, \tilde{K} \cup (p_i, p_j) \cup (p_j, p_i))$$

Ш а г 4. Удалить граф $\tilde{H}$ из списка и добавить три построенные графа H_1, H_2, H_3 в его конец. Если количество графов в списке меньше заданного количества областей разбиения, перейти к шагу 3.

Данный алгоритм позволяет так осуществлять разбиение на области, что критические пути в графах областей разбиения будут существенно отличаться между собой за счёт выбора на шагах 3.2 и 3.3 рабочих интервалов, отстоящих как можно “дальше” друг от друга: первый рабочий интервал имеет малое количество предшественников, второй является предшественником малого количества рабочих интервалов. Это позволяет отбросить большое количество областей (без запуска в них

алгоритма имитации отжига), в которых критический путь больше времени выполнения расписаний, полученных при запуске алгоритма имитации отжига в других областях.

3.6.3. Операции преобразования расписания внутри области

Алгоритмы выполнения операций O_1 и O_2 должны обеспечивать их замкнутость внутри области.

Введём следующие обозначения: $I_1 = \{p_i \in P \mid (p_i, p) \in J\}$ – множество рабочих интервалов, которые должны выполняться на одном процессоре с рабочим интервалом *p*; $I_2 = \bigcup_{p_i \in I_1} \{p_j \in P \mid (p_j, p_i) \in K\}$ - множество рабочих интервалов, для которых запрещено выполнение на одном процессоре с рабочим интервалом *p*; *SP* – множество всех процессоров; SP_{HP}^{p} - процессор, на который распределён рабочий интервал *p* в расписании *HP*.

Алгоритм выполнения операции O_1 включается следующие шаги:

Ш а г 1. Случайным образом выбирается рабочий интервал *p*.

Ш а г 2. Строится множество процессоров SP', на которые возможно перенести рабочий интервал *p*: $SP' = SP \setminus \bigcup_{p_k \in I_2} SP_{HP}^{p_k}$.

Ш а г 3. Случайным образом выбирается процессор из SP' и на него переносятся все рабочие интервалы из множества I_1.

Алгоритм выполнения операции O_2 включается следующие шаги:

Ш а г 1. Случайным образом выбирается рабочий интервал *p*.

Ш а г 2. Определяется допустимый диапазон ярусов $[L_{low}+1, L_{high}-1]$. Этот диапазон выбирается таким образом, что рабочий интервал *p* может быть перенесён на любой ярус из найденного диапазона.

Ш а г 3. Ярус из допустимого диапазона выбирается случайным образом и на него переносится рабочий интервал *p*.

Сложность алгоритмов применения операций O_1 и O_2 равна *O(N)*. Так как операции преобразования в области совпадают с операциями преобразования на всём пространстве решений, но применяются к графам с дополнительными дугами, то сохраняется свойство полноты и замкнутости для области.

3.6.4. Распределение областей по узлам вычислительной системы

Распределение областей по узлам вычислительной системы осуществляется так, чтобы обеспечить максимально равномерную загрузку процессоров в течении всего времени работы параллельного алгоритма имитации отжига. В процессе работы алгоритма области, графы которых имеют критические пути большие, чем время выполнения наилучшего расписания, полученного к данному моменту, исключаются из дальнейшего рассмотрения. Таким образом, необходимо так распределить области по узлам вычислительной системы, чтобы в ходе работы алгоритма не возникали ситуации, когда на одном из узлов количество рассматриваемых областей существенно больше, чем на другом. Пусть n – количество областей разбиения, а m – количество узлов вычислительной системы, осуществляющих поиск решений в областях $(n > m)$. Области упорядочиваются по критическому пути в графах. Тогда в i-й узел $(1 \le i \le m)$ будут распределены области с номерами $j : j \bmod m = i \ (1 \le j \le n)$. (рис. 3.6.).

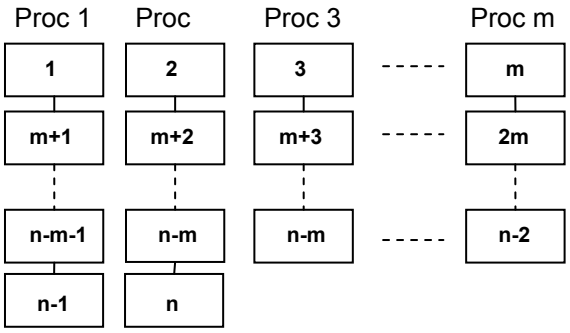

Рис. 3.6. Распределение областей по узлам вычислительной системы.

Для параллельной работы алгоритма и отсечения областей используется следующая стратегия:

1) каждый узел осуществляет поиск решений в областях, начиная просматривать области с наименьшим критическим путем;

2) в каждой области выполняется фиксированное число итераций. После этого каждый узел исключает из дальнейшего рассмотрения области с критическим путем, большим, чем время выполнения наилучшего расписания, полученного этим узлом;

3) если было отсечено заданное количество областей, то узел инициирует обмен, передавая время наилучшего расписания остальным узлам, которые в свою очередь выполняют отсечение областей с учетом переданного им времени;

4) узел производит отсечение области, если в течение заданного числа итераций в данной области не произошло уменьшение целевой функции (времени выполнения расписания). При этом отсечение областей остальными узлами не инициируется;

5) узел заканчивает поиск, если в течение заданного числа итераций не произошло уменьшение целевой функции (время выполнения расписания) или, если все области оказались отсечены, или превышено допустимое число итераций. В качестве итогового расписания выбирается наилучшее среди расписаний, полученных в областях разбиения в результате работы всех узлов.

3.7. Сравнение эффективности алгоритмов

Рассмотрим классический последовательный алгоритм имитации отжига, последовательный алгоритм, использующий разбиение на области, и параллельный алгоритм.

Для исследования использовались графы модели прикладной программы со следующими характеристиками: количество рабочих интервалов от 100 до 250, число процессов от 40 до 90.

По степени связности были выделены три класса графов:

малое количество связей - $K / N \le 1.2$;

среднее количество связей – $1.2 < K / N < 1.6$;

большое количество связей – $K / N \ge 1.6$.

Здесь K – число дуг в графе, N - число вершин.

Было проведено сравнительное исследование времени, затрачиваемого на поиск решения одного и того же качества классическим последовательным алгоритмом, последовательным алгоритмом, реализующим разбиение на области и параллельным алгоритмом. Были получены следующие результаты.

На одном узле, без обмена данными с другими узлами возможно отсечение до 25% областей (рис. 3.7.), ещё до 20% от оставшихся областей исключается после обмена данных между узлами (рис. 3.8.). На рис. 3.9. показано количество отсеченных областей, в зависимости от количества итераций алгоритмов в этих областях.

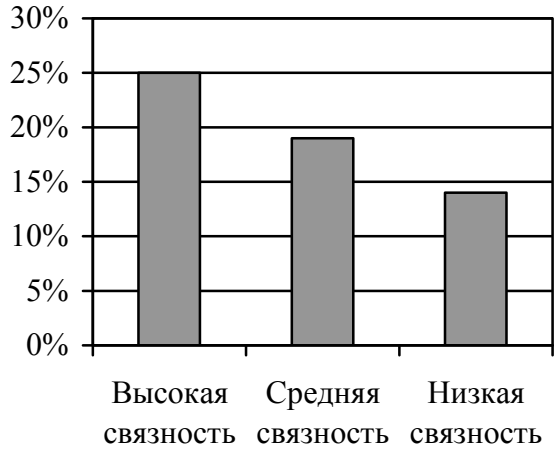

Рис. 3.7. Отсечение областей на одном узле без обмена с другими узлами.

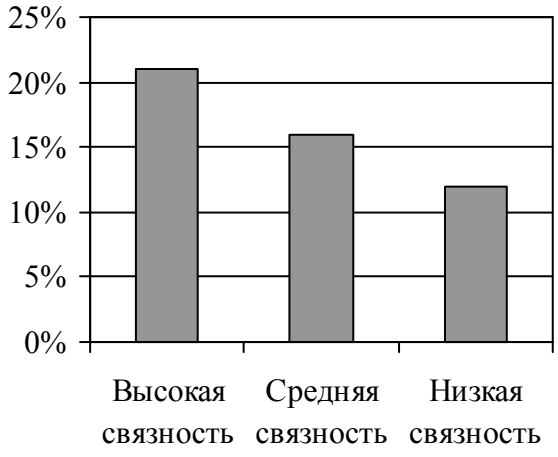

Рис. 3.8. Отсечение областей в результате обмена с другими узлами.

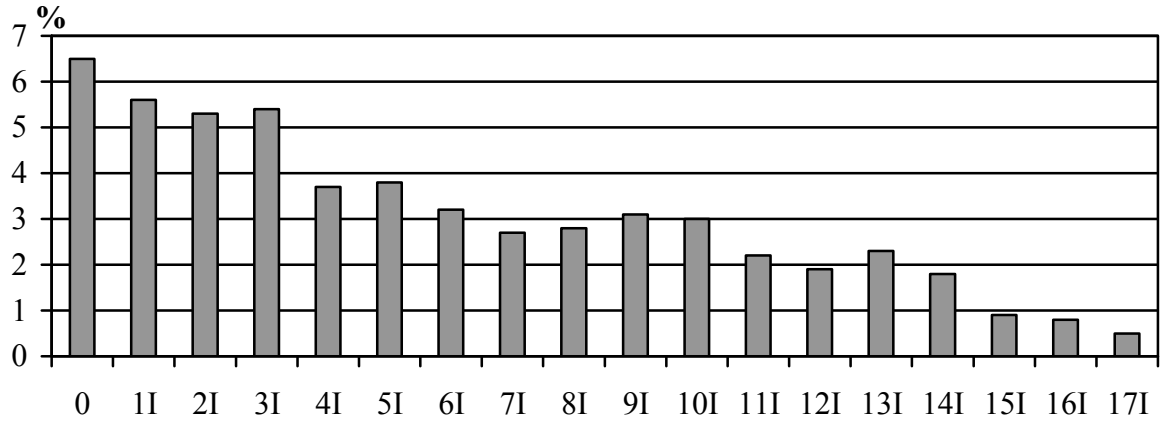

Рис 3.9. Зависимость количества отсеченных областей от количества итераций алгоритмов запущенных в этих областях. $I = 1000$.

Последовательный алгоритм, использующий разбиение на области по сравнению с классическим делает в 2-3 раза меньше итераций для графов с высокой связностью. Для графов со средней и низкой связностью этот показатель существенно ниже.

В среднем параллельный алгоритм, запущенный на четырех процессорах, получал решение в 2.21 раза быстрее, чем последовательный, применяющий разбиение на области; на восьми процессорах параллельный алгоритм осуществлял поиск решения в 3.34 раза быстрее, чем последовательный. Предел увеличения быстродействия параллельного алгоритма связан с тем, что около 20% операций (разбиение на области) выполняются на одном узле последовательно. Однако следует заметить, что операция разбиения на области также может быть распараллелена, что позволит достичь лучшего отношения времен работы последовательного и параллельного алгоритмов. В среднем увеличение скорости выполнения параллельного алгоритма по сравнению с последовательным не зависит от параметров входного графа.

Время работы алгоритмов, использующих разбиение на области, увеличивается с уменьшением связности графа (рис. 3.10.). Это обусловлено тем, что с уменьшением связности графа, уменьшается процент областей, отсекаемых алгоритмом в ходе работы. На графах с высокой связностью критический путь близок к времени выполнения расписания, получаемого алгоритмом, и большинство областей с высокими критическими путями исключается из рассмотрения.

На рис. 3.10. приведены обобщённые результаты исследования. За 100% взято наибольшее зафиксированное время работы алгоритмов.

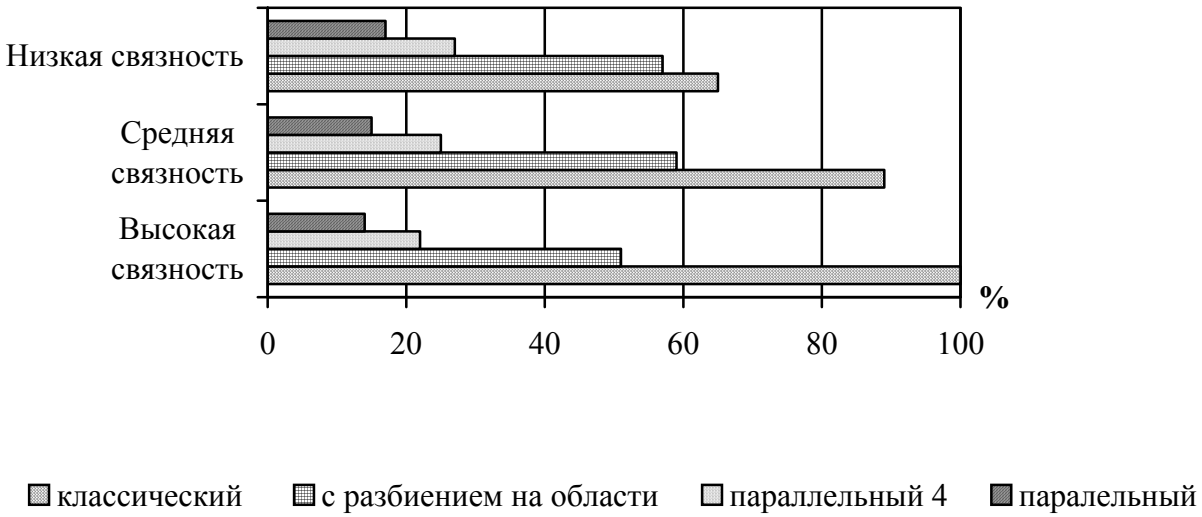

Рис. 3.10. Сравнение времени работы алгоритмов.

Также было проведено сравнительное исследование качества решений при одинаковом времени работы алгоритмов. Получены следующие результаты. Параллельный алгоритм, работающий на четырех процессорах, в среднем получает расписания с временем выполнения на 1.8% меньшим, чем последовательный, и на 2.7% меньшим, чем классический. Наибольшее уменьшение времени выполнения построенного расписания наблюдается на графах с низкой связностью и составляет в среднем 3.8% по сравнению с последовательным и 5.1% по сравнению с классическим. Это обусловлено тем, что на графах с низкой степенью связности время выполнения оптимального расписания существенно больше критического пути в графе и за малое время классический и последовательный алгоритмы не успевают довести расписание до «хорошего» локального минимума. На рис. 3.11. приведены обобщённые результаты исследования.

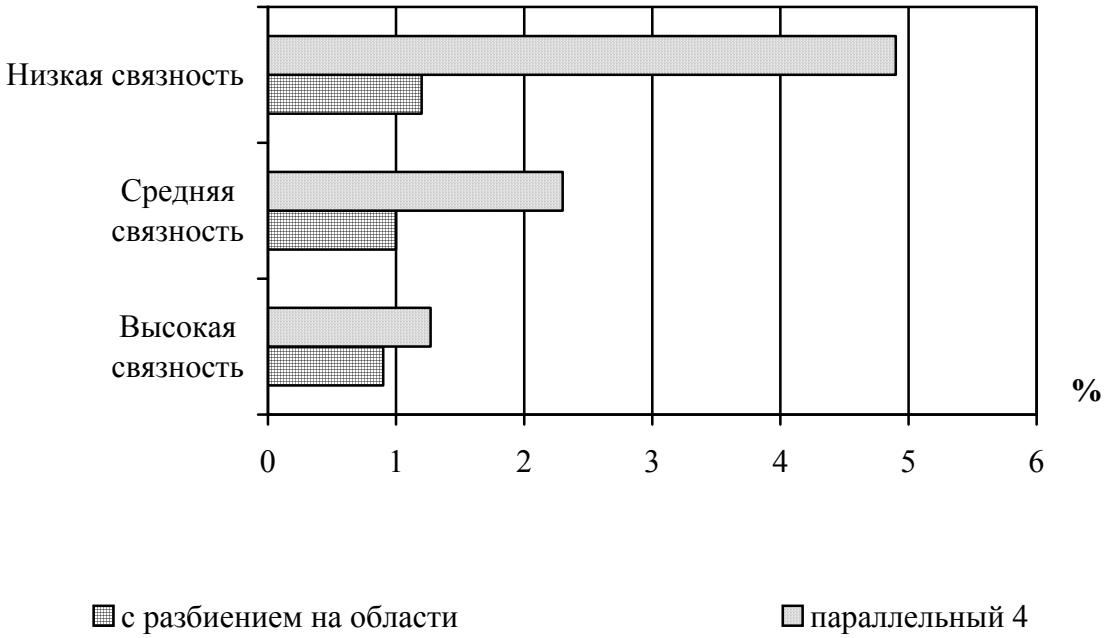

Рис. 3.11. Улучшение качества решений при использовании разбиения на области и параллельного алгоритма по сравнению с решениями, полученными классическим алгоритмом.

4. Генетические и эволюционные алгоритмы

4.1.Простой генетический алгоритм (алгоритм Холланда)

Генетические и эволюционные алгоритмы основаны на использовании механизмов естественной эволюции. Эволюция, по Дарвину [Charles Darwin. The Origin of Species. John Murray, London, 1859.], осуществляется в результате взаимодействия трех основных факторов: изменчивости, наследственности, естественного отбора. Изменчивость служит основой образования новых признаков и особенностей в строении и функциях организма. Наследственность закрепляет эти признаки. Естественный отбор устраняет организмы, плохо приспособленные к условиям существования.

Генетические и эволюционные вычисления получили общее признание после выхода книги Холланда "Адаптация в естественных и искусственных системах" [Holland J.N. Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan: Univ. of Michigan Press, 1975.]. Предложенная в данной работе концепция построения генетических алгоритмов оказалась чрезвычайно эффективной для решения многих задач, не поддающихся решению традиционными методами. Концепцию построения алгоритмов, предлагаемую Холландом, схематично можно представить следующим образом:

1. *Сгенерировать случайным образом популяцию размера P.*
2. *Выполнить операцию скрещивания.*
3. *Выполнить операцию мутации.*
4. *Вычислить функцию выживаемости для каждой строки популяции.*
5. *Выполнить операцию селекции.*
6. *Если критерий останова не достигнут, перейти к шагу 2, иначе завершить работу.*

Популяция - это множество битовых строк. *Каждая строка* представляет в закодированном виде одно из возможных решений задачи. По строке может быть вычислена *функция выживаемости*, которая характеризует качество решения. В качестве начальной популяции может быть использован произвольный набор строк. Основные операции алгоритма: селекция, скрещивание и мутация выполняются над элементами популяции. Результатом их выполнения является очередная популяция. Данный процесс продолжается итерационно до тех пор, пока не будет, достигнут критерий останова.

Кодирование решений. Способ кодирования решений должен обладать следующими свойствами:

1. Однозначность, т.е. каждая закодированная строка должна соответствовать единственному решению исходной задачи. Выполнение данного критерия необходимо для статистической прогнозируемости поведения алгоритма и улучшения нахождения алгоритмом областей оптимальных решений.
2. Возможность кодирования любого допустимого решения.
3. Получение в результате генетических операций корректных вариантов решений.
4. Возможность перехода от любого корректного решения к любому другому корректному решению.

Универсальные подходы к кодированию решений очень часто или невозможны или не эффективны для решения сложных задач условной оптимизации. Однако, для задач непрерывного и целочисленного математического программирования, наиболее часто оптимизируемые параметры задаются, или двоичным кодом числа, или кодами Грея. Битовая строка получается склейкой битовых полей параметров.

Операция селекции обеспечивает формирование на очередной итерации алгоритма из строк, полученных на шагах 2 и 3, новой популяции. Операция должна удовлетворять следующим требованиям:

1. Выделять каждой строке количество потомков в соответствии со значением её функции выживаемости, т.е. если $f(x_1)>f(x_2)$, то вероятность того, что $Ch(x_1)\geq Ch(x_2)$ должна быть достаточно велика. Здесь f – целевая функция, x_1 и x_2 – строки популяции, $Ch(x)$ – количество потомков, присваиваемых строке x;
2. Обеспечивать сохранение в популяции лучших решений, т.е. решения, имеющие максимальное значение целевой функции, должны сохраняться в популяции, в противном случае алгоритм может утрачивать найденные хорошие значения;
3. Не допускать обеднения и вырождения популяции, т.е. недопустимы популяции, в которых одно решение с высокой целевой функцией заполняет всю популяции, так как в этом случае алгоритм утрачивает способность поиска по всему пространству решений и скатывается к локальному оптимуму.

Приведем два классических варианта выполнения операции: схема пропорциональной селекции и схема рулетки.

Схема пропорциональной селекции выглядит следующим образом:

1. Вычислить среднее значение функции выживаемости $\overline{F}$ для популяции.

2. Для *i*-ой строки популяции выделить $\left\lfloor F_i \middle/ \overline{F} \right\rfloor$ потомков, где F_i – значение функции выживаемости *i*-ой строки.
3. Из полученных строк сформировать новую популяцию.

Схема рулетки работает следующим образом:

1. Вычислить среднее значение функции выживаемости для популяции.
2. Для *i*-ой строки популяции выделить сектор рулетки с центральным углом $2\pi F_i \middle/ \overline{F}$, где F_i – значение функции выживаемости *i*-ой строки.
3. Сделать *P* запусков рулетки, где *P* – размер популяции. Каждый раз выделяем строке в чей сектор мы попали 1-го потомка.
4. Из полученных строк сформировать новую популяцию.

Схема рулетки может давать большие ошибки, в том смысле, что конечное число потомков данной строки может сильно отличаться от ожидаемого числа потомков. Конечное число приближается к ожидаемому числу только в популяциях очень больших размеров.

Операция скрещивания (рис)заключается в выполнении следующих действий: 1)выбрать пары строк для скрещивания; 2)для каждой выбранной пары: с заданной вероятностью выполнить скрещивание, получить двух потомков и произвести в популяции замену родителей на их потомков или просто добавить в популяцию двух потомков. Параметр операции - вероятность скрещивания (p_c). Простейший вариант операции (одноточечное скрещивание) выполняется следующим образом:

1. Вся популяция случайным образом разбивается на пары.
2. Для каждой пары случайным образом генерируется число p'_c, если $p'_c < p_c$, то выбирается случайное целое число *i* в интервале *(1*, *l-1)*, где *l* - длина строки, и строки обмениваются фрагментами, находящимися после *i*-го бита, в противном случае ничего не происходит.

При многоточечном скрещивании выбирается несколько точек разрыва строки.

Операция мутации (рис) заключается в инвертировании каждого бита с заданной вероятностью**.** Параметр операции - вероятность мутации (p_m). Операция выполняется следующим образом:

1. Для каждого бита генерируется случайное число p'_m.
2. Если $p'_m < p_m$, то бит инвертируется.

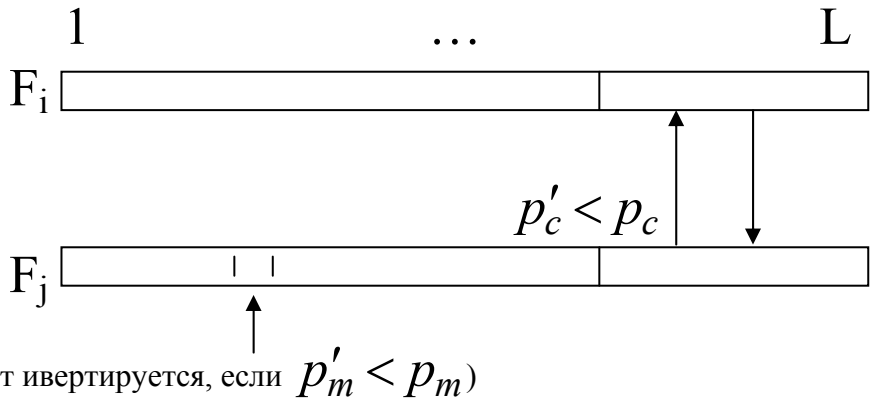

Рис. Схема выполнения операций мутации и скрещивания

Критерий останова. В большинстве алгоритмов используется один или некоторая комбинация следующих способов останова:

- выполнение алгоритмом априорно заданного числа итераций;
- выполнение алгоритмом априорно заданного числа итераций без улучшения функции выживаемости;
- достижение некоторого априорно заданного значения функции выживаемости.

Эволюционные алгоритмы используют целочисленное или вещественное кодирование решений. Операции выполнения операций скрещивания и мутации для этих способов кодирования решений приведены в приложении 1.

Среди основных проблем использования ГА и ЭА можно выделить следующие:

1. Выбор способа кодирования решений.
2. Определение операций скрещивания и мутации для работы с используемым представлением решения.
3. Определение параметров алгоритма:
   - размера популяции;
   - вероятностей скрещивания и мутации.
4. Задание целевой функции и критерия останова.

4.2. Теория схем. Гипотеза строительных блоков

Несмотря на успешное использование ГА, до сих пор нет полной ясности как работает ГА. Теория схем и гипотеза строительных блоков, предложенные Холландом и Голдбергом [(Holland J.N. Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan: Univ. of Michigan Press, 1975.), (Golberg D.E. Genetic Algorithms in Search, Optimization and Machine Learning.

Addison-Wesley, Reading, Mass., 1989.)], отражают основные механизмы работы ГА, объясняют: почему все-таки ГА могут работать.

Схема - представляет подмножество всех возможных строк, которые имеют те же самые биты в некоторых фиксированных позициях. Схема **000 представляет все строки с 0 в последних трёх позициях: 00000, 01000, 10000 и 11000. Подобным же образом схема 1*00* представляет строки: 10000, 10001, 11000 и 11001. Каждая строка представленная схемой называется *примером схемы*. Количество фиксированных позиций в схеме - это её *порядок* (**000 имеет 3 порядок). *Определяющая длина* схемы - это расстояние между самыми дальними фиксированными позициями. Так у схемы **000 определяющая длина равна 2, у 1*00* - равна 3. Каждая строка одновременно является примером в 2^l схем (l - длина строки). Так как схема определяет набор строк, то можно ввести понятие *среднего значения целевой функции схемы*. В очередной популяции оно определяется средним значением целевых функций примеров схемы в популяции. Среднее значение целевой функции схемы варьируется вместе с составом популяции на различных итерациях алгоритма.

Представим схему с k фиксированными позициями. Существует 2^k-1 других схем с теми же самыми фиксированными позициями, которые могут быть получены, перестановками 0 и 1 в этих k позициях. Каждый такой набор фиксированных позиций образует соревнование схем, борьбу за выживание среди 2^k схем. Так как существует 2^l возможных комбинаций фиксированных позиций - следовательно, возможно 2^l различных соревнований. ГА одновременно пытается решить 2^l соревнований схем и выделить лучшую схему для каждого набора фиксированных позиций. Мы можем представить себе поиск оптимального решения как одновременное соревнование между схемами за увеличение количества их примеров в популяции.

Следующее уравнение известно как теорема схем:

$$N(h,t+1) \geq N(h,t)\frac{F(h,t)}{\overline{F}(t)}\left[1 - p_c \frac{\delta(h)}{l-1} - p_m o(h)\right],$$

где $N(h,t+1)$ - ожидаемое количество примеров схемы h на шаге $t+1$, $F(h,t)$ - среднее значение целевой функции схемы h на шаге t, $\overline{F}(t)$ - средние значение целевой функции в популяции на шаге t, p_c - вероятность скрещивания, $\delta(h)$ - определяющая длина схемы h, p_m - вероятность мутации, $o(h)$ - порядок схемы h, l - количество бит в строке.

Выражение $N(h,t)\frac{f(h,t)}{\bar{f}(t)}$ определяет ожидаемое число примеров схемы *h* в новой популяции, выражение $\left[1-p_c\frac{\delta(h)}{l-1}-p_m o(h)\right]$ определяет вероятность выживания схемы *h* после выполнения операций скрещивания и мутации, слагаемое $p_c\frac{\delta(h)}{l-1}$ определяет вероятность того, что пример схемы *h* будет разрушен скрещиванием, слагаемое $p_m o(h)$ определяет вероятность того, что пример будет разрушен мутацией.

Данная теорема описывает несколько важных аспектов поведения ГА. Мутация с большей вероятностью разрушает схемы высокого порядка, а скрещивание - схемы с большой определяющей длиной. Селекция обеспечивает сходимость популяции пропорционально мере давления селекции - отношение значения целевой функции лучшей строки к среднему значению целевой функции в популяции. Увеличение p_c, p_m или уменьшение меры давления селекции увеличивает разнообразие популяции, но не позволяет использовать все хорошие схемы, имеющиеся в популяции. Уменьшение p_c, p_m или увеличение меры давления селекции приводит к улучшению использования найденных схем, но замедляет исследование пространства решений в поисках новых хороших схем. Поддержание равновесия известно как проблема “баланса исследования и использования”. Проблема выбора параметров ГА (вероятность скрещивания и мутации, размер популяции и т.п.) является для многих приложений сложной задачей, так как параметры не могут быть определены независимо, без учета взаимодействия операций, способа кодирования и размера популяции. Какие-либо теоретические результаты для решения данной проблемы (за исключением качественного анализа работы ГА) на настоящий момент времени отсутствуют.

Если мы опишем оптимальную строку в виде комбинации схем с маленькими определяющими длинами, низким порядком и высокими значениями средних целевых функций, тогда победители индивидуальных соревнований схем потенциально могут сформировать оптимальную строку (это не всегда справедливо, очень плохие строки могут быть созданы из хороших схем). Схемы с короткими определяющими длинами, низким порядком и высокими значениями средних целевых функций - *строительные блоки*. *Гипотеза строительных блоков* утверждает, что комбинирование хороших строительных блоков даёт хорошую строку. Операции скрещивание и мутация создают, улучшают и комбинируют строительные блоки таким образом, чтобы получить оптимальную строку. Скрещивание старается сохранить генетическую информацию, имеющуюся в скрещиваемых строках. Мутация является не консервативной операцией и

может создавать совершенно новые строительные блоки. Селекция обеспечивает увеличение в популяции количества примеров строительных блоков с высокими значениями целевых функций.

Гипотеза строительных блоков и теорема схем дают качественную картину того, как ГА мог бы работать. Из этих результатов следует важность выбора способа кодирования и значений параметров операций мутации, скрещивания и селекции. При “неудачном” выборе способа кодирования и значений параметров операций алгоритм работает как алгоритм ненаправленного случайного поиска. Каких либо конструктивных теоретических результатов, позволяющие разработать методики решения этих проблем неизвестно. На примере описания динамики работы ГА с помощью цепей Маркова и подхода к анализу динамики поведения ГА на макроскопическом уроне, что препятствует созданию методик выбора способа кодирования и значений параметров операций при построении ГА для решения практических задач условной оптимизации. В большинстве случаев проблема выбора способа кодирования и значений параметров операций решается в настоящее время путем формирования эвристических гипотез и проверке их экспериментальным путем.

Описание динамики работы ГА с помощью цепей Маркова. При описании динамики работы ГА с помощью цепей Маркова проблема размерности и размера модели становится сдерживающей. Данный подход предполагает [(Vose M.D., Liepins G.E. Complex Systems, 1991, v.5, p.31.), (Nix A., Vose M.D. Ann. Math. and Artificial Intelligence, 1991, v.5, N 79, p.88.)], что строится система детерминированных уравнений, которая определяет вероятность того, что каждая конкретная строка будет присутствовать в популяции в момент времени t с учетом вероятностей для момента времени $t-1$. Динамика работы ГА может быть описана вероятностями появления каждой строки (вектором размера 2^l на каждой итерации ГА). Поскольку мутация является линейной операцией, она описывается матрицей размера $2^l \times 2^l$, операция скрещивания является двухместной и описывается тензором размера $2^l \times 2^l \times 2^l$. Следует также отметить, что Марковские модели не учитывают конечный размер популяции и предполагают выбор родителей в операции скрещивания лишь в соответствии с равновероятным выбором.

Макроскопический уровень описания динамики работы ГА. Переход на макроскопический уровень описания динамики работы ГА заключается в описании ГА в терминах статистических свойств популяции, т.е. вместо описания каждой возможной строки, описываются выделенные статистические свойства, характеризующие популяцию в целом [(Golberg D.E., Deb K., Clark J.N. Complex Systems, 1992, v.6, p.333.),(Bulmer M.G. Mathematical Theory of Quantitative Genetics. Cambridge: Claredon Press, 1980.)]. Таким образом, динамика системы

с огромным числом степеней свободы сводится к динамике системы всего с несколькими параметрами, которая может быть легко рассчитана или смоделирована. Этот подход не будет исследовать свойств первичных элементов популяции, что затрудняет оценку его точности. При использовании данного подхода возникают также нетривиальные задачи правильного выбора макропараметров (позволяющих потерять как можно меньше информации о популяции) и оценки влияния операций ГА на эти макропараметры.

Известен подход к анализу динамики поведения ГА, основанный на принципах *статистической механики* [Шапиро Дж., Рэттрэй М., Прюгель-Беннетт А. Применение методов статистической механики для изучения динамики генетического алгоритма// Обозрение прикладной математики. Серия "Методы оптимизации". Эволюционные вычисления и генетические алгоритмы, 1996.,Т.3, выпуск 5. - С.670-687.]. Он позволяет изучать динамику поведения ГА при решении простейших комбинаторных задач (цитируя авторов: "игрушечных задач"). Тип допустимой целевой функции в этих задачах жестко ограничен. Целевая функция является функцией от величины h^α: $F^\alpha=F(h^\alpha)$, где $h^\alpha = \sum_{i=1}^{l} w_i x_i^\alpha$, w_i - фиксированные весовые коэффициенты, x_i^α - i-й бит строки с номером α в очередной популяции. Этот подход основан на использовании статистической механики для расчета ожидаемых результатов (значений макропараметров: четыре семиинварианта h^α и корреляции между строками - меры сходства строк) применения операций ГА, а также для расчета их отклонений от ожидаемых значений. Для описания макропараметра h^α используются четыре первых семиинварианта: матожидание, дисперсия, семиинварианты, связанные с асимметрией и эксцессом. Все они усредняются по строкам текущей популяции, а не по всем допустимым строкам. То есть, они будут изменяться во времени вместе с популяцией. Если требуется информация, отличная от той, которая содержится в макропараметрах, то она может быть получена из макропараметров методом максимума энтропии. Данный подход может оказаться наиболее перспективным для анализа динамики поведения ГА при решении задач комбинаторной оптимизации, если он будет расширен для задач, в которых на целевую функцию не накладываются столь жесткие ограничения.

4.3. Модификации генетических алгоритмов

4.3.1. Алгоритмы, использующие функции значимости фрагментов

Данный подход к построению генетических алгоритмов предполагает выделение фрагментов решения и введение для них функций значимости. Значение функции значимости фрагмента является оценкой вклада фрагмента в значение функции

выживаемости. Значения параметров операций мутации и скрещивания фрагментов определяются в зависимости от значений функций значимости фрагментов решения, т.е. в этом алгоритме вероятности мутации и скрещивания различных фрагментов различны. Скрещивание старается сохранить фрагменты решения с высоким значением функции значимости, т.е. операция скрещивания является консервативной, что необходимо для принципиальной работоспособности генетического алгоритма. Мутация является не консервативной операцией и позволяет изменять фрагменты решения. Сказанное выше позволяет избежать решения проблемы выбора такой кодировки решения и выбора значений параметров операций мутации и скрещивания, что бы могли существовать строительные блоки. Напомним, что генетический алгоритм может работать, только если в ходе работы алгоритма образуются строительные блоки. В данном разделе покажем применение этого подхода для построения алгоритмов распознавания участков фазовых траекторий динамических систем по обучающей выборке с помощью генетических алгоритмов.

Задача распознавания участков фазовых траекторий динамических систем

Рассмотрим систему, которая окружена набором датчиков. Будем считать, что датчики, окружающие систему, опрашиваются с одинаковой постоянной частотой. Фазовая траектория X в пространстве показаний датчиков представляет собой последовательные измерения всех датчиков системы:

$$X = (\bar{x}_1, \bar{x}_2, ..., \bar{x}_n)$$

где $\bar{x}_i = \bar{x}(t_0 + i \cdot \tau)$ – это точка в многомерном пространстве показаний датчиков (в фазовой плоскости); $\frac{1}{\tau}$ – частота опроса датчиков.

Состояние системы характеризуется показаниями датчиков, окружающих ее. Со временем система может изменять свое состояние. Последовательные изменения состояния системы будем называть ее поведением. Состояния системы можно разделить на два базовых типа:

- Штатное состояние. Это такое состояние наблюдаемой системы, при котором она стабильно выполняет заложенные в нее функции.
- Аварийное состояние. Это такое состояние системы, при котором она не выполняет часть заложенных в нее функций или в скором времени гарантированно перестанет их выполнять.

Поведение системы, которое переводит ее из штатного состояния в аварийное, будем называть нештатным (или аварийным) поведением. Может существовать несколько классов нештатного поведения, приводящих к аварийному состоянию системы.

Будем считать, что каждому классу нештатного поведения соответствует некоторая характерная траектория X_{Anom}, такие траектории будем называть эталонными.

Пусть число классов нештатного поведения системы равно L. Обозначим: $W = \{w\}_{w=1}^{w=L} \cup \{0\}$ – множество ответов, где 0 – соответствует штатному поведению системы, w – соответствует нештатному поведению под номером w из L возможных.

Участки траекторий, соответствующие различным классам нештатного поведения, могут входить в анализируемую траекторию X в искаженном относительно эталонных траекторий виде. Можно выделить следующие типы искажений:

- искажения по амплитуде,
- искажения по времени,
- наложение шума.

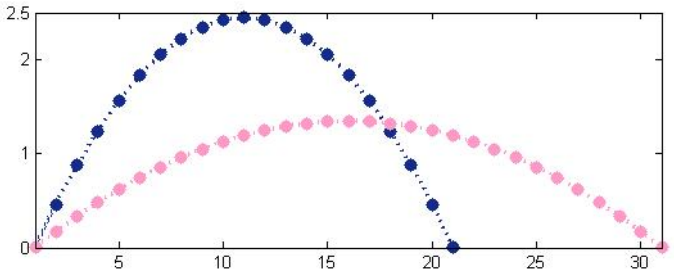

Рисунок 4.1. Пример искаженных траекторий.

Под искажением траектории по амплитуде будем понимать изменение абсолютных значений точек траектории, без изменения числа отсчетов. Под искажением траектории по времени будем понимать изменение числа отсчетов, на которых определена траектория, то есть добавление в траекторию новых отсчетов или удаление из нее уже существующих. На рисунке показаны траектории, которые искажены по амплитуде и времени друг относительно друга.

Задача распознавания нештатного поведения состоит в следующем:

Дано:

- Наблюдаемая многомерная траектория X.
- Набор из L классов нештатного поведения системы, для каждого из которых задана эталонная траектория X_{Anom}^{l}.
- Ограничения на полноту и точность распознавания:

$$e_1 < const_1 \text{ и } e_2 < const_2$$

где:

e_1 – число ошибок распознавания первого рода;

e_2 – число ошибок распознавания второго рода;

$const_1$, $const_2$ – заданные числовые ограничения.

Требуется с учетом ограничений на полноту и точность провести распознавание нештатного поведения в работе системы.

Под ошибкой распознавания первого рода будем понимать такую ошибку распознавания, когда алгоритм распознал вхождение траектории нештатного поведения в наблюдаемую траекторию на тех отсчетах, на которых не наблюдается нештатного поведения. Под ошибкой второго рода будем понимать такую ошибку распознавания, когда траектория нештатного поведения реально входит в наблюдаемую траекторию, но не распознается алгоритмом распознавания. Неверная классификация траектории нештатного поведения так же относится к ошибкам второго рода.

Эта задача относится к классу задач распознавания образов. Для решения такого рода задач используется большое число методов, таких как: нейронные сети, алгоритм k-ближайших соседей, алгоритмы на основе SSA и др. Однако, их применение для данной задачи осложнено наличием искажений траекторий нештатного поведения, как по длине, так и по времени. Чтобы учесть данные особенности задачи, было разработано параметрическое семейство алгоритмов распознавания и метод обучения на основе генетического алгоритма.

Параметрическое семейство алгоритмов распознавания

Разработанное семейство алгоритмов распознавания основано на аксиоматическом подходе. Основой аксиоматического подхода является разметка анализируемой траектории аксиомами и анализ траектории по полученному ряду разметки.

Элементарное условие $ec = ec(x_t^*, t, p)$ – это бинарная функция, определенная на отчете t и некоторой его окрестности x_t^* на траектории X, зависящая от набора параметров p. Пример элементарного условия:

$$ec(x_t^*, t, p) = \begin{cases} true, \text{если } \forall i \in [t-k, t+j], i \in N: \\ \qquad x_{i-1} < x_i < x_{i+1}, f(x^*, i) - x_i > \vartheta \geq 0 \\ false, \text{иначе} \end{cases}$$

где: $f(x^*, i) = \frac{1}{2}(x_{i-1} + x_{i+1})$, $p = \{k, j, \vartheta\}$.

Аксиома $a = a(x_t^*, t)$ – это бинарная функция, определенная в точке t и некоторой ее окрестности x_t^* на траектории X. Аксиома представляет собой булеву формулу над множеством элементарных условий. Точка траектории размечается аксиомой, если в данной точке условия аксиомы выполняются. Траектория размечается набором аксиом, если каждая точка траектории размечается некоторой аксиомой из данного набора.

Определение 1: Набор аксиом $as = \{a_1, a_2, ..., a_m\}$ будем называть системой аксиом, если он удовлетворяет следующим ограничениям:

- Условие полноты. Это условие означает, что для любой точки допустимой траектории найдется аксиома из набора аксиом её размечающая.
- Условие однозначности. Это условие заключается в том, что любая точка допустимой траектории может быть размечена лишь одной аксиомой из набора.

Разметкой траектории $X = (\bar{x}_1, \bar{x}_2, ..., \bar{x}_n)$, полученной при помощи системы аксиом as, является последовательность $J = (j_1, j_2, ..., j_n)$, где j_i – номер аксиомы из набора as, которая размечает отсчет x_i.

В рамках аксиоматического подхода распознавание нештатного поведения в работе наблюдаемой системы происходит следующим образом:

1) Размечаются эталонные траектории $\{X_{Anom}\}$, соответствующие различным классам нештатного поведения.
2) Размечается наблюдаемая траектория X и формируется ряд разметки J.
3) В ряду разметки J ищутся подпоследовательности номеров аксиом соответствующие разметкам эталонных траекторий.

Таким образом, определение нештатного поведения в работе наблюдаемой системы ведется не путем поиска эталонных траекторий $\{X_{Anom}\}$ в наблюдаемой траектории X, а путем поиска разметок эталонных траекторий в ряду разметки J.

Решение рассматриваемой задачи распознавания будем искать среди множества алгоритмов S, каждый из которых включает в себя:

- алгоритм разметки и система аксиом,
- алгоритм поиска разметок эталонных траекторий в разметке наблюдаемой траектории.

Для поиска разметок эталонных траекторий можно использовать следующие алгоритмы: алгоритмы на основе метрики Минковского, и DTW (Dynamic Time Warping) , алгоритмы на основе нейросетей.

Задача построения алгоритма распознавания

Пусть задан набор прецедентов TS в виде экземпляров траекторий X, полученных в различных условиях работы системы. Траектории X из набора TS содержат как участки нештатного поведения, так и участки нормального поведения. Для каждой из траекторий X указаны:

- Отсчеты начала и окончания участка участков нештатного поведения;
- Для каждомго участка нештатного поведения указан тип нештатной ситуации из множества $W = \{l\}_1^L \cup \{0\}$.

Весь набор TS разделим на обучающую $\overline{TS}$ и контрольную $\overline{\overline{TS}}$ выборку:

$$TS = \overline{TS} \cup \overline{\overline{TS}} \quad \overline{TS} \cap \overline{\overline{TS}} = \varnothing$$

Пусть определена целевая функция $\varphi(e_1, e_2)$, где e_1 и e_2 - число ошибок распознавания первого и второго рода, которая отвечает следующим требованиям:

- Функция $\varphi(e_1, e_2)$ определена для любых неотрицательных целых значений аргументов e_1 и e_2.
- Функция $\varphi(e_1, e_2)$ монотонно неубывает по каждому из аргументов.

Число ошибок распознавания первого рода e_1 (второго рода e_2) определяется для некоторого алгоритма распознавания Al на некоторой траектории X. Под числом ошибок распознавания на выборке TS будем понимать сумму ошибок распознавания на всех траекториях данной выборки. Задача построения алгоритма распознавания нештатных ситуаций в работе динамической системы состоит в следующем.

Дано:

- Обучающая выборка $\overline{TS}$ и контрольная выборка $\overline{\overline{TS}}$,
- Целевая функция $\varphi(e_1, e_2)$.

Требуется построить алгоритм распознавания $Al : Al \in S$, который будет удовлетворять следующему набору ограничений:

1. Алгоритм Al не должен выдавать ошибок на обучающей выборке $\overline{TS}$:

$$e_1(Al, \overline{TS}) = e_2(Al, \overline{\overline{TS}}) = 0$$

2. Алгоритм Al должен минимизировать значение целевой функции $\varphi(e_1, e_2)$ на контрольной выборке $\overline{\overline{TS}}$:

$$Al = \underset{Al}{argmin}(\varphi(e_1(Al, \overline{\overline{TS}}), e_2(Al, \overline{\overline{TS}})))$$

3. Вычислительная сложность работы алгоритма распознавания $\Theta_{Al}(m)$ на произвольной траектории, длины не больше m, должна быть ограничена наперед заданной функцией $\theta(m)$, которая определяется характеристиками используемого вычислителя и скоростью развития процессов в анализируемой системе:

$$\Theta_{Al}(m) \le \theta(m)$$

Данная задача построения алгоритма распознавания соответствует классической постановке задачи обучения по прецедентам.

Генетический алгоритм построения системы аксиом

Особью в популяции Pl является система аксиом as:

$$Pl = \{as_1, as_2, \ldots, as_n\}$$

Цель оптимизации состоит в получении системы аксиом as, минимизирующей целевую функцию $\varphi(e_1, e_2)$ на обучающей выборке $\overline{TS}$. Далее будет подробно рассмотрен каждый их этапов генетического алгоритма построения системы аксиом.

Для определения эталонных траекторий для каждого из классов нештатного поведения разделим всю известную выборку TS на 3 части следующим образом:

$$TS = \{X^j_{Anom}\}_{j=1}^{L} \cup \overline{TS} \cup \overline{\overline{TS}},$$
$$\overline{TS} \cap \overline{\overline{TS}} = \varnothing, \quad \{X^j_{Anom}\}_{j=1}^{L} \cap \overline{TS} = \varnothing, \quad \overline{\overline{TS}} \cap \{X^j_{Anom}\}_{j=1}^{L} = \varnothing$$

Из траекторий выборки TS случайным образом выделим по одному участку нештатного поведения от каждого класса нештатного поведения и поместим в множество $\{X^j_{Anom}\}_{j=1}^{L}$:

$$\{X^j_{Anom}\}_{j=1}^{L} : \forall j \in [1, L] \exists (I^i_{start}, I^i_{end}, j) \in \Psi, X^j_{Anom} \in TS$$

где: L - число классов нештатного поведения рассматриваемой технической системы. Траектории из сформированного множества $\{X^j_{Anom}\}_{j=1}^{L}$ будем использовать как эталонные траектории для различных классов нештатного поведения. После изъятия траекторий $\{X^j_{Anom}\}_{j=1}^{L}$ из выборки TS, разделим оставшуюся выборку в равной части между обучающей $\overline{TS}$ и контрольной $\overline{\overline{TS}}$ выборками. Теперь рассмотрим каждый их этапов генетического алгоритма построения системы аксиом подробнее.

Создание начальной популяции

На первом этапе генетического алгоритма требуется создать заданное числоразличных систем аксиом, которые будут составлять начальную популяцию Pl. Формирование каждой системы аксиом as в начальной популяции Pl происходит по следующему алгоритму:

1. Выбирается число аксиом в системе as: $N_a = rand(1, N_a^{max})$. Фу
2. Создается N_a аксиом, каждая из которых составляется следующим образом:
   a) Выбирается число элементарных условий в аксиоме: $N_{ec} = rand(1, N_{ec}^{max})$.
   b) Из множества элементарных условий $\{ec\}$, заданного в рамках шаблона S_k выбирается N_{ec} условий. Их параметры определяются случайным образом. При этом типы условий среди выбранного набора могут повторяться.
   c) Выбранные N_{ec} элементарных условий объединяются при помощи связок $\{\vee,\wedge\}$. Тип связки, с которой элементарное условие добавляется в аксиому выбирается случайно. При этом, с вероятностью 1/2 условие будет входить в аксиому с отрицанием, т.е. в виде $\overline{ec}$.

Полученные таким образом наборы аксиом могут не удовлетворять условиям однозначности и полноты. Для того, чтобы созданные системы аксиом удовлетворяли условию однозначности вводится приоритет аксиом в системе. Все аксиомы в системе пронумерованы:

$$as = \{a_1, a_2, \ldots, a_{N_a}\}$$

Будем считать что, если аксиома с индексом i выполняется в некоторой точке траектории X, то никакая аксиома с индексом $j : j > i$ не может выполняться в данной точке траектории. Чем меньше индекс аксиомы, тем больше ее приоритет в системе. Для любой системы аксиом с приоритетами условие однозначности выполняется.

Для того, чтобы созданные системы аксиом удовлетворяли условию полноты в каждую систему аксиом добавляется тождественная аксиома с наименьшим приоритетом:

$$as = \{a_1, a_2, \ldots, a_{N_a}, a_\infty\}$$

Тождественная аксиома a_∞ - это аксиома, которая выполняется в любой точке любой траектории. Ее приоритет не может быть изменен и всегда остается наименьшим, что отражает ее индекс ∞. Для любой системы аксиом, дополненной тождественной аксиомой, выполняется условие полноты.

Таким образом, в результате работы первого шага рассматриваемого генетического алгоритма построения системы аксиом формируется начальная популяция, состоящая из систем аксиом с приоритетами, дополненными тождественными аксиомами:

$$Pl = \{as_1, as_2, \ldots, as_n\}$$

Операции мутации и скрещивания генетического алгоритма

Операция мутации особи популяции преобразует эту особь:

$$as \to as^*$$

Операция скрещивания для двух особей популяции создает новую особь:

$$[as' \times as''] \to as^*$$

Особью в популяции Pl является система аксиом, которая имеет многоуровневую структуру: система аксиом представляет собой упорядоченный набор аксиом, каждая аксиома – это булева функция над множеством элементарных условий, каждое условие имеет собственный набор параметров. Операции мутации и скрещивания должны иметь возможность изменять каждую составляющую особи. Поэтому эти операции определены на трех уровнях:

I. Уровень систем аксиом.

Операция мутации.

Для каждой особи в популяции выполняется операция мутации:

$$\forall as \in Pl:$$

$$as = \begin{cases} as, \text{ с вероятностью } 1 - P_{as}^{mut} \\ m_{as}(as, \Delta_{as}^{mut}, R_{as}^{mut}), \text{ с вероятностью } P_{as}^{mut} \end{cases}$$

где: $m_{as}(as, \Delta_{as}^{mut}, R_{as}^{mut})$ - функция мутации системы аксиом as.

С вероятностью P_{as}^{mut} система аксиом as изменяется функцией мутации m_{as}. Функция мутации на уровне систем аксиом m_{as} использует следующие варианты изменения системы аксиом:

- Удаление некоторой аксиомы из системы аксиом.
- Замена некоторой аксиомы в системе аксиом на новую. Новая аксиома создается случайным образом, как и при генерации начальной популяции.
- Добавление новой аксиомы в систему аксиом. Новая аксиома создается случайным образом, как и при генерации начальной популяции. Приоритет новой аксиомы в системе аксиом определяется случайным образом.

- С вероятностью R_{as}^{mut} изменение приоритета некоторой аксиомы в системе аксиом.

Функция мутации работает по следующему алгоритму:

1. Выбирается первая аксиома $a_i, i = 1$ в системе аксиом as.
2. Для выбранной аксиомы a_i выполняется:
   a) Случайным образом выбирается число: $r \in [0,1]$.
   b) Если $r > \Delta_{as}^{mut}$, то происходит переход на шаг 3.
   c) С вероятностью 1/2 выбирается одно из действий, которое затем применяется к аксиоме a_i:
      - Удаление аксиомы из системы аксиом.
      - Замена аксиомы на новую. При этом, новая аксиома создается случайным образом, как и при генерации начальной популяции.
3. Если в системе аксиом as были выбраны все аксиомы, за исключением тождественной a_∞, то переход на шаг 4. Иначе, выбор следующей аксиомы a_{i+1} и переход на шаг 2.
4. Приоритет аксиом в системе выравнивается: если аксиома с индексом i отсутствует в системе, то все аксиомы с индексом $j > i$ уменьшают значение индекса на единицу.
5. Выбирается первая аксиома $a_i, i = 1$ в системе аксиом as.
6. Для выбранной аксиомы a_i выполняется:
   a) Случайным образом выбирается число: $r \in [0,1]$.
   b) Если $r > R_{as}^{mut}$, то происходит переход на шаг 7.
   c) Случайным образом выбирается аксиома a_j, с которой a_i обменивается индексами.
7. Если в системе аксиом as были выбраны все аксиомы, за исключением тождественной a_∞, то переход на шаг 8. Иначе, выбор следующей аксиомы a_{i+1}, которая еще не была выбрана, и переход на шаг 6.
8. Случайным образом выбирается целое число p от 0 до $[\Delta_{as}^{mut} \cdot N_a]$, где: N_a - это число аксиом в системе аксиом as до начала выполнения данного алгоритма мутации. Если $(N_a + p) > N_a^{max}$, то p уменьшается: $p = N_a^{max} - N_a$.

9. Случайным образом создается p новых аксиом по алгоритму, используемому при генерации начальной популяции.
10. Новые аксиомы добавляются в систему as. Приоритет каждой новой аксиомы определяется случайным образом из диапазона $[1, N_a^* + 1]$, где N_a^* - текущее число аксиом в системе as. При добавлении новой аксиомы с выбранным индексом i все аксиомы с индексом $j > i$ увеличивают значение индекса на единицу.

Параметрами операции мутации на уровне систем аксиом являются: P_{as}^{mut} - вероятность мутации особи на уровне систем аксиом; Δ_{as}^{mut} - вероятность изменения аксиомы в системе as; R_{as}^{mut} - вероятность изменения приоритета аксиомы в системе аксиом as.

Операция скрещивания.

Выбор особей для скрещивания в популяции Pl происходит по следующему алгоритму:

1. Индекс особей в популяции устанавливается на первую систему аксиом: $i = 1$.
2. Для системы аксиом as_i выбирается случайное число $r \in [0,1]$. Если $r > P_{as_i}^{cr}$, то переход на шаг 4.
3. Для системы аксиом as_i выбирается пара для скрещивания:
   a) Выбирается случайное действительное число r от 0 до $(\sum_{\forall as_j \in Pl} P_{as_j}^{cr} - P_{as_i}^{cr})$.
   b) Определяется индекс k, для которого выполняется:
   $$\sum_{\forall j \in [1,k-1],\ j \neq i} P_{as_j}^{cr} \le r \le \sum_{\forall j \in [1,k],\ j \neq i} P_{as_j}^{cr}$$
   Если таких индексов оказалось несколько[1], то значение k выбирается случайным образом из них.
   c) Пара индексов (i,k) сохраняется в множестве C_{cr}.
4. Если индекс i соответствует последней особи в популяции Pl, то алгоритм выбора особей для скрещивания завершается. Иначе, выбор следующей системы аксиом ($i = i + 1$) и переход на шаг 2 данного алгоритма.

[1]Это возможно, если несколько систем аксиом имеют нулевое значение $P_{as}^{cr} = 0$.

В результате работы данного алгоритма в множестве C_{cr} формируется набор пар индексов особей в популяции, для каждой из которых запускается процедура скрещивания.

Операция скрещивания двух систем аксиом формирует новую систему аксиом:

$$[as'\{a'_1, a'_2, \ldots, a'_m, a_\infty\} \times as''\{a''_1, a''_2, \ldots, a''_n, a_\infty\}] \rightarrow$$
$$\rightarrow as^*\{a^*_1, a^*_2, \ldots, a^*_k, a_\infty\}$$

Каждая аксиома a^*_i системы as^* получена одним из следующих способов:

- a^*_i взята из as';
- a^*_i взята из as'';
- a^*_i получена в результате скрещивания аксиом $a'_j \in as'$ и $a''_l \in as''$.

Операция скрещивания систем аксиом as' и as'' проходит по следующему алгоритму:

1. Случайным образом определяется число аксиом от первой системы аксиом:

$$n_1 = \left[N'_a \cdot \frac{rand(1,2)}{3} \right]$$

   где: N'_a --- число аксиом в системе аксиом as'; $rand(1,2)$ --- функция генерации случайных чисел из заданного диапазона с равномерным распределением.

2. Случайным образом определяется число аксиом от второй системы аксиом:

$$n_2 = \left[N''_a \cdot \frac{rand(1,2)}{3} \right]$$

   где: N''_a --- число аксиом в системе аксиом as''.

3. Если $(n_1 + n_2) > N^{max}_a$, то выбирается наименьшее целое число k, при котором:

$$n_1 + n_2 - 2 \cdot k < N^{max}_a$$

   Значения числа аксиом от каждой из систем уменьшается на k:

$$n_1 = n_1 - k, n_2 = n_2 - k$$

4. От первой системы аксиом as' выбирается n_1 аксиом. Множество выбранных аксиом обозначим A'. Каждая из аксиом выбирается следующим образом:

   а) Вычисляется сумма:

$$S = \sum_{a_i \in as' \setminus A'} P_{a_i}^{cr}$$

b) Случайным образом выбирается действительное число r из диапазона $[0, S]$.

c) Выбирается такая аксиома a_k, для которой выполняется:

$$\sum_{a_i \in as' \setminus A', i<k} P_{a_i}^{cr} \le r \le \sum_{a_i \in as' \setminus A', i \le k} P_{a_i}^{cr}$$

5. Аналогично от второй системы аксиом as'' выбирается n_2 аксиом. Множество выбранных аксиом обозначим A''.
6. Пусть для определенности $n_1 \ge n_2$. Обозначим: $n_{cr} = [n_2/2]$. Из множества A' случайным образом выделяется n_{cr} аксиом A'_{cr}, из множества A'' выделяется n_{cr} аксиом A''_{cr}:

$$A' = A'_{no} \cup A'_{cr}, A'' = A''_{no} \cup A''_{cr}, |A'_{cr}| = |A''_{cr}| = n_{cr}$$

7. Для каждой аксиомы из множества A'_{cr} случайным образом выбирается соответствующая аксиома из A''_{cr}. Для сформированных пар аксиом производится скрещивание на уровне аксиом. Полученное в результате множество аксиом обозначим A_{cr}.
8. Аксиомы из множеств A'_{no}, A''_{no} и A_{cr} объединяются в систему аксиом as^*. Приоритеты аксиом в новой системе определяются случайным образом. Кроме того, в as^* добавляется тождественная аксиома a_∞ с минимальным приоритетом.

Параметрами операции скрещивания на уровне систем аксиом является: P_{as}^{cr} - вероятность участия системы аксиом as в скрещивании; P_a^{cr} - вероятность участия аксиомы a в скрещивании.

II. Уровень аксиом.

Операция мутации.

Для всех аксиом a из каждой системы аксиом $as \in Pl$ производится операция мутации на уровне аксиом. Эта операция изменяет аксиому с заданной вероятностью:

$$a = \begin{cases} a, \text{ с вероятностью } 1 - P_a^{mut} \\ m_a(a, \Delta_a^{mut}), \text{ с вероятностью } P_a^{mut} \end{cases}$$

где: $m_a(a, \Delta_a^{mut})$ - функция мутации аксиомы.

Как было указано выше, аксиома представляет собой булеву формулу над множеством элементарных условий:

$$a(x_t^*, t) = \bigvee_i \bigwedge_j ec_{ij}(x_t^*, t, P_{ij})^{\delta_{ij}}$$

Функция мутации m_a использует следующие варианты изменения аксиомы a:

- Изменение логической связки между элементарными условиями в аксиоме:

$$\vee \to \wedge, \wedge \to \vee$$

- Добавление или снятие знака отрицания для некоторого элементарного условия в аксиоме:

$$ec \to \overline{ec}, \overline{ec} \to ec$$

- Удаление элементарного условия из аксиомы.
- Замена условия на новое.
- Добавление нового элементарного условия в аксиому.

Функция мутации m_a работает по следующему алгоритму:

1. Выбирается первое элементарное условие $ec_i, i = 1$, входящее в аксиому a.
2. Для выбранного элментарного условия ec_i выполняется:
   a) Случайным образом выбирается число: $r \in [0,1]$.
   b) Если $r > \Delta_a^{mut}$, то происходит переход на шаг 3.
   c) Случайным образом выбирается один из следующих вариантов мутации, который, затем, применяется к условию ec_i:
      - Изменение логической связки перед элементарным условием:

$$\vee \to \wedge, \wedge \to \vee$$

      - Добавление или снятие знака отрицания:

$$ec_i \to \overline{ec_i}, \overline{ec_i} \to ec_i$$

      - Удаление элементарного условия из аксиомы.
      - Замена условия на новое. Тип нового условия выбирается случайным образом из множества $\{ec\}$, которое фиксировано в используемом шаблоне S_k. Параметры условия определяются случайным образом.
3. Если в аксиоме a были выбраны все элементарные условий, то переход на шаг 4. Иначе, выбор следующего условия ec_{i+1}, которое еще не было выбрано, и переход на шаг 2.

4. Случайным образом выбирается целое число p от 0 до $[\Delta_a^{mut} \cdot N_{ec}]$, где: N_{ec} - это число элементарных условий в аксиоме a до начала выполнения данного алгоритма мутации. Если $(N_{ec} + p) > N_{ec}^{max}$, то значение p полагается равным:

$$p = N_{ec}^{max} - N_{ec}$$

5. Случайным образом выбирается p элементарных условий из множества $\{ec\}$, которое фиксировано в используемом шаблоне S_k. Типы выбранных условий могут совпадать, параметры каждого условия определяются случайным образом.
6. Новые элементарные условия добавляются в аксиому a. При добавлении каждое элементарное условие либо образует новый дизъюнкт в формуле аксиомы a, либо добавляется к одному из уже существующих дизъюнктов. Выбор способа вхождения в аксиому и выбор дизъюнкта осуществляется случайным образом.

Параметрами операции мутации на уровне аксиом являются: P_a^{mut} - вероятность участия аксиомы a в мутации; $\Delta_a^{mut} \in [0,1]$ - параметр, который определяет степень изменения аксиомы a в ходе мутации.

Операция скрещивания.

Для каждой пары аксиом $[a', a'']$, выбранной в алгоритме скрещивания для систем аксиом, производится операция скрещивания на уровне аксиом. Эта операция создает новую аксиому на основе двух выбранных аксиом:

$$[a' \times a''] \to a^*$$

Новая аксиома формируется путем объединения двух подформул от скрещиваемых аксиом. Кроме того, происходит скрещивание элементарных условий одного типа, входящих в обе аксиомы. Операция скрещивания аксиом a' и a'' работает по следующему алгоритму:

1. От аксиомы a' выбираются дизъюнкты, которые будут входить в новую аксиому. Вероятность выбора каждого дизъюнкта равна $1/2$. Множество выбранных дизъюнктов обозначим E'.
2. От аксиомы a'' выбираются дизъюнкты, которые будут входить в новую аксиому. Вероятность выбора каждого дизъюнкта равна $1/2$. Множество выбранных дизъюнктов обозначим E''.

3. Обозначим число элементарных условий в E' символом n'_{ec}. Число условий в E'' - символом n''_{ec}. Если $(n'+n'') > N_{ec}^{max}$, то из каждого из множеств удаляется по одному дизъюнкту до тех пор, пока данное условие не будет выполнено.
4. Множество всех элементарных условий в дизъюнктах из множества E' обозначим E'_{ec}. Множество всех элементарных условий в E'' обозначим E''_{ec}. Для каждого элементарного условия ec'_i в E'_{ec} выполняется:
    a) Случайным образом выбирается число $r \in [0,1]$.
    b) Если $r \le P_{ec'_i}^{cr}$, то среди условий E''_{ec} случайным образом выбирается элементарное условие ec''_j того же типа, что и ec'_i.
    c) Если была выбрана пара элементарных условий ec'_i и ec''_j, то происходит процедура их скрещивания на уровне элементарных условий. Полученное в результате условие ec^* заменяет ec'_i в соответствующем дизъюнкте в E'.
5. Для каждого элементарного условия ec''_j в E''_{ec} выполняется:
    a) Случайным образом выбирается число $r \in [0,1]$.
    b) Если $r \le P_{ec''_j}^{cr}$, то среди условий E'_{ec} случайным образом выбирается элементарное условие ec'_i того же типа, что и ec''_j.
    c) Если была выбрана пара элементарных условий ec''_j и ec'_i, то происходит процедура их скрещивания на уровне элементарных условий. Полученное в результате условие ec^* заменяет ec''_j в соответствующем дизъюнкте в E''.
6. Все дизъюнкты из E' и E'' объединяются при помощи операции $\vee$ и образуют аксиому a^*, которая считается результатом скрещивания аксиом a' и a''.

Параметром операции скрещивания на уровне аксиом является: P_{ec}^{cr} - вероятность участия элементарного условия ec в скрещивании.

III. Уровень элементарных условий.

Операция мутации.

Для каждого элементарного условия из всех аксиом, входящих в системы аксиом из популяции Pl, производится операция мутации на уровне элементарных условий. Эта операция изменяет параметры элементарного условия с заданной вероятностью:

$$ec(x_t^*, t, p) = \begin{cases} ec(x_t^*, t, P), \text{ с вероятностью } 1 - P_{ec}^{mut} \\ ec(x_t^*, t, P'), \text{ с вероятностью } P_{ec}^{mut} \end{cases}$$

где: $P' = m_{ec}(P, \Delta_{ec}^{mut})$ - новый набор значений параметров элементарного условия ec; $m_{ec}(P, \Delta_{ec}^{mut})$ - функция мутации параметров элементарного условия.

Для изменения каждого из действительных параметров $p_i \in P_{ec}$ элементарного условия ec используется следующий алгоритм:

1. Выбирается действительное число r из диапазона $[0, \Delta_{ec}^{mut}]$.
2. Параметр p_i изменяется на величину $c_i \cdot r$, где c_i - константа, соответствующая параметру p_i. Знак изменения: $(p_i + c_i \cdot r)$ или $(p_i - c_i \cdot r)$ - выбирается случайным образом.

Для изменения целочисленных параметров дополнительно используется округление величины, на которую происходит изменение значения параметра элементарного условия.

Параметрами операции мутации на уровне элементарных условий являются: P_{ec}^{mut} - вероятность участия элементарного условия ec в мутации; Δ_{ec}^{mut} - максимальная степень изменения значения параметра элементарного условия ec в ходе мутации.

Операция скрещивания.

Для каждой пары элементарных условий $[ec', ec'']$, выбранной в алгоритме скрещивания на уровне аксиом, производится операция скрещивания на уровне элементарных условий. Эта операция создает новое элементарное условие из двух выбранных элементарных условий одного типа:

$$[ec'(x_t, t, P') \times ec''(x_t, t, P'')] \rightarrow ec^*(x_t, t, P^*)$$

Тип нового условия совпадает с типом скрещиваемых условий. Каждый из действительных параметров $p_i^* \in P^*$ нового элементарного условия ec^* определяется как среднее между соответствующими параметрами ec' и ec'':

$$p_i^* = \frac{p_i' + p_i''}{2}$$

Для целочисленных параметров дополнительно применяется процедура округления:

$$p_i^* = \left[\frac{p_i' + p_i''}{2} \right]$$

Если элементарное условие ec^* имеет нечисловые параметры, то они наследуются у одного из условий ec' и ec''. Для каждого такого параметра случайным образом определяется то условие, от которого наследуется значение данного параметра.

В операциях мутации и скрещивания для выбора случайных чисел из заданного диапазона используется функция с равномерным распределением на этом диапазоне.

Функции значимости

Операции мутации и скрещивания являются достаточно сложными и обладают большим набором параметров. В зависимости от значений этих параметров операция мутации может быть более или менее дестабилизирующей. Для того, чтобы параметры операций мутации и скрещивания определялись автоматически для каждой системы аксиом на каждой итерации генетического алгоритма, было предложено использовать функции значимости для особей популяции и для элементов особей в отдельности.

Функция значимости системы аксиом M_{as} определяется следующим образом:

$$M_{as} = c_1 e_1 + c_2 e_2 + c_3 \frac{\varphi_{as}(e_1, e_2) + 1}{\varphi_{min}(e_1, e_2) + 1}$$

где: e_1, e_2 - число ошибок распознавания I и II рода алгоритма распознавания на основе системы аксиом as; $\varphi_{as}(e_1, e_2)$ - значение целевой функции для системы аксиом as на обучающей выборке $\overline{TS}$; $\varphi_{min}(e_1, e_2)$ - минимальное значение целевой функции среди всех систем аксиом в популяции; $c_i, i = \overline{1,3}$ - положительные константы.

Слагаемые $(c_1 \cdot e_1)$ и $(c_2 \cdot e_2)$ входят в функцию M_{as} для того, чтобы функция значимости отражала насколько качественно алгоритм работает на контрольной выборке. Последнее слагаемое в формуле для M_{as} позволяет оценить относительное качество работы алгоритма распознавания на основе системы аксиом as по сравнению с другими системами аксиом в популяции. Чем лучше система аксиом, тем меньше она изменяется в ходе мутации и с большей вероятностью участвует в скрещивании.

Функция значимости аксиомы M_a определяется как:

$$M_a = c_4 M_{as} + c_5 \left| \frac{Num(\overline{X}) - num_a}{Num(\overline{X})} \right| + c_6 \left| \frac{L - eth_a}{L} \right|$$

где: M_{as} - функция значимости системы аксиом as, в которую входит аксиома a; $Num(\overline{X})$ - число участков нештатного поведения в обучающей выборке $\overline{TS}$; num_a - число срабатываний аксиомы на траекториях обучающей выборки; L - число эталонных

траекторий; eth_a - число срабатываний аксиомы на эталонных траекториях; $c_i, i = \overline{4,6}$ - положительные константы.

Слагаемое $(c_4 \cdot M_{as})$ входит в формулу M_a для того, чтобы функция значимости аксиомы отражала качество работы алгоритма распознавания на основе всей системы аксиом, в которую входит аксиома a. Второе и третье слагаемое в формуле позволяют оценить относительный вклад аксиомы в распознавание участков нештатного поведения. Если условия аксиомы выполняются по разу только на участках нештатного поведения, то вклад аксиомы в распознавание считается большим. Если же условия аксиомы не выполняются нигде или выполняются многократно и не только на участках нештатного поведения, то вклад аксиомы считается меньшим. Чем больше значение функции значимости M_a, тем сильнее изменяется аксиома в ходе мутации и с меньшей вероятностью участвует в скрещивании.

Параметры операций мутации и скрещивания определяются в зависимости от функций значимости:

$$[P_{as}^{mut}, \Delta_{as}^{mut}, R_{as}^{mut}, P_{as}^{cr}] = F_1(M_{as})$$

$$[P_{ec}^{mut}, \Delta_{ec}^{mut}, P_{ec}^{cr}, P_a^{mut}, \Delta_a^{mut}, P_a^{cr}] = F_2(M_a)$$

Функции F_1, F_2 построены таким образом, чтобы выполнялись условия:

- Все параметры операций скрещивания и мутации принимают значения от 0 до 1.
- Все параметры мутации: P_{ec}^{mut}, Δ_{ec}^{mut}, P_a^{mut}, Δ_a^{mut}, P_{as}^{mut}, Δ_{as}^{mut}, R_{as}^{mut} --- прямо пропорциональны соответствующей функции значимости.
- Все параметры скрещивания: P_{ec}^{cr}, P_a^{cr}, P_{as}^{cr} --- обратно пропорциональны соответствующей функции значимости.

Таким образом, чем меньше значение функции значимости, тем меньше изменяется особь или элемент особи при мутации и больше участвует в скрещивании.

В данной работе для параметров мутации используются функции F_1 и F_2 следующего вида:

$$P_{as}^{mut} = \frac{M_{as}}{M_{as}^{max}} \cdot (1 - d_{as}^{mut}) + d_{as}^{mut}$$

где: $d_{as}^{mut} \in (0,1)$ - константа, которая ограничивает минимальное значение параметра P_{as}^{mut}; M_{as}^{max} - параметр, который определяет максимальное значение для функции M_{as}: если вычисленное значение M_{as} превышает M_{as}^{max}, то M_{as} устанавливается равным M_{as}^{max}.

Использование максимального ограничения для M_{as} позволяет гарантировать, что значение P_{as}^{mut} находится в диапазоне [0,1]. Остальные параметры операции мутации (P_{ec}^{mut} , Δ_{ec}^{mut} , P_{a}^{mut} , Δ_{a}^{mut} , Δ_{as}^{mut} , R_{as}^{mut}) определяются аналогичным образом.

Для параметров скрещивания используются функции вида:

$$P_{as}^{cr} = \frac{1 - d_{as}^{cr}}{1 + M_{as}} + d_{as}^{cr}$$

где: d_{as}^{cr} --- константа, которая ограничивает минимальное значение параметра P_{as}^{cr} .

Остальные параметры операции скрещивания (P_{ec}^{cr} , P_{a}^{cr}) определяются аналогичным образом.

Автоматическое определение параметров операций мутации и скрещивания на каждой итерации алгоритма для всех систем аксиом, на основании M_{as}, и параметров этих операций для аксиом и элементарных условий, на основании M_{a}, позволяет:

- избежать ручного подбора параметров операций мутации и скрещивания;
- улучшить сходимость генетического алгоритма по сравнению со случаем, когда параметры операций не изменяются на итерациях генетического алгоритма.

Эффективность алгоритма

Использование для построения системы аксиом генетического алгоритма с функциями значимости позволяет получать алгоритмы распознавания, точность которых в 2-3 раза выше по сравнению с алгоритмами распознавания для построения которых использовался простой генетический алгоритм.

С материалами подраздела 4.3.1. можно ознакомиться в работах:

1. Коваленко Д.С., Костенко В.А., Щербинин В.В. Параметрическое семейство алгоритмов распознавания нелинейно искаженных фазовых траекторий динамических систем// XIV Всероссийская научно-техническая конференция "Нейроинформатика-2012": Сборник научных трудов. Ч.1. М.: НИЯУ МИФИ, 2012. – С.266-276.
2. В.А. Костенко, Д.С. Коваленко. Алгоритмы распознавания нештатного поведения динамических систем устойчивые к нелинейным искажениям фазовых траекторий системы// Труды Международной научно-практической конференции «Передовые информационные технологии, средства и системы автоматизации и их внедрение на российских предприятиях» AITA-2011. – М.: Институт проблем управления им. В. А. Трапезникова РАН, 2011. – С. 897–905.
3. D. Kovalenko, V. Kostenko A Genetic Algorithm for Construction of Recognizers of Anomalies in Behaviour of Dynamical Systems// Proceedings of the IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications, IEEE Press, China. 2010. - pp.258-263.

4. Коваленко Д.С. Методы и программные средства обучения алгоритмов распознавания участков фазовых траекторий: дис. кан. физ.-мат. наук: 05.13.11/ МГУ им. М.В. Ломоносова. – М.: 2011. – 168с.

4.4 Генетический алгоритм для решения задачи о рюкзаке

Будет прочитан на основе материалов раздела 3.1. учебного пособия [Скобцов Ю.А. Основы эволюционных вычислений. Донецк.: ДонНТУ, 2008.- 326с.].

4.5. Генетический алгоритм для решения задачи определения минимально необходимого числа процессоров и построения расписания выполнения функциональных задач со временем выполнения не превышающим заданный директивный срок

4.5.1. Математическая формулировка задачи

Задачу построения расписания как задачу условной оптимизации можно сформулировать следующим образом (см. раздел 1.4.1.).

Дано: H(PR)=(P, $\prec$*)*- модель программы, *T=f(HP,HW)*- функция вычисления времени выполнения расписания *HP* на архитектуре *HW* (целевая функция), T^{dir} - директивный срок выполнения программы.

Требуется построить: HP– расписание выполнения программы такое, что:

$$\min_{HP} M(HP, HW)$$
$$T = f(HP, HW) \le T^{dir}$$
$$HP \in HP^{*}_{1-5}$$

Здесь $M(HP, HW)$ - число процессоров, на котором выполняется расписание. Функция вычисления времени выполнения расписания может быть задана в аналитическом виде или в виде имитационной модели.

4.5.2. Кодирование решений

В данном разделе рассмотрим математические структуры для параметрического представления расписаний, соответствующие алгоритмы восстановления расписания по его параметрическому представлению не нарушающие ограничений 1-5, введем операции изменения параметров, докажем возможность задания любого допустимого расписания параметрическим представлением с использованием приоритетов и соответствующих быстрых алгоритмов восстановления и введем способ кодирования решений на основе параметрического представления расписаний.

Параметрическое представление расписаний с использованием приоритетов

Расписание задается вектором Y_{N+K} :

$$Y_{N+K} \equiv (\bigcup_{i=1}^{K} \langle YPR \rangle_i)$$

$$\langle YPR \rangle_i \equiv (\langle YE \rangle_i \bigcup \langle YP \rangle)$$

$$\langle YP \rangle \equiv (\bigcup_{j=1}^{N_i} \langle YP \rangle_j)$$

K - число процессов в H, N_i – число рабочих интервалов в i-ом процессе, $N = \sum_{i=1}^{K} N_i$ - число рабочих интервалов в H, $Y \equiv (\langle y1 \rangle \bigcup \langle y2 \rangle)$ - операция построения вектора $Y = (y1, y2)$ из параметров $\langle y1 \rangle$ и $\langle y2 \rangle$.

Параметр $\langle YE \rangle_i \in [1,\ldots,K] \subset Z$ содержит номер процессора, на котором выполняются рабочие интервалы i-го процесса, т.е. параметры $\langle YE \rangle$ однозначно определяют распределение рабочих интервалов по SP (привязку). Параметры $\langle YE \rangle$ могут принимать значения от 1 до K. Если значения всех параметров $\langle YE \rangle$ равны, то все рабочие интервалы выполняются на одном процессоре, если все значения различны, то рабочие интервалы каждого процессора выполняются на своем процессоре. В силу ограничения 5, максимальное число не пустых процессоров равно числу процессов K в H.

Параметры $\langle YP \rangle \in [1,\ldots,L] \subset Z$ используются алгоритмом восстановления расписания (определение отношения полного порядка в каждом SP_i) в качестве приоритетов рабочих интервалов.

При данном способе представления расписания число целочисленных переменных равно $N+K$.

Для описания алгоритма восстановления, множество рабочих интервалов разобьем на три подмножества: $P = P1 \bigcup P2 \bigcup P3$. $P1$ - множество рабочих интервалов распределенных в SP (определен порядковый номер рабочего интервала) алгоритмом восстановления на предыдущих шагах; $P2$ - множество рабочих интервалов, у которых все предшественники принадлежат $P1$; $P3$ - множество рабочих интервалов, у которых хотя бы один из предшественников не принадлежит $P1$.

Алгоритм восстановления полного порядка рабочих интервалов в SP (A1).

1. Начальное разбиение множества P:

$P1 = \varnothing$;

$P2 = \{p_j : \forall j,k \in [1,\ldots,N] \mid \prec_{jk} = \varnothing\}$ - множество рабочих интервалов в H без предшественников;

$P3 = \{p_j : p_j \in P \setminus P2\}$ - множество рабочих интервалов в H, у которых имеются предшественники.

2. Находим в *P2* рабочий интервал с наименьшим значением параметра $\langle YP \rangle$ (если таких интервалов более одного, то выбираем интервал с наименьшим номером):

- размещаем его в конец соответствующего списка *SP* (номер списка определяется значением параметра $\langle YE \rangle$);
- переносим его из *P2* в *P1*.

3. Проверяем *P3* с целью возможности переноса рабочих интервалов в *P2*:
если есть рабочие интервалы, у которых все предшественники принадлежат *P1*, то переносим их в *P2*.

4. Если $P2 \neq \varnothing$, то к п.2, иначе завершить работу.

Утверждение 1. Алгоритм *A1* восстановления расписания по его параметрическому представлению Y_{N+K} получает расписание $HP \in HP^*_{1-5}$, и расписание восстанавливается однозначно.

Доказательство. Ограничение 5 не нарушается в силу того, что все рабочие интервалы, принадлежащие одному и тому же процессу, имеют одно и тоже значение параметра $\langle YE \rangle$ и будут размещены (п.2 алгоритма) в один и тот же список.

Ограничения 1,2 не нарушается в силу того, что каждый рабочий интервал переносится из *P2* в соответствующий *SP*, причем перенос осуществляется лишь один раз в ходе работы алгоритма.

Выполнение ограничений 3,4 и однозначность восстановления расписания докажем показав, что алгоритм *A1* получает на каждом шаге допустимое частичное расписание и это расписание получается однозначно. Будем представлять расписание в ярусной форме максимальной высоты. Расписание, полученное после размещения первого выбранного из *P2* рабочего интервала, всегда удовлетворяет ограничениям 3,4. Множество *P2* определено однозначно (содержит рабочие интервалы без предшественников), выбор рабочего интервала из *P2* однозначен и его порядковый номер в соответствующем *SP* определен однозначно, равен 1 (п.2 алгоритма). Пусть частичное расписание, полученное после размещения *(j-1)* рабочих интервалов допустимо и однозначно. Покажем, что на *j*-м шаге рабочий интервал выбирается и размещается алгоритмом *A1* в расписание однозначно и полученное частичное расписание

удовлетворяет ограничениям 3,4. Множество *P2* на *j*-м шаге алгоритма *A1* определяется однозначно рабочими интервалами, размещенными в расписание на предыдущих шагах. Выбор рабочего интервала из *P2* однозначен и его порядковый номер в соответствующем *SP* определен однозначно, максимальный порядковый номер рабочего интервала в данном *SP* плюс 1 (п.2 алгоритма). Данный рабочий интервал размещается на *j*-й ярус. Поскольку все его предшественники (в соответствии с определением множества *P2*) уже распределены на предыдущих шагах на ярусы с меньшими номерами, то отношения частичного порядка заданное в *H* не нарушается (выполняется ограничение 3). Ограничение 4 выполняется в силу того, что полученное промежуточное расписание представлено в ярусной форме. Таким образом, после выполнения алгоритмом *A1 N* шагов полученное расписание будет удовлетворять ограничениям 3,4 и порядковые номера рабочих интервалов в списках определяются однозначно.

Теорема 2. Любое допустимое расписание $HP \in HP^{*}_{1-5}$ может быть задано параметрическим представлением с использованием приоритетов (Y_{N+K}) и однозначно восстановлено алгоритмом *A1*, если допустимая верхняя граница *L* значений параметров $\langle YP \rangle$ больше или равна числу рабочих интервалов ($L \geq N$).

Доказательство. Расписания могут отличаться друг от друга привязкой и порядком рабочих интервалов (смотри раздел 3). Любая допустимая привязка (распределение рабочих интервалов по *SP*) может быть задана соответствующими значениями параметров $\langle YE \rangle \in [1,\ldots,K]$ и однозначно восстановлена алгоритмом *A1*.

Возможность задания любого допустимого порядка для фиксированной привязки можно доказать методом математической индукции. Для любого произвольно выбранного SP_k можно показать, что для рабочего интервала с порядковым номером равным 1, значение параметра $\langle YP \rangle$ для этого рабочего интервала можно выбрать таким образом, что он может иметь любой допустимый порядковый номер в SP_k, который однозначно получает алгоритм *A1* (место возможного размещения рабочего интервала определяется размещением его последователей в *HP* с использованием ярусной формы *HP*). Далее полагаем, что для (*i-1)* рабочих интервалов из списка SP_k с наименьшими порядковыми номерами, можно получить любые допустимые варианты порядка, выбирая соответствующий набор значений параметров $\langle YP \rangle$. Можно показать, что для рабочего интервала с порядковым номером равным *i*, значение параметра $\langle YP \rangle$ для этого рабочего интервала можно выбрать таким образом, что он может иметь любой допустимый порядковый номер в SP_k, который однозначно получает алгоритм *A1* (место возможного

размещения рабочего интервала определяется размещением его предшественников и последователей в *HP* с использованием ярусной формы *HP*).

Выбор значений параметров $\langle YP \rangle$, позволяющих получить требуемое расписание алгоритмом восстановления *A1*, можно сделать, проведя соответствующую топологическую сортировку вершин требуемого расписания. Поскольку, в *P2* максимум может находиться *N* рабочих интервалов, то для получения любого допустимого варианта порядка может потребоваться максимум *N* различных значений параметров $\langle YP \rangle$, т.е. может потребоваться полная топологическая сортировка.

Кодирование решений выполняется следующим образом:

$Y \equiv Y_{task} \bigcup Y_{priority}$ – закодированное решение,

$Y_{task} \equiv \bigcup_{i=1}^{K} \langle YE \rangle_i$ – привязка,

$Y_{priority} \equiv \bigcup_{i=1}^{K} \bigcup_{j=1}^{N_i} \langle YP \rangle_j$ – приоритеты,

Здесь *K* – число процессов в *H*, N_i – число рабочих интервалов в *i*-ом процессе, $N = \sum_{i=1}^{K} N_i$ – число рабочих интервалов в *H*, $\bigcup$ – оператор конкатенации строк.

Параметр $\langle YE \rangle_i \in [1, \ldots, N] \subset Z$ содержит номер процессора, на котором выполняются рабочие интервалы *i*-го процесса, т.е. параметры $\langle YE \rangle$ однозначно определяют распределение рабочих интервалов по *SP* (привязку) и число процессоров в ВС. Параметры $\langle YE \rangle$ могут принимать значения от 1 до *K*. Если значения всех параметров $\langle YE \rangle$ равны, то все рабочие интервалы выполняются на одном процессоре, если все значения различны, то рабочие интервалы каждого процесса выполняются на своем процессоре. В силу ограничения 5, максимальное число непустых процессоров равно числу процессов *K* в *H*.

Параметры $\langle YP \rangle_j \subset Z$ используются алгоритмом восстановления расписания (для данного представления расписания – определение отношения полного порядка в каждом SP_i) в качестве приоритетов рабочих интервалов.

Операции преобразования решения

При параметрическом представлении расписания с использованием приоритетов операции преобразования решения могут быть введены как операции изменения значений параметров $\langle YE \rangle$ и $\langle YP \rangle$:

1) параметры $\langle YE \rangle$ могут принимать целочисленные значения в диапазоне *[1,…,K]*,
2) параметры $\langle YP \rangle$ могут принимать целочисленные значения в диапазоне *[1,…,L]*,
3) количество изменяемых параметров может изменяться от 1 до $N+K$.

Операции преобразования решений, удовлетворяющие условиям 1-3 при использовании алгоритма восстановления *A1*, будут получать решения удовлетворяющие ограничениям на корректность расписаний 1-5, что следует непосредственно из утверждения 1. Из теоремы 2 следует, что данные операции преобразования решения и алгоритм восстановления *A1* позволяют получить любой допустимый вариант решения.

4.5.3. Операции генетического алгоритма

Операция селекции.

Предложенная в работе [Костенко В.А., Смелянский Р.Л., Трекин А.Г. Синтез структур вычислительных систем реального времени с использованием генетических алгоритмов// Программирование, 2000., №5, С.63-72.] комбинированная операция селекции позволяет сочетать преимущество схемы пропорциональной селекции и схемы рулетки. В комбинированном варианте операции для вычисления целого числа потомков используется схема пропорциональной селекции, а для распределения остатка – схема рулетки. При этом в алгоритм добавлен параметр N_{best}, определяющий количество экземпляров лучшей строки, гарантированно остающихся в популяции.

Общая схема работы комбинированной операции селекции такова:

1. На этапе вычисления функции выживаемости выбирается строка популяции X'_{best} с наилучшим значением функции выживаемости F'_{best}, значение функции выживаемости сравнивается со значением функции выживаемости лучшей строки предыдущих итераций F_{best}. Если $F'_{best} > F_{best}$ или это первая итерация алгоритма, то запоминаем X'_{best} как лучшую строку $X_{best} = X'_{best}$.
2. В популяции вычисляется количество строк равных X_{best} и запоминается как N_{best_exist}.
3. По схеме пропорциональной селекции вычисляется количество потомков для каждой строки. Если количество строк в новой популяции равно N_{pop}, где N_{pop} – размер популяции, то выбирается (N_{best} - N_{best_exist}) худших строк в популяции (так как N_{best_exist} имеющихся лучших строк всегда перейдут в новую популяцию по

определению схемы пропорциональной селекции) и заменяется на копии лучшей строки X_{best} и операция селекции завершается, в противном случае перейти к 4.

4. По схеме рулетки распределить N_{pop} - N_{sel1} - (N_{best} - N_{best_exists}) потомков, где N_{sel1} – количество строк, полученное по схеме пропорциональной селекции, затем добавить в популяцию N_{best} - N_{best_exists} лучших строк X_{best}, в случае если данная величина отрицательна, то добавления не происходит.

Операция мутации и скрещивания.

Поскольку ограничения на изменения значений параметров предложенного в разделе 2.5.2. способа представления расписаний задаются в виде $a_i \le y_i \le b_i$, $y_i \in Z$, то могут быть использованы любые известные операции мутации и скрещивания, используемые при целочисленном кодировании решений (см. Приложение 1).

4.5.4. Функция выживаемости и критерий останова

Функция выживаемости может быть задана следующим образом:

$F(k_t, k_e)=C_1k_t+C_2k_e$, $C_1+C_2=1$, $C_i \le 0$ (i=1,2)

$$k_t = \begin{cases} T^{dir} / T, & \text{при } T^{dir} \le T \\ 1, & \text{при } T^{dir} > T \end{cases}$$

$$k_e = 1 - \frac{M(HP, HW0}{K}$$

В качестве критерия останова может быть использован любой из критериев приведенных в разделе 4.1.

4.6. Алгоритмы дифференциальной эволюции

Алгоритмы дифференциальной эволюции являются модификацией эволюционных алгоритмов, в которой предпринята попытка ослабить влияние способа кодирования решения на принципиальную работоспособность алгоритма и повысить скорость сходимости алгоритма к оптимуму.

Как и генетические алгоритмы, алгоритмы дифференциальной эволюции [Storn R., Price K. Minimizing the real functions of the ICEC'96 contest by differential evolution // IEEE Conference on Evolutionary Computation, Nagoya, 1996, pp. 842-844.] используют набор решений (популяцию) и на каждом шаге преобразовывают ее последовательным применением операций селекции, мутации и скрещивания. Основной особенностью алгоритма является использование

"дифференциальной" операции мутации, которая целенаправленно изменяет решения в текущей популяции.

Общую схему алгоритмов дифференциальной эволюции можно представить следующим образом.

1. *Сгенерировать случайным образом начальную популяцию* P^0 *, состоящую из* N_{pop} *допустимых (удовлетворяющих всем ограничениям) решений.*
2. *Вычислить целевую функцию для каждого элемента популяции* P^0.
3. *Положить популяцию* P^0 *текущей популяцией:* $P = P^0$.
4. *Изменить популяцию P посредством операции мутации:* $P^m = mutate(P)$;
5. *Выполнить операцию скрещивания над элементами измененной популяции* P^m *и текущей популяции P:* $P^c = crossover(P^m, P)$.
6. *Вычислить целевую функцию и функции ограничений для каждого элемента популяции* P^c.
7. *Выполнить операцию селекции:* $P^{new} = select(P^c, P)$.
8. *Положить популяцию* P^{new} *текущей популяцией:* $P = P^{new}$
9. *Если критерий останова не достигнут, перейти к шагу 4, иначе завершить работу.*

Операция мутации $P^m = mutate(P)$ порождает элементы популяции $P^m = \{X_1^m, X_2^m, ..., X_{N_{pop}}^m\}$ с помощью прибавления к элементам исходной популяции $P = \{X_1, X_2, ..., X_{N_{pop}}\}$ взвешенных разностей ("дифференциалов") других элементов популяции *P*. Элемент популяции P^m с номером *i* может вычисляться по следующим правилам:

- стандартная схема: $X_i^m = X_{r_1} + \mu \cdot (X_{r_2} - X_{r_3})$
- "жадная" схема: $X_i^m = X_i + \lambda \cdot (X_b - X_i) + \mu \cdot (X_{r_2} - X_{r_3})$

Здесь, $r_1, r_2, r_3 \in \{1,2,...,N_{pop}\}$ - случайно выбираемые номера элементов популяции *P* (попарно различные), X_b - наилучшее найденное на текущий момент решение, $\lambda, \mu > 0$ - настраиваемые параметры операции.

Отличие "жадной" схемы от стандартной заключается в целенаправленном смещении текущего решения в направлении наилучшего найденного на текущий момент решения (X_b), что заметно повышает скорость сходимости алгоритма (за счет некоторого ослабления глобально-поисковых свойств алгоритма).

Операция скрещивания $P^c = crossover(P^m, P)$ формирует популяцию $P^c = \{X_1^c, X_2^c, ..., X_{N_{pop}}^c\}$, *i*-й элемент которой $X_i^c = (x_{i,1}^c x_{i,2}^c ... x_{i,N}^c)$ представляет собой комбинацию элементов популяций P^m и P с тем же номером *i* (эти элементы обозначены ниже, соответственно, как $X_i^m = (x_{i,1}^m x_{i,2}^m ... x_{i,N}^m)$ и $X_i = (x_{i,1} x_{i,2} ... x_{i,N})$):

$$x_{i,j}^c = \begin{cases} x_{i,j}^m & \text{для } j = \langle S_i \rangle_N, \langle S_i + 1 \rangle_N, ..., \langle S_i + L_i - 1 \rangle_N \\ x_{i,j} & \text{иначе} \end{cases}$$

Здесь, S_i - целое число, случайно и равновероятно выбранное из интервала $[0, N-1]$; L_i - случайное целое число, выбранное из интервала $[0, N-1]$, при этом вероятность выбора значения *k* определяется формулой $\Pr(L_i = k) = CR^k$. Здесь $CR \in [0,1]$ - параметр операции; $\langle \cdot \rangle_N$ - операция взятия числа по модулю *N*.

Фактически, операция скрещивания в некоторой степени "нейтрализует" действие мутации, отменяя изменения определенных элементов вектора X_i. При этом малые значения L_i более вероятны - это означает, что последовательное применение операций мутации и скрещивания обеспечивают высокую вероятность малых изменений в текущих решениях, и меньшую (но не нулевую) вероятность больших изменений.

Операция селекции $P^{new} = select(P^c, P)$ обеспечивает выборку из текущей популяции *P* и модифицированной операциями мутации и скрещивания популяции P^c, новой текущей популяции P^{new}. Элемент модифицированной популяции P^c выживает, только если он удовлетворяет всем ограничениям задачи и улучшает значение целевой функции по сравнению с элементом текущей популяции с тем же номером:

$$X_i^{new} = \begin{cases} X_i^c, & \text{если } f(X_i^c) < f(X_i) \text{ и } \mathrm{X_i^c} \in \mathrm{S} \vee g_j(X_i^c) \le 0, \forall j = 1,2,..,k \\ X_i, & \text{иначе} \end{cases}$$

Критерий останова. Возможен один или некоторая комбинация следующих способов останова:

- выполнение алгоритмом априорно заданного числа итераций;
- выполнение алгоритмом априорно заданного числа итераций без улучшения целевой функции;
- достижение некоторого априорно заданного значения целевой функции.

5. Муравьиные алгоритмы

5.1. Концепция построения алгоритмов (биологическая модель)

Интересным результатом кооперативного поведения биологических муравьев является нахождение кратчайшего маршрута от источника пищи к гнезду. Рассмотрим, как кооперативное поведение биологических муравьев позволяет найти кратчайший маршрут к источнику пищи. Пусть гнездо соединено с источником пищи двумя ребрами разной длины (рис.5.1.). Муравьи при прохождении маршрута откладывают феромоны и чем больше муравьев прошло по ребру, тем выше концентрация феромонов на этом ребре. Чем выше концентрация феромонов на ребре, тем выше вероятность, что муравей пойдет по нему. Изначально вероятности выбора маршрутов равны. Муравьи, выбравшие короткое ребро, возвращаются быстрее. Количество феромонов на коротком ребре растет быстрей и следовательно повышается вероятность его выбора, т.е. через некоторое время по нему будет проходить большинство муравьев. Со временем феромоны испаряются, что позволяет муравьям адаптировать поведение под изменение внешней среды.

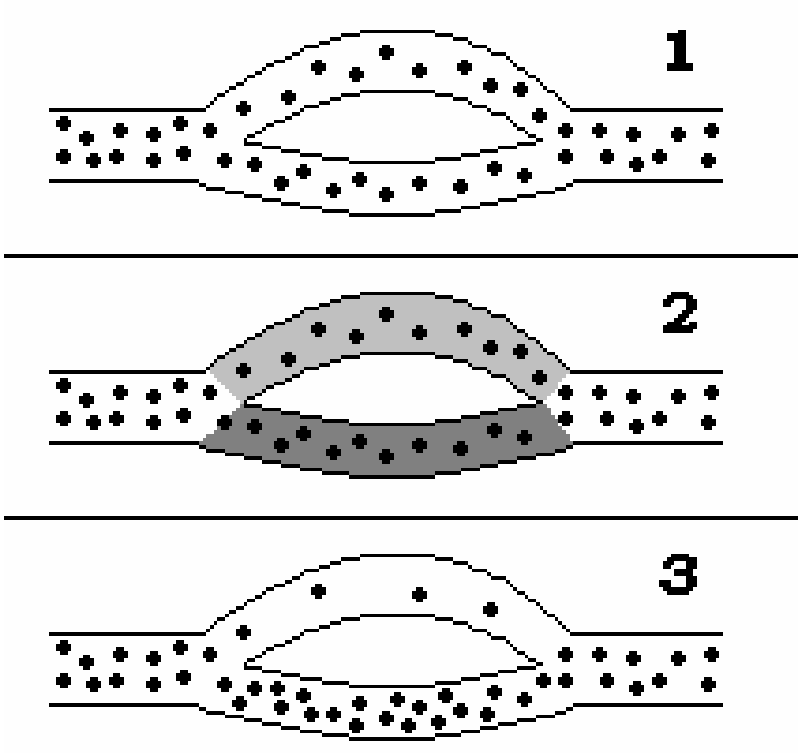

Рис. 5.1.

Таким образом, при решении задачи нахождения кратчайшего пути муравьиные алгоритмы позволяют автоматически настраиваться на пример задачи (заданы конкретные значения исходных данных) путем дополнительной разметки исходных данных, которая используется для построения решения на каждой итерации алгоритма и уточняется по мере увеличения числа итераций.

5.2. Общая схема работы муравьиных алгоритмов

Общую схему работы муравьиных алгоритмов можно представить в следующем виде:

1. Задание начального количества феромона на ребрах графа, количества и начального положения муравьев.
2. Построение муравьями пути (каждый муравей строит путь независимо от остальных).
3. Вычисление целевой функции для каждого пути.
4. Обновление количества феромона на ребрах.
5. Если условие останова не выполнено, то переход к п.2.

Муравьи строят маршруты, последовательно переходя от вершины к вершине, руководствуясь при этом определенным алгоритмом для определения списка вершин, доступных на данном этапе (например, при решении задачи коммивояжера используется табу-список недоступных вершин, в который добавляются пройденные муравьем вершины). Выбор очередной вершины зависит от количества феромонов и значения локальной эвристической функции на ребрах, ведущих из текущей вершины в доступные. Эти значения определяют вероятность перехода в ту или иную вершину, после чего очередная вершина определяется по правилу рулетки.

В конце каждой итерации для каждого найденного маршрута вычисляется значение целевой функции. Оно используется для вычисления количества феромона, добавляемого на ребра, входящие в данный маршрут.

Операция построение муравьем пути. Муравей строит путь, переходя из одной вершины в другую. Пройденные муравьем вершины добавляются в табу-список (память муравья), чтобы избежать повторного их посещения. Вероятность перехода муравья из i-й вершины в j-ю зависит от количества феромона на данном ребре, значения локальной целевой функции на ребре и состояния табу-списка. Вероятность перехода k-го муравья из i-й вершины в j-ю на t-й итерации алгоритма рассчитывается по следующей формуле:

$$P_{ij,k}(t)=\begin{cases}\dfrac{(\tau_{ij}(t))^{\alpha}\cdot(\eta_{ij}(t))^{\beta}}{\sum\limits_{l\in J_k}(\tau_{il}(t))^{\alpha}\cdot(\eta_{il}(t))^{\beta}}, j\notin L_k\\ 0, j\in L_k\end{cases}$$

Здесь $\tau_{ij}(t)$ - количество феромона на ребре (i,j), $\eta_{ij}(t)$ - значение локальной целевой функции на ребре (i,j), $\alpha\geq 0$ и $\beta\geq 0$ - параметры алгоритма, определяющие важность феромонного следа и локальной целевой функции, L_k – множество вершин, включенных в табу-список муравья k.

Операция обновление количества феромона на ребрах. После того, как все муравьи завершили построение путей, обновляется количество феромона на ребрах:

$$\tau_{ij}(t+1)=(1-p)\cdot\tau_{ij}(t)+\sum_{k=1}^{m}\Delta\tau_{ij,k}(t)$$

$$\Delta\tau_{ij,k}(t)=\begin{cases}F(T_k(t)),(i,j)\in T_k(t)\\ 0,(i,j)\notin T_k(t)\end{cases}$$

Здесь $T_k(t)$ – путь, построенный k-м муравьем, а $F(T)$ – целевая функция, определяющая качество пути, m – количество муравьев, $p\in[0,1]$ – коэффициент испарения феромонов. Испарение феромонов вводится для избегания попадания алгоритма в локальный оптимум, когда первый найденный путь с относительно хорошим значением целевой функции становится единственно значимым.

Основными задачами, которые необходимо решить для того, чтобы использовать муравьиный алгоритм для решения конкретной задачи условной оптимизации, являются:

1. Сведение задачи условной оптимизации к задаче нахождения на графе маршрута, обладающего определенными свойствами.
2. Задание локальной эвристической функции на ребрах графа.
3. Определение алгоритма построения маршрута муравьем (например, определение правила формирования табу-списка вершин).

5.3. Модификации муравьиных алгоритмов

Модификации муравьиных алгоритмов были разработаны с целью устранить следующие основные недостатки базового алгоритма:

- возможность потери наилучшего найденного решения;
- низкой скоростью сходимости к оптимуму из-за приблизительно равного вклада в обновление феромонов хороших решений из различных областей пространства решений;
- хранением в памяти алгоритма (количество феромона на ребрах) заведомо не перспективных решений.

Решить эти проблемы путем выбора соответствующих значений коэффициента испарения феромонов (p) и параметров α, β, определяющих важность феромонного следа и локальной целевой функции удается не для всех задач.

5.3.1. Максиминный алгоритм

Максиминная схема алгоритма позволяет решить проблему быстрой сходимости муравьиного алгоритма к локальному оптимуму. Для этого к базовой схеме добавляются три правила:

1. На каждой итерации феромон добавляется только на лучшее из решений, среди найденных на данной итерации;
2. Ограничивается минимальное и максимальное количество феромона на ребрах: $\tau_{ij} \in [\tau_{\min}; \tau_{\max}]$;
3. Изначально количество феромона на всех ребрах равно τ_{max}.

Изначально одинаковое количество феромона на всех ребрах уменьшает вероятность выбора одного и того же маршрута разными муравьями, а условие $\tau_{ij} \in [\tau_{\min}; \tau_{\max}]$ не позволяет одному относительно хорошему решению доминировать над другими. Таким образом, эти ограничения позволяют разнообразить находимые алгоритмом маршруты и избежать попадания в локальный оптимум. Кроме того, для дополнительного расширения области поиска в максиминном алгоритме применяется механизм «сглаживания следов»: количество откладываемого на ребрах феромона пропорционально величине $\tau_{\max} - \tau_{ij}$.

5.3.2. Модификация с поглощением феромона

Модификация с поглощением феромона – ещё один метод, применяющийся для расширения области поиска решений. Проходя по ребру при построении пути, муравей поглощает часть феромона на этом ребре:

$\tau_{ij} = \tau_{ij} \cdot (1 - d)$, где d – доля поглощаемого феромона.

Таким образом, ребро, по которому уже прошел один из муравьев, сразу теряет свою привлекательность для других муравьев, что заставляет их выбирать другие ребра.

5.3.3. Совместное использование с алгоритмами локального поиска

Суть подхода заключается в том, что на каждой итерации муравьиного алгоритма алгоритмы локального поиска пытаются улучшить найденные решения. Обычно применяются локально-оптимальные алгоритмы.

Для задачи коммивояжера, часто используются алгоритмы локального поиска, которые улучшают маршрут заменой заданного количества ребер.

5.4. Муравьиные алгоритмы для решения задачи построения статико-динамических расписания (два способа представления задачи)

Математическая постановка задачи приведена в разделе1.4.2.

5.4.1. Первый способ сведения задачи построения статико-динамического расписания к задаче нахождения на графе маршрута

Построим полносвязный граф $G=<N,A>$, где:

- $N=\{n_i | i \in [1..n]\} \cup \{O\}$ – множество вершин;
- $A=\{(n_i,n_j) | i,j \in [1..n], i \neq j\} \cup \{(O,n_i) | i \in [1..n]\} >$ – множество ребер.

Каждой вершине графа соответствует одна из размещаемых работ. Кроме того, добавляется еще одна вершина O, соответствующая началу расписания.

Муравьиный алгоритм строит маршруты, в которых все вершины включены ровно один раз. Каждому такому маршруту соответствует последовательность работ, такая что, в нее включены все работы из исходно заданного набора и каждая работа включена в данную последовательность один раз. Опишем алгоритм для построения расписания по такой последовательности работ. Пусть дана последовательность работ $SL=\{a_{ik} | a_{ik} \in SW; i,k \in [1..n]\}$. Первый индекс означает номер работы, второй - номер места работы в последовательности. Построим расписание SP по следующему алгоритму:

4. Инициализация расписания: $Time=0$ – текущая длина расписания; $k=1$ – номер места размещаемой работы в SL; $SW^0=\varnothing$ – список работ в добавляемом окне;

5. Установка начальных параметров окна: $S=\max(Time,s_{ik})$, $F=f_{ik}$ – границы добавляемого окна; $T=\Delta 2$ – минимальная необходимая длина окна с учетом добавленных работ; $R=r_{ik}$ – раздел для работ в окне;
6. Обновление значений параметров окна с учетом новой добавляемой работы: $S'=\max(S,s_{ik})$, $F'=\min(F,f_{ik})$, $T'=T+t_{ik}$;
7. Добавление работы в текущее окно: если $r_{ik}=R$, $T'\le F'-S'$ (условия корректности не нарушаются), то: $S=S'$, $F=F'$, $T=T'$, $SW^0=SW^0\cup\{a_{ik}\}$, $k=k+1$, если $k\le n$ (список SL еще не пройден), переход к п.3;
8. Проверка возможности добавления работы в новое окно: если $f_{ik}-F-\Delta 1<t_{ik}$, то $k=k+1$, если $k\le n$ переход к п.3 – данная работа не может быть размещена ни в текущее окно, ни в новое окно (дальше в списке еще могут быть работы, которые можно разместить в текущее окно);
9. Закрытие окна: $F=S+T$ – устанавливаем время закрытия окна минимально возможным;
10. Добавление данного окна в расписание: $SP=SP\cup\{<S,F,SW^0>\}$;
11. Пересчет длины расписания: $Time=F+\Delta 1$, $k=k+1$;
12. Если $k \le n$ (список еще не пройден), переход к п.2.

Расписание, построенное при помощи данного алгоритма, будет удовлетворять всем условиям корректности:

13. Условие корректности 1 выполняется, т.к. каждая вершина встречается в построенном маршруте ровно один раз, и соответствующая ей работа размещается лишь в одно окно;
14. Условие 2 выполняется, т.к. $S'=\max(S,s_{ik})\ge s_{ik}$; $F'=\min(F,f_{ik})\le f_{ik}$ (п.3);
15. Выполнение условий 3 и 5 обеспечивается проверками в п.4 алгоритма – если ограничения нарушаются, работа не размещается в данное окно;
16. Условие 4 выполняется, т.к. $S_{i+1}\ge Time$ (п.2), где $Time=F_i+\Delta 1$ (п.8).

Таким образом, при помощи описанного алгоритма по заданной последовательности работ однозначно строится корректное расписание. Фактически, данный алгоритм устанавливает соответствие между пространством расписаний и пространством последовательностей работ, которое, в свою очередь, является пространством поиска для муравьиного алгоритма.

Недостатком данного подхода является то, что множество корректных расписаний содержит больше элементов, чем множество последовательностей работ, т.е. существуют расписания, для которых не существует соответствующих им цепочек работ. Причина

заключается в том, что последовательность работ не задает однозначно распределение работ по окнам. Более того, можно привести пример задач, для которых оптимальное расписание не будет принадлежать к пространству поиска муравьиного алгоритма. Пример исходных данных такой задачи: $SW=\{a_1=<0,11,4,1>,a_2=<4,11,2,1>,a_3=<4,11,2,1>\}$, $\Delta1=\Delta2=1$ (рис.5.2.). Прямоугольниками показаны директивные интервалы, заштрихованными областями – время выполнения работ. Раздел у всех работ одинаковый.

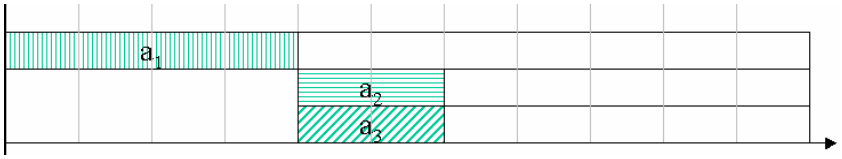

Рисунок 5.2. Исходные данные.

На рис. 5.3. показаны все возможные расписания, которые могут быть построены приведенным алгоритмом по различным маршрутам в графе. Все эти расписания не являются оптимальными.

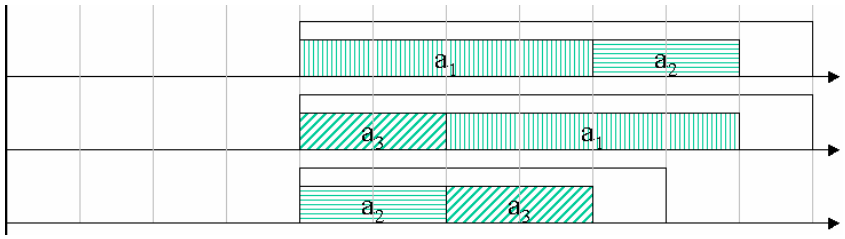

Рисунок 5.3. Построенные расписания.

При этом расписание, в котором все работы размещены, существует (рис.5.4.).

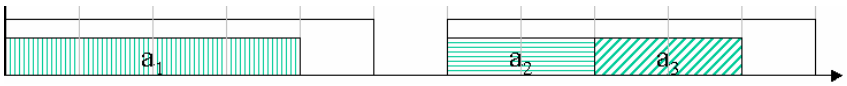

Рисунок 5.4. Оптимальное расписание.

Можно выдвинуть гипотезу, что не существует детерминированного полиномиального алгоритма A построения расписания по маршруту в графе G такого, что для $\forall(SW,\Delta1,\Delta2):\exists SL:A(SL)=SP^0$, где SP^0 – оптимальное расписание. Т.е. не существует детерминированного полиномиального алгоритма, который для любой частной задачи мог бы построить оптимальное расписание по одному из возможных маршрутов. По крайне

мере, пока такой алгоритм найти не удалось, однако нет и доказательства, что такого алгоритма не существует.

В качестве локальной целевой функции на ребрах графа *G* используется следующая функция: $\eta_{ij} = \begin{cases} 1/(\theta+1), \text{ если } \theta \geq 0 \\ 0, \text{ если } \theta < 0 \end{cases}$, где $\theta = (f_j - t_j) - (s_i + t_i)$ – разница во времени между минимальным временем завершения *i*-й работы и максимальным временем начала *j*-й работы. Если $\theta < 0$, то *j*-я работа не может идти в расписании после *i*-й. В противном случае, чем меньше значение θ, тем меньше максимальный возможный промежуток в расписании между работами, соответствующими вершинам *i* и *j*, и тем больше значение локальной целевой функции.

Целевая функция для оценки качества маршрута задается отношением количества работ, размещенных в расписание без нарушения условий корректности, к количеству исходных заданных работ. Значение данной функции вычисляется алгоритмом, строящим расписание по маршруту.

5.4.2. Второй способ сведения задачи построения статико-динамического расписания к задаче нахождения на графе маршрута

Чтобы устранить недостаток первого подхода, предлагается использовать измененное представление задачи в виде поиска маршрута на графе, в котором каждой работе будет соответствовать не одна, а две вершины. Появление в маршруте первой из этих вершин будет означать размещение работы без закрытия текущего окна. Другая вершина будет соответствовать размещению работы с закрытием окна. Соответственно, в маршруте может присутствовать только одна вершина из каждой такой пары.

Построим граф $G'=<N',A'>$, где

- $N'=\{n_i^+, n_i^- | i \in [1..n]\} \cup \{O\}$ – множество вершин; вершина n_i^+ соответствует размещению работы без закрытия текущего окна, n_i^- – размещению работы с закрытием окна. Вершина *O* является началом всех маршрутов.
- $A'=\{(n_i^+, n_j^-),(n_i^+, n_j^+),(n_i^-, n_j^-) | i,j \in [1..n], i \neq j\} \cup \{(O, n_i^+),(O, n_i^-) | i \in [1..n]\}$ – множество ребер. Вершины, соответствующие одной работе, ребрами не связаны.

Алгоритм построения расписания по заданной последовательности работ полностью аналогичен предыдущему, за исключением того, что в п.4 при размещении работы, соответствующей вершине n_j^- текущее окно закрывается принудительно (происходит переход к п.6). Расписание, построенное при помощи такого алгоритма, также будет удовлетворять условиям корректности.

Заметим, что расписания, которые строятся приведенными алгоритмами (как базовым, так и модифицированным), помимо условий корректности, обладают следующими свойствами:

- $\sum_{a_j \in SW_i} t_j + \Delta 2 = F_i - S_i$ - длина временного интервала окна является минимально допустимой.
- $S_i = \begin{cases} \max\limits_{a_j \in SW_i}(s_j), i = 1 \\ \max\limits_{a_j \in SW_i}(s_j, F_{i-1} + \Delta 1), i > 1 \end{cases}$ - время открытия окна является минимальным, без нарушения ограничений 2 и 4.

Покажем, что модифицированный алгоритм всегда может построить оптимальное расписание по некоторому пути в *G'*.

Теорема. Для любого корректного расписания *SP* можно найти такую последовательность вершин *SL*, что по данной последовательности модифицированный алгоритм восстановления расписания построит такое корректное расписание SP^*, что $\forall i \in [1..k]: SW_i^* = SW_i$, количество окон в SP^* не меньше количества окон в *SP*.

Доказательство: построим искомую последовательность $SL=\{n_i^p | n_i^p \in N', i \in [1..n], p \in \{+,-\}\}$ следующим образом: $SL=SL_1 \cup SL_2 \cup \ldots \cup SL_m$, где SL_i – последовательность из вершин, соответствующих работам из *i*-го окна такая, что все вершины, за исключением последней соответствуют размещению работы без закрытия окна. Т.е., фактически, данная последовательность состоит из размещенных работ, упорядоченных по времени открытия окна, в которое они размещены. Вершины, соответствующие работам, не размещенным в *SP*, добавляются в конец последовательности *SL* в произвольном порядке (добавляется одна произвольная вершина из двух). При этом в последовательности *SL* присутствует только одна вершина, соответствующая каждой работе.

Докажем совпадение списков работ в окнах расписания SP^* и расписания *SP* по индукции.

17. *Базис индукции*: Т.к. все вершины, соответствующие работам из первого окна, стоят в последовательности вершин друг за другом, не прерываются вершиной с закрытием окна, то они будут размещаться в одно и то же окно. Т.к. исходное расписание является корректным, это возможно без нарушения условий корректности 1-5. Следовательно, списки работ в первом окне в *SP* и

SP^* будут совпадать (поскольку список вершин, соответствующих работам из первого окна, заканчивается вершиной с закрытием окна, в данное окно не попадут «лишние» работы). Время открытия и закрытия первого окна могут различаться в SP^* и SP, но при этом из свойств (1) и (2) следует, что $S^*_1 \le S_1$, $F^*_1 \le F_1$.

18. *Шаг индукции*: $F^*_{i-1} \le F_{i-1}$, и, следовательно (из свойства (2)), $S^*_i \le S_i, F^*_i \le F_i$; далее повторяя предыдущие рассуждения, получаем, что корректность i-го окна в SP гарантирует корректность соответствующего окна в SP^* и совпадение списка работ, размещенных в них.

Неразмещенные в SP работы могут оказаться добавлены только в новые окна, т.к. последовательность вершин, соответствующих размещенным в SP работам, заканчивается вершиной с закрытием окна.

Теорема доказана.

Из утверждения теоремы также следует, что значение целевой функции для расписания SP^* не меньше, чем для SP. Таким образом, доказано, что для любого корректного расписания SP существует последовательность вершин графа G' такая, что модифицированный алгоритм способен построить корректное расписание, по крайней мере, не хуже данного с точки зрения целевой функции.

Следствие. Если SP^0 корректное оптимальное расписание для множества работ SW, то существует такая последовательность вершин SL, что модифицированный алгоритм построит по ней корректное оптимальное расписание. Построенное расписание будет совпадать с расписанием SP^0 по количеству размещенных в нем работ, количеству окон, по множеству работ выполняемых внутри каждого окна, но может отличаться по времени открытия и закрытия окон.

С материалами данного раздела можно ознакомиться в работах:

3. Балаханов В.А., Костенко В.А. Способы сведения задачи построения статико-динамического однопроцессорного расписания для систем реального времени к задаче нахождения на графе маршрута // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007. – С.148-156.
4. Балаханов В.А., Кокарев В. Муравьиный алгоритм построения статико-динамических расписаний и исследование его эффективности// Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2008.

5.5. Муравьиный алгоритм для решения задачи построения расписания обменов по шине с централизованным управлением

Будет прочитан на основе материалов статей:

1. Костенко В.А. Алгоритмы построения расписаний для одноприборных систем, входящих в состав систем реального времени// Методы и средства обработки информации: Третья Всероссийская научная конференция. Труды конференции. - М.: Издательский отдел факультета ВМиК МГУ имени М.В. Ломоносова; МАКС Пресс, 2009. - С.245-258.
2. Balashov V.V., Balakhanov V.A., Kostenko V.A., Smeliansky R.L., Kokarev V.A., Shestov P.E. A technology for scheduling of data exchange over bus with centralized control in onboard avionics systems // Proc. Institute of Mechanical Engineering, Part G: Journal of Aerospace Engineering. – 2010. – Vol. 224, No. 9. – P. 993–1004.

6. Алгоритмы, использующие аппроксимирующую модель целевой функции

Этот класс алгоритмов представляет особый интерес для задач с "дорогой" целевой функцией. Задачи с "дорогой" целевой функцией характеризуются большими вычислительными затратами на вычисление целевой функции. Соответственно эффективные алгоритмы должны минимизировать количество ее вычислений для нахождения оптимального решения X или поддерживать возможность параллельной работы.

Для простоты изложения алгоритмов оптимизации, использующих аппроксимирующую модель целевой функции будем полагать, что вычисление функций ограничений $g_i(x)$ не требует существенных затрат ресурсов (хотя все рассмотренные ниже подходы могут быть обобщены и на этот случай). Также будем предполагать, что все оптимизируемые параметры представимы в виде вещественных чисел (т.е. что область поиска $S \subseteq R^N$). Расширения рассмотренных ниже методов на случай разнотипных/нечисловых оптимизируемых параметров в принципе возможны, однако, как правило, они не обладают достаточной общностью и строятся индивидуально для каждого класса задач.

В процессе работы алгоритма с использованием аппроксимирующей модели все вычисленные значения целевой функции (т.е. пары вида $(X, f(X))$ заносятся в специальную базу данных (БД). БД используется для построения модели M целевой функции $f(X)$ на некотором подмножестве S^{cur} множества S допустимых значений параметров (на текущей области поиска). Модель M аппроксимирует значение функции $f(X)$ на всем множестве S^{cur}, а также, при необходимости, оценивает достоверность аппроксимации (математическое ожидание ошибки аппроксимации) и производные функции $f(X)$. В процессе работы алгоритма модель периодически уточняется вместе с пополнением БД. На основании анализа модели M алгоритм принимает решения об уточнении текущей области поиска S^{cur}, и о направлении поиска.

Базовая схема алгоритмов оптимизации с использованием аппроксимирующей модели может быть представлена следующим образом.

1. *Спланировать и провести серию начальных испытаний (вычислений значений функции $f(X)$) на множестве точек $X^T = \{X_i\}, i = 1,2,...,s_{init}$.*

   Инициализировать БД: $T = \{(X, f(X)\}, X \in X^T$.

2. *Установить текущую область поиска* $S^{cur} = S$.
3. *Построить аппроксимационную модель M целевой функции* $f(X)$ *на основании информации, содержащейся в БД T. Модель M состоит из:*
   - *функции оценки значения целевой функции* $\tilde{f}(X) : S^{cur} \to R$
   - *[опционально] функции оценки мат. ожидания квадрата ошибки аппроксимации* $\tilde{\sigma}^2(X) = E\left(\tilde{f}(X) - f(X)\right)^2$, $\tilde{\sigma}^2(X) : S^{cur} \to R$
   - *[опционально] функции оценки значений производных целевой функции.*
4. *Выбрать точку следующего испытания посредством решения т.н. задачи оптимизации на модели:*

$$\min_{X \in S^{cur}} I(X, M, T)$$
$$g_i(X) \le 0, \quad i = 1,2,...,k \qquad (6.1.)$$

Здесь, $I(X, M, T)$ *- функция выбора точки след. испытания. Значение* $I(X, M, T)$ *позволяет оценить (используя модель M), насколько желательным является получение информации о значении "настоящей" целевой функции в точке* $X \in S^{cur}$ *в плане решения исходной задачи (1). Вычисление значения* $I(X, M, T)$ *не требует вычисления значения "настоящей" целевой функции* $f(X)$. *Различные способы построения* $I(X, M, T)$ *будут рассмотрены ниже. Точку, доставляющую минимум функции* $I(X, M, T)$, *далее будем обозначать как* X^*.

5. *Вычислить значение "настоящей" целевой функции* $f(X)$ *в точке* X^*.
6. *Дополнить БД :* $T = T \cup (X^*, f(X^*))$
7. *Уточнить область поиска* S^{cur}.
8. *Обновить модель* M *с учетом дополненной БД и уточненной области поиска.*
9. *Если не выполнен критерий завершения, перейти к шагу 4.*
10. *Решением задачи (1) положить наилучшее из решений, сохраненных в БД:* $X_{\min} = \arg\min_{X \in T} f(X)$, $f_{\min} = \min_{X \in T} f(X)$.

На практике производительность алгоритма оптимизации с использованием аппроксимирующей модели сильно зависит от выбора его параметров, в число которых входят:

- алгоритм планирования начальных испытаний;
- метод построения модели *M*;

- стратегия выбора точки след. испытания (определяющая способ построения функции $I(X, M, T)$);
- алгоритм решения задачи оптимизации на модели (6.1.);
- алгоритм уточнения области поиска;
- критерий завершения.

При удачном выборе параметров алгоритм часто бывает способен найти решение задачи (1.1.) с удовлетворительной точностью за небольшое (порядка нескольких десятков) число итераций, и, соответственно, небольшое число вычислений целевой функции.

Планирование начальных испытаний. На данном этапе основной целью является получение наиболее общей информации о целевой функции, достаточной для построения модели *M* "в первом приближении". Значение целевой функции вычисляется в небольшом числе точек, распределенных по области *S*. На практике для выбора точек X_i, в которых должна быть вычислена целевая функция, часто используются следующие подходы:

- случайное распределение заданного числа точек по области *S*;
- покрытие области *S* равномерной сеткой;
- распределение точек X_i с использованием классических методов планирования эксперимента (план "латинского гиперкуба", частичный/полный факториальный план, и т.д.).

Метод построения модели M. Для построения модели *M* могут использоваться многие известные алгоритмы аппроксимации или интерполяции непрерывных функций (например, аппроксимация квадратичной формой, аппроксимация с помощью нейронной сети). Результаты измерений целевой функции, занесенные в БД $T = \{(X_i, f(X_i)\}, X_i \in X^T$, используются для формирования обучающей выборки (множества узлов аппроксимации или интерполяции).

Для успешного применения алгоритма оптимизации с использованием аппроксимирующей модели необходимым условием является адекватность модели целевой функции. Поэтому в процессе работы алгоритма необходимо постоянно (на каждой итерации) контролировать качество модели *M*. Для этого применяются такие методы, как использование тестового подмножества обучающей выборки, кросс-проверка. При невозможности обеспечить достаточное качество оценки в рамках выбранного метода построения модели, алгоритм выполняет одну из следующих корректирующих операций:

- "переключение" на использование альтернативного метода построения модели;
- нелинейная трансформация множества значений целевой функции;

- проведение серии дополнительных испытаний (вычислений целевой функции) с целью получения дополнительной информации о целевой функции.

Стратегия выбора точки следующего испытания. Наиболее распространенные подходы описаны ниже.

Простейшая стратегия заключается в поиске точки, для которой оценка $\widetilde{f}(X)$ значения целевой функции минимальна. Функция выбора точки следующего испытания в этом случае принимает следующий вид: $I(X,M,T)=\widetilde{f}(X)$. Недостатком данной стратегии является высокая вероятность "зацикливания" алгоритма, что происходит, если для точки X^*, доставляющей минимум функции $\widetilde{f}(X)$ и , тем самым, являющейся решением задачи (4), уже существует запись в БД (т.е. если значение "настоящей" целевой функции $f(X^*)$ уже было вычислено ранее). В этом случае повторное вычисление значения $f(X^*)$ не дает оснований для уточнения модели M или области поиска S^{cur}, и последующие итерации алгоритма не приносят новых результатов. При этом точка X^* даже не обязательно является точкой локального минимума функции $f(X)$. Способом избежать зацикливания является введение в функцию $I(X,M,T)$ аддитивного "поощрения" за отдаленность точки X от точек, помещенных в БД: $I(X,M,T)=\widetilde{f}(X)-\alpha\cdot d(X,T)$, где $d(X,T)=\min\limits_{X'\in T}\|X-X'\|$, и $\alpha>0$ - параметр (величина поощрения).

Более надежной является стратегия, предложенная в рамках *P*-алгоритма [A. Zilinskas. Axiomatic characterization of a global optimization algorithm and investigation of its search strategy. Operations Research Letters, 4(6):35–39, 1985.]. Пусть $f_{\min}=\min\limits_{X\in T}f(X)$ - наименьшее значение целевой функции, найденное на текущий момент. Идея *P*-алгоритма заключается в поиске точки, для которой с максимальной вероятностью значение целевой функции будет меньше чем $f_{\min}$ как минимум на некоторую пороговую величину ε:

$$\text{Prob}(f(X)<f_{\min}-\varepsilon)\to\max \qquad (6.2.)$$

Для вычисления вероятности (6.2.) используются оценка $\widetilde{f}(X)$ значения целевой функции и оценка $\widetilde{\sigma}(X)$ математического ожидания ошибки аппроксимации $\varepsilon(X)=\left|f(X)-\widetilde{f}(X)\right|$ (стандартное отклонение для оценки $\widetilde{f}(X)$). Предполагается, что для любого $X'\in S^{cur}$,

- оценка $\widetilde{f}(X')$ является несмещенной: $E\varepsilon(X')=0$, и
- ошибка аппроксимации $\varepsilon(X')$ имеет нормальное распределение.

В этом случае распределение значений целевой функции в точке $X' \in S^{cur}$ может быть описано нормальным распределением: $f(X') \approx N(\tilde{f}(X'), \tilde{\sigma}(X'))$. В данных предположениях вероятность в формуле (6.2.) может быть вычислена явно, и функция выбора точки след. испытания принимает следующий вид:

$$I(X, M, T) = -\Phi\left(\frac{f_{\min} - \varepsilon_f - \tilde{f}(X)}{\tilde{\sigma}(X)}\right), \qquad (6.3.)$$

где $\Phi(\cdot)$ - функция распределения стандартного нормального распределения $N(0,1)$.

Можно показать, что задача минимизации функции (6.3.) эквивалентна задаче минимизации следующей функции:

$$I(X, M, T) = -\frac{\tilde{\sigma}(X)}{\left[\tilde{f}(X) - f_{\min} + \varepsilon_f\right]^2} \qquad (6.4.)$$

Рассмотрим свойства функции (6.4.). Пусть X^T - множество точек, значения целевой функции для которых известны (есть записи вида $\{(X_i, f(X_i))\} \in T$, где $X_i \in X^T$). Если модель *M* целевой функции строится посредством точной интерполяции (например, с использованием DACE модели [DaceMod]), то для $X \in X^T$ выполняется $\tilde{f}(X) = f(X)$ и $\tilde{\sigma}(X) = 0$, из чего следует $I(X_i, M, T) = 0$. С другой стороны, для $X \notin X^T$ выполняется $\tilde{\sigma}(X) > 0$, из чего следует $I(X, M, T) < 0$. Таким образом, функция $I(X, M, T)$ достигает минимума в некоторой точке $X^* \notin X^T$, что предотвращает повторные вычисления целевой функции и "зацикливание" (исключая вырожденный случай $\tilde{\sigma}(X) \equiv 0$).

Недостатком *P*-алгоритма является сложность выбора порога ε, который непосредственно влияет на скорость сходимости алгоритма и сложность минимизации функции (7). Один из возможных подходов к выбору ε заключаются в применении динамической адаптации. На разных итерациях алгоритма используются разные значения порога ε_f, для оценки пригодности конкретного значения ε_f^* анализируются данные, собранные по предыдущим итерациям алгоритма (предсказанная вероятность (5) наступления события $f(X^*) < f_{\min} - \varepsilon_f^*$, и факт наступления/ненаступления данного события, проверяемый апостериорно, после вычисления значения $f(X^*)$).

На рис. 6.1. приведена иллюстрация работы P-алгоритма. Критерий выбора точки испытания - максимум вероятности (5) (черная линия).

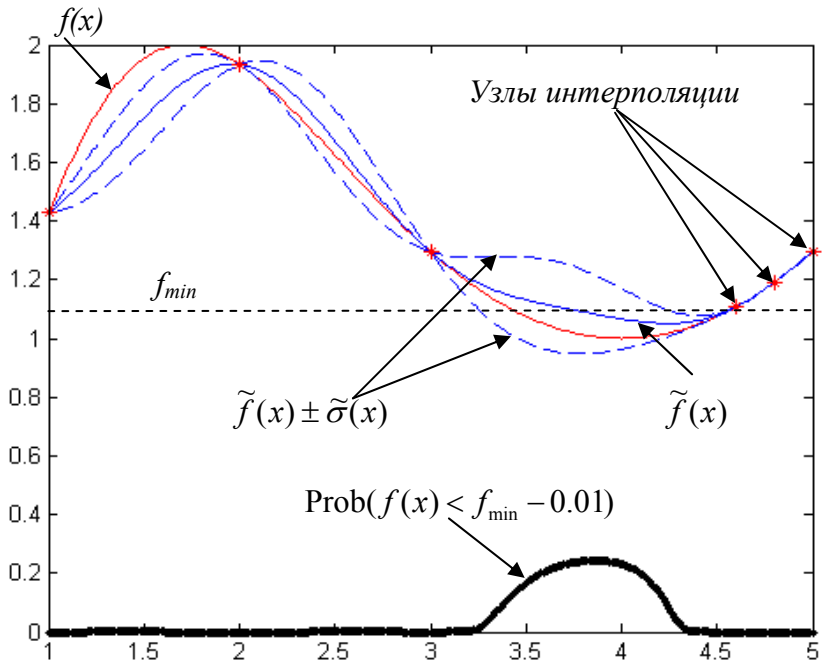

Рис.6.1. Иллюстрация работы P-алгоритма (красные линии - моделируемая функция, синие линии - DACE интерполяция, черная линия - вероятность (5)) .

Стратегия максимального ожидаемого улучшения [J. Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. Journal of Global Optimization, 4:347–365, 1994.]. Обозначим $\mathrm{Imp}(X) = \max\{f(X) - f_{\min}, 0\}$ - величину улучшения наименьшего значения целевой функции, найденного на текущий момент, которое может произойти после проведения испытания в точке X. Стратегия максимального ожидаемого улучшения заключается в поиске точки, для которой математическое ожидание ожидаемого улучшения максимально:

$$\mathrm{E}[\mathrm{Imp}(X)] \to \max \qquad (6.5.)$$

Вычислить значение $\mathrm{Imp}(X)$ возможно при принятии тех же предположений о модели M, что и в случае с P-алгоритмом (несмещенность оценки $\tilde{f}(X')$ и нормальное распределение ошибки аппроксимации $\varepsilon(X')$). Функция выбора точки след. испытания принимает следующий вид:

$$I(X,M,T) = \begin{cases} -\left((f_{\min} - \tilde{f}(X)) \cdot \Phi\left(\frac{f_{\min} - \tilde{f}(X)}{\tilde{\sigma}(X)}\right) + \tilde{\sigma}(X) \cdot \phi\left(\frac{f_{\min} - \tilde{f}(X)}{\tilde{\sigma}(X)}\right) \right), & \tilde{\sigma}(X) > 0 \\ 0, & \tilde{\sigma}(X) = 0 \end{cases}$$

Здесь, $\Phi(\cdot)$ и $\phi(\cdot)$ - функции распределения и плотности стандартного нормального распределения $N(0,1)$.

Как для *P*-алгоритма, так и для стратегии максимального ожидаемого улучшения, при использовании точной интерполяции при построении модели *M* минимум функции $I(X,M,T)$ достигается в точке $X^* \notin X^T$.

Стратегия достижения наибольшей гладкости (наименьшей "холмистости") интерполяционной функции [D.R. Jones, M. Schonlau, and W.J. Welch. E_cient global optimization of expensive black-box functions. Journal of Global Optimization, 13(4):455–492, 1998.]. Пусть $f_a < f_{\min}$ - некоторая априорная оценка минимального достижимого значения функции $f(X)$, $X^T = \{X_i\}$ - множество точек, для которых значения целевой функции $f(X_i)$ известны, $\widetilde{f}(X)$ - интерполяционная оценка значения целевой функции $f(X)$ на множестве S^{cur} (однозначно определяемая набором узлов интерполяции), $S[\widetilde{f}]$ - некоторый функционал, оценивающий гладкость (в каком-либо смысле) интерполяционной функции $\widetilde{f}(X)$ на всем множестве S^{cur}. Предположим, что точка $X_a \notin X^T$ является решением задачи (1), т.е. что $f(X_a) = f_a$. В этом случае интерполяция $\widetilde{f}(X)$ зависит от X_a и однозначно задается уравнениями:

$$\widetilde{f}(X_i) = f(X_i) \text{ для } X_i \in X^T$$

$$\widetilde{f}(X_a) = f_a$$

Оценка гладкости $S[\widetilde{f}]$ может рассматриваться как функция от X_a. Стратегия достижения наибольшей гладкости заключается в поиске такой точки X_a, для которой результирующая интерполяция была бы наиболее "гладкой": $S[\widetilde{f}] \to \min$.

При реализации данной стратегии существенное значение имеют способ оценки гладкости интерполяции $\widetilde{f}(X)$, и способ выбора значения f_a. В общем случае, хорошей оценкой гладкости является интеграл квадрата второй производной $\widetilde{f}(X)$: $S[\widetilde{f}] = \int_{S^{cur}} \left[\widetilde{f}''(X)\right]^2 dX$, однако на практике вычисление такого интеграла часто является весьма трудной задачей. Тем не менее, для интерполяции с помощью функций радиального базиса в работе [Gutmann H-M. A radial basis function method for global optimization. // Technical report DAMTP 1999/NA22, Department of Applied Mathematics and Theoretical Physics, University of Combridge, England, (1999).] разработаны эффективные методы оценки значения $S[\widetilde{f}]$. Значения f_a могут выбираться в зависимости от номера итерации циклически из множества $\{f_{\min} - k_1, ..., f_{\min} - k_n\}$, где $k_1, ..., k_n$ - возрастающая последовательность чисел.

Большие значения k_i соответствуют "глобальному" поиску в области S^{cur}, малые значения k_i с большей вероятностью направляют поиск в окрестность текущего наилучшего найденного решения.

Алгоритм решения задачи оптимизации на модели. Для решения задачи (4) может использоваться любой алгоритм оптимизации, обеспечивающий отыскание глобального минимума с достаточной вероятностью. Поскольку затраты на вычисление функции выбора точки испытания $I(X,M,T)$ обычно малы по сравнению с затратами на вычисление "настоящей" целевой функции $f(X)$, вполне допускается выполнение большого числа итераций в процессе решения задачи (4) - приоритетом здесь является нахождение глобального минимума с как можно большей точностью.

В качестве примера алгоритмов, пригодных для решения задачи (6.1.), можно привести алгоритм дифференциальной эволюции и генетический алгоритм, использующий кодирование решения числами с плавающей точкой. Стоит отметить, что для ряда способов построения модели M и функции $I(X,M,T)$ возможно получение аналитических выражений для градиента $\nabla I(X,M,T)$, что дает возможность эффективно использовать алгоритмы локальной оптимизации для ускорения решения задачи и быстрой доводки текущих решений до ближайшего локального оптимума. Таким образом, для решения задачи (6.1.) весьма эффективным представляется подход, основанный на совместном применении методов локальной и глобальной оптимизации (т.н. гибридные алгоритмы).

Алгоритм уточнения области поиска. В процессе работы алгоритма увеличивается количество доступной информации о целевой функции $f(X)$, что влечет за собой повышение достоверности и точности аппроксимирующей модели *M*. Поэтому, начиная с некоторого момента, на основании анализа модели M становится возможно с определенной степенью достоверности идентифицировать "перспективные" и "неперспективные" области (в смысле вероятности отыскания глобального минимума функции $f(X)$). Периодическое уточнение области поиска $S^{cur} \subseteq S$ преследует следующие цели:

- сконцентрировать поисковые усилия в наиболее "перспективных" областях множества *S* допустимых значений параметров;
- избежать ненужных затрат на построение и исследование аппроксимирующей модели M в областях, где отыскание глобального минимума функции $f(X)$ маловероятно.

Один из методов состоит в ограничении текущей области поиска некоторой окрестностью точки $X_{\min}$, которая представляет наилучшее найденное на настоящий момент решение. Текущая область поиска S^{cur} определяется как пересечение исходной области поиска *S* и гиперкуба со сторонами $r_i, i = 1,2,..,N$ и центром в точке $X_{\min}$:

$$S^{cur} = \{S \mid \left| S_{\min}^{(i)} - S^{(i)} \right| < r_i\} \cap S,$$

Размеры гиперкуба, заданные значениями r_i, постепенно уменьшаются с работой алгоритма. Степень уменьшения может зависеть от результатов периодически проводящейся оценки корректности модели *M*.

Альтернативный метод состоит в ограничении текущей области поиска множеством *L(C)* точек, где оценка $\widetilde{f}(X)$ значения целевой функции не превосходит наилучшего найденного значения $f_{\min}$ на некоторую константу *C*: $L(C) = \{X \mid \widetilde{f}(X) \le f_{\min} + C\} \cap S$. В процессе работы алгоритма константа *C* уменьшается, что приводит к постепенному сужению области поиска. При практической реализации данного метода возникают затруднения, связанные с тем, что множество *L(C)* может иметь сложную форму, быть несвязным. Поэтому сначала приближенно оценивается минимальный гиперкуб *B(C)*, целиком содержащий *L(C)*, затем текущая область поиска устанавливается как $S^{cur} = B(C) \cap S$.

Критерий завершения. Алгоритм завершается при выполнении одного или нескольких из следующих условий:

- выполнение заданного числа вычислений "настоящей" целевой функции $f(X)$;
- неулучшение решения за заданное число итераций;
- оценка значений производных в точке $X_{\min}$, которая представляет наилучшее найденное на настоящий момент решение, позволяет говорить о достижении локального минимума: $\left\| \nabla \widetilde{f}(X_{\min}) \right\| < \varepsilon$
- оценка вероятности улучшения решения на величину ε на следующей итерации меньше некоторого порога p_{crit}: $\mathrm{Prob}(f(X^*) < f_{\min} - \varepsilon) < p_{crit}$;
- оценка величины $\mathrm{Imp}(X)$ ожидаемого на следующей итерации улучшения решения меньше, чем некоторый порог ε_{crit}: $\mathrm{Imp}(X^*) < \varepsilon_{crit}$.

Приложение 1. Операторы мутации и скрещивания

Оператор мутации

Обозначим символом $x = (x_1 x_2 ... x_n)$ "родительское" решение, представленное как вектор числовых значений, в свою очередь обозначенных символами x_i. Для элементов x_i заданы ограничения $lb_i \le x_i \le ub_i$. Каждое из значений x_i имеет один из следующих типов: вещественное число, целое число, номер элемента перечислимого множества. Оператор мутации $\hat{x} = m(x, P_m, S_m)$ порождает новое решение $\hat{x} = (\hat{x}_1 \hat{x}_2 ... \hat{x}_n)$ по следующей схеме:

- для каждого $i = 1,2,..,n$ выбирается случайное число $r_i \in (0,1)$.
- новое решение определяется по следующей формуле : $\hat{x}_i = \begin{cases} ms(x_i, S_m), & \text{if } r_i \le P_m \\ x_i, & otherwise \end{cases}$

Здесь $ms(x_i, S_m)$ - функция мутации скалярного числового значения, которая мутирует отдельные элементы вектора решения *x*. Различные способы определения $ms(x_i, S_m)$ описаны ниже по тексту.

Оператор мутации $\hat{x} = m(x, P_m, S_m)$ зависит от двух основных параметров:

- $P_m \in (0,1)$ - вероятность мутации (чем больше это число, тем выше вероятность того, что любой наперед взятый элемент родительского решения будет изменен в результате применения оператора мутации);
- $S_m > 0$ - степень ("сила") мутации (чем больше это число, тем выше вероятность того, что изменения в элементах родительского решения будут скорее большими, нежели малыми; значение $S_m = 1$ соответствует "стандартному" малому изменению)

Функция мутации скалярного числового значения $ms(x_i, S_m)$ определяется, в общем случае, по разному в зависимости от типа значения x_i (вещественное число, целое число, элемент перечислимого множества). Более того, для каждого типа значений существует несколько альтернативных способов определения $ms(x_i, S_m)$. Некоторые из возможных способов определения $ms(x_i, S_m)$ описаны ниже.

Для *вещественных* x_i, функция $ms(x_i, S_m)$ может быть определена двумя способами:

- "*Простая" функция мутации числового значения*. На первом шаге вычисляется t – начальное приближение (предварительная оценка) для значения функции $ms(x_i, S_m)$:

$$t = x_i - \frac{ub_i - lb_i}{2} + \frac{ub_i - lb_i}{\sigma} \sum_{i=1}^{\max(1,[\sigma])} r_i \text{, где } \sigma = \begin{cases} \frac{ub_i - lb_i}{S_m}, & if\ S_m > 10^{-4} \\ ub_i - lb_i, & otherwise \end{cases}$$

Здесь $r_i \in (0,1)$ - равномерно распределенные и независимые случайные величины, а символом $[x]$ обозначается целочисленное значение, ближайшее к x.

На втором шаге начальное приближение t корректируется исходя из необходимости попасть в интервал $lb_i \le x_i \le ub_i$:

$$ms(x_i, S_m) = \begin{cases} t + ub_i - lb_i & if\ t < lb_i \\ t & if\ lb_i \le t \le ub_i \\ t - (ub_i - lb_i) & if\ t > ub_i \end{cases}$$

Корректирующий шаг повторяется несколько раз, если это необходимо.

- *"Полиномиальная" функция мутации числового значения* основана на полиномиальном операторе мутации, впервые описанном Дебом в работе [Kalyanmoy Deb and Mayank Goyal. A Combined Genetic Adaptive Search GeneAS for Engineering Design. Computer Science and Informatics, 26(4):30--45, 1996.]. В настоящей работе используется модифицированная версия оригинального оператора:

  1. Выбирается случайное значение $u \in (0,1)$.
  2. Вычисляется параметр $\eta_m = \min(S_m, 1 - 10^{-2})$, называемый индексом распределения мутации (mutation distribution index). Чем больше значение этого индекса, тем менее сконцентрированным является распределение вероятных значений $ms(x_i)$ вокруг исходного значения x_i.
  3. Вычисляется параметр δ, используя функцию распределения плотности вероятности $P(\delta) = 0.5(\eta_m + 1)(1 - |\delta|)^{\eta_m}$:

$$\delta = \begin{cases} (2u)^{\frac{1}{1-\eta_m}} - 1, & if\ u < 0.5 \\ 1 - (2(1-u))^{\frac{1}{1-\eta_m}}, & if\ u \ge 0.5 \end{cases}$$

  4. Вычисляется начальное приближение для значения $ms(x_i)$:

$$t = x_i + \delta \cdot (ub_i - lb_i)$$

  5. Начальное приближение корректируется исходя из необходимости попасть в интервал $lb_i \le x_i \le ub_i$:

$$ms(x_i, S_m) = \begin{cases} t + ub_i - lb_i & if\ t < lb_i \\ t & if\ lb_i \le t \le ub_i \\ t - (ub_i - lb_i) & if\ t > ub_i \end{cases}$$

Корректирующий шаг повторяется несколько раз, если это необходимо.

Для *целочисленных* x_i, функция $ms(x_i, S_m)$ может быть определена двумя способами:

- *"Простая" функция мутации числового значения* почти полностью аналогична соответствующей функции, определенной для вещественных переменных (см. выше по тексту). Отличие здесь заключается в том, что функция для вещественных переменных не обязательно всегда возвращает целочисленное значение. Поэтому вводятся дополнительные шаги пост-обработки, призванные обеспечить целочисленность результата. Эти шаги приведены ниже (результат функции мутации, определенной для вещественных переменных, обозначен ниже символом ms_r):

  1. $t = \begin{cases} \lfloor ms_r \rfloor + 1, & if\ \lfloor ms_r \rfloor = x_i \\ \lfloor ms_r \rfloor, & otherwise \end{cases}$
  2. $ms(x_i, S_m) = \begin{cases} lb_i, & if\ t > ub_i \\ t, & otherwise \end{cases}$

- *"Бинарная" функция мутации числового значения.* В этой функции x_i представляется как битовая строка, к которой затем применяется “классический” оператор побитовой мутации. Процесс вычисления бинарной функции мутации включает следующие шаги:

  1. Определяется минимальное число битов b необходимых для представления x_i как битовой строки: $b = \lceil \log_2(ub_i - lb_i) \rceil$, где символами lb_i, ub_i обозначены соответственно нижняя и верхняя границы x_i и символом $\lceil x \rceil$ обозначена минимальное целое число, большее или равное x.
  2. Значение x_i представляется как битовая строка: $x_i = lb_i + \sum_{k=1}^{b} \alpha_k 2^k$
  3. Вычисляется максимальное количество мутирующих битов: $ni = \lfloor (b-1) \cdot (S_m + 0.5) \rfloor$

4. Выбирается случайный номер l, $1 \le l \le b$, и в битовой строке инвертируется бит с номером l: $\alpha_l = 1 - \alpha_l$. Данный шаг повторяется ni раз.
6. Вычисляется предварительный результат t функции мутации, используя мутированную битовую строку: $t = lb_i + \sum_{k=1}^{b} \alpha_k 2^k$
7. Результат корректируется исходя из необходимости попасть в интервал $lb_i \le x_i \le ub_i$:

$$ms(x_i, S_m) = \begin{cases} t + ub_i - lb_i & if\ t < lb_i \\ t & if\ lb_i \le t \le ub_i \\ t - (ub_i - lb_i) & if\ t > ub_i \end{cases}$$

Для x_i, представляющих собой номер элемента некоторого *перечислимого множества,* функция $ms(x_i, S_m)$ возвращает случайное целое число, равномерно распределенное в интервале $[lb_i, ub_i]$.

Оператор скрещивания

Предположим, что $x = (x_1 x_2 ... x_n)$ и $y = (y_1 y_2 ... y_n)$, где x_i, y_i - некоторые числовые значения (имеющие один из следующих типов: вещественное число, целое число, номер элемента перечислимого множества), являются двумя "родительскими" решениями, выбранными для скрещивания. Оператор скрещивания порождает два новых решения, $\hat{x} = (\hat{x}_1 \hat{x}_2 ... \hat{x}_n)$ и $\hat{y} = (\hat{y}_1 \hat{y}_2 ... \hat{y}_n)$, используя один из следующих методов:

- *одноточечное скрещивание:*

  Выбирается случайный номер k, $1 \le k \le n$. Затем элементы нового решения определяются по следующим формулам:

$$\hat{x}_i = \begin{cases} x_i, & if\ \ i \le k \\ y_i, & if\ \ i > k \end{cases}$$

$$\hat{y}_i = \begin{cases} y_i, & if\ \ i \le k \\ x_i, & if\ \ i > k \end{cases}$$

- *смешивающее скрещивание:*

  Независимо для каждого номера i, $1 \le i \le n$, выбирается случайное число r, $0 \le r < 1$, затем определяются (по нижеприведенным формулам) элементы нового решения:

$$\hat{x}_i = \begin{cases} lmix(x_i, y_i), & \text{if } r < P_{cr} \\ x_i, & otherwise \end{cases}$$

$$\hat{y}_i = \begin{cases} rmix(x_i, y_i), & \text{if } r < P_{cr} \\ y_i, & otherwise \end{cases}$$

- *одноточечное скрещивание со смешиванием центральной точки:*

  Выбирается случайный номер k, $1 \le k \le n$. Элементы нового решения вычисляются по следующим формулам:

$$\hat{x}_i = \begin{cases} x_i, & if \ i < k \\ lmix(x_i, y_i), & if \ i = k \\ y_i, & if \ i > k \end{cases}$$

$$\hat{y}_i = \begin{cases} x_i, & if \ i < k \\ rmix(x_i, y_i), & if \ i = k \\ y_i, & if \ i > k \end{cases}$$

Символами $lmix(x_i, y_i)$ и $rmix(x_i, y_i)$ выше обозначены два значения, вычисляемые так называемым *смешивающим оператором* $[lmix, rmix] = mix(x_i, y_i)$, а символом P_{cr} обозначена вероятность скрещивания (это настраиваемый параметр). Смешивающий оператор $mix(x_i, y_i)$ выполняет "скрещивание" двух числовых значений (не векторов, а чисел). Смешивающий оператор определяется по разному для целочисленных и вещественных значений.

Для *вещественных* значений, смешивающий оператор может быть определен одним из следующих способов:

- *смешивание с использованием оператора SBX (имитация бинарного скрещивания, simulated-binary crossover).* Оператор SBX был определен Дебом в работе [Deb, K. and Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. Complex Systems, 9 115--148.]. В настоящей работе для определения оператора смешивания используется модифицированная версия оператора SBX.

  1. Если x_i больше чем y_i, то обменять значения x_i и y_i.
  2. $\beta^x_{\max} = 1 - \dfrac{2\min(x_i - lb_i, ub_i - y_i)}{y_i - x_i}$, $\beta^y_{\max} = \dfrac{0.5 \cdot (y_i + x_i) - ub_i}{y_i - x_i}$
  3. $m^x = 1 - \dfrac{[\beta^x_{\max}]^{-n-1}}{2}$, $m^y = \dfrac{[\beta^y_{\max}]^{-n-1}}{2}$

4. Выбрать случайные числа $r^x \in (0, m^x)$, $r^y \in (0, m^y)$

5. $$\beta^x = \begin{cases} [2r^x m^x]^{\frac{1}{n+1}}, & \text{if } r^x < 0.5 \\ [2\cdot(1-r^x m^x)]^{-\frac{1}{n+1}}, & otherwise \end{cases},$$

$$\beta^y = \begin{cases} [2r^y m^y]^{\frac{1}{n+1}}, & \text{if } r^y < 0.5 \\ [2\cdot(1-r^y m^y)]^{-\frac{1}{n+1}}, & otherwise \end{cases}$$

6. $lmix = \frac{x_i + y_i}{2} + \beta^x \frac{x_i - y_i}{2}$, $rmix = \frac{x_i + y_i}{2} - \beta^x \frac{x_i - y_i}{2}$

7. Если установлен признак f_{chg} (это настраиваемый параметр), то выполняется обмен значений *lmix* и *rmix*.

Для *целочисленных* значений, смешивающий оператор может быть определен одним из следующих способов:

- *Смешивание с использованием побитового кодирования.* Оба значения x_i и y_i представляются как битовые строки, к которым затем применяется "классический" оператор одноточечного скрещивания. Процесс вычисления функции смешивания с использованием побитового кодирования включает следующие шаги:
  1. Вычисляется минимальное число битов b необходимое для представления x_i и y_i как битовых строк: $b = \lceil \log_2(ub_i - lb_i) \rceil$, где символами lb_i, ub_i обозначены соответственно нижняя и верхняя границы x_i и символом $\lceil x \rceil$ обозначена минимальное целое число, большее или равное *x*.
  2. Значения x_i и y_i представляются как битовые строки: $x_i = lb_i + \sum_{k=1}^{b} \alpha_k 2^k$, $y_i = lb_i + \sum_{k=1}^{b} \beta_k 2^k$.
  3. Выбирается случайный номер l, $1 \le l \le b$.
  4. Полагается $l_k = \begin{cases} \alpha_k, & if\ k \le l \\ \beta_k, & otherwise \end{cases}$, $r_k = \begin{cases} \beta_k, & if\ k \le l \\ \alpha_k, & otherwise \end{cases}$
  5. Полагается $lmix = lb_i + \sum_{k=1}^{b} l_k 2^k$, $rmix = lb_i + \sum_{k=1}^{b} r_k 2^k$

6. Если установлен признак f_{chg} (это настраиваемый параметр), то выполняется обмен значений *lmix* и *rmix*.

- *Смешивание с использованием SBX (имитация бинарного скрещивания, simulated-binary crossover).* Смешивание с использованием SBX для целочисленных переменных и переменных, представляющих номер элемента перечислимого множества, определяется на основе аналогичной функции для вещественных переменных (см. описание выше по тексту). Фактически, здесь x_i и y_i обрабатываются как обычные вещественные числа, а их нижняя и верхняя границы релаксируются следующим образлм: $lb_i' = lb_i - 0.5$, $ub_i' = ub_i + 0.5$. После выполнения смешивания с использованием SBX по алгоритму для вещественных переменных, получившиеся значения, *lmix* и *rmix* округляются до ближайшего целого числа (легко показать, что округленные таким образом значения всегда будут лежать в исходном интервале, ограниченном значениями lb_i, ub_i).