**АЛГОРИТМЫ ОПТИМИЗАЦИИ, ОСНОВАННЫЕ НА МЕТОДЕ ПРОБ И ОШИБОК**

**ЛЕКЦИЯ 1**

Математическая формулировка задач условной оптимизации и их классификация.

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

Классические задачи комбинаторной оптимизации: задача коммивояжера и задача о рюкзаке.

**ЛЕКЦИЯ 2**

Задачи построения расписаний и их классификация.

Задачи, возникающие при проектировании вычислительных систем реального времени:

* Задача определения минимально необходимого числа процессоров и построения расписания выполнения функциональных задач со временем выполнения не превышающим заданный директивный срок.
* Задача построения статико-динамических расписаний.
* Задача построения расписания обменов по шине с централизованным управлением (на примере стандарта MIL-STD 1533В).

**ЛЕКЦИЯ 3**

***Алгоритмы случайного поиска***

Ненаправленный случайный поиск (метод Монте-Карло)

Алгоритмы направленного случайного поиска без самообучения

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. Различные способы коррекции вектора направления наилучших шагов

**ЛЕКЦИЯ 4**

***Алгоритмы имитации отжига***

Концепция построения алгоритмов

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

Модификация закона понижения температуры, позволяющая ускорить выход из локальных оптимумов

Методы распараллеливания алгоритмов имитации отжига:

* асинхронный параллельный алгоритм
* параллельный алгоритм с синхронизацией
* алгоритм, основанный на декомпозиции целевой функции
* подходы, основанные на декомпозиции пространства решений

**ЛЕКЦИЯ 5**

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

* математическая формулировка задачи
* способы представления расписания и операций его преобразования
* стратегии применения операций преобразования текущего решения
* стандартные операции алгоритма

**ЛЕКЦИЯ 6**

Параллельный алгоритм имитации отжига для построения статических многопроцессорных расписаний:

* метрика в пространстве расписаний
* разбиение исходного пространства решений на области
* операции преобразования расписания внутри области
* распределение областей по узлам вычислительной системы
* сравнение эффективности алгоритмов

**ЛЕКЦИЯ 7**

***Генетические и эволюционные алгоритмы***

Простой генетический алгоритм (алгоритм Холланда)

Теория схем. Гипотеза строительных блоков

Модификации генетических алгоритмов

Алгоритмы дифференциальной эволюции

**ЛЕКЦИЯ 8**

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

математическая формулировка задачи

кодирование решений

операции генетического алгоритма

функция выживаемости и критерий останова

**ЛЕКЦИЯ 9**

***Муравьиные алгоритмы***

Концепция построения алгоритмов (биологическая модель)

Общая схема работы муравьиных алгоритмов

Модификации муравьиных алгоритмов:

* Максиминный алгоритм
* Модификация с поглощением феромона
* Совместное использование с алгоритмами локального поиска

**ЛЕКЦИЯ 10**

Муравьиные алгоритмы для решения задачи построения статико-динамических расписания:

* Первый способ сведения задачи построения статико-динамического расписания к задаче нахождения на графе маршрута
* Второй способ сведения задачи построения статико-динамического расписания к задаче нахождения на графе маршрута

**ЛЕКЦИЯ 11**

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

**Лекция 1**

**Билет 1**

Задача условной оптимизации заключается в нахождении компонентов вектора *X=(x1,…,xn)* минимизирующих целевую функцию *f(X)* при выполнении ограничений *gi(X)* и *X ∈ S*:

*min f(X)*

*gi(X) ≤ 0, i=1,…,m* (1.1.)

*X ∈ S*

Если , то как минимум одна из функций *f, gi* в этом случае будет не определена на этом значении *X*.

По свойствам функций *f, gi* и определению множества *S* можно ввести классификацию различных задач оптимизации

**[М. Мину. Математическое программирование. Теория и алгоритмы.- М.: Наука, 1990.]**.

Например, если функции *f,gi* – линейные и *S ⊂ Zn* , то задача (1.1) относится к классу задач целочисленного линейного программирования.

В отдельный класс выделим задачи комбинаторной оптимизации.

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.

1

2

3

4

5

4(1,3)

3(1,2)

5(1,3)

2(2,1)

4(2,2)

1(1,2)

5(1,4)

Расписание: {“привязка к процессору”, “порядковый номер”}

{“привязка к процессору”,“приоритет ”}

**↓**

{“привязка к процессору”,“порядковый номер”}

Сложные задачи условной оптимизации – задачи для которых отсутствует априорная информация о функциях *f,gi* и множестве *S*, которая может быть использована для организации поиска оптимального решения, или сложность получения этой информации неприемлема:

* Функций *f,gi* могут быть операторами, заданными правилами/алгоритмами их вычисления.
* Множество *S* может быть компонентно разнородным: *S = SR ∪ SZ ∪ SF ∪ SKS*.

Идея о целесообразности случайного поведения, при наличии неопределенности, сформулирована У.Р. Эшби в работе

**[У. Росс Эшби. Конструкция мозга.- М.:, ИЛ, 1962.]**

Алгоритмы, опирающиеся на метод проб и ошибок:

* **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.].

**Задача коммивояжера**

Задачу коммивояжёра можно представить в виде нахождения маршрута на графе.

* Вершины графа – города.
* Ребра - пути сообщения между этими городами.
* Каждому ребру сопоставлен вес.

Задача заключается в отыскании кратчайшего маршрута, в который входит по одному разу каждая вершина графа.

Длина маршрута - сумма весов входящих в него ребер.

Маршрут может быть описан циклической перестановкой номеров городов:

,

где  - номер города находящийся в *i* –ой позиции перестановки.

Пространством поиска решений является множество перестановок *n* городов, таких, что все  разные и .

**Cимметричная задача коммивояжера:**

* неориентированный граф,
* для любого ребра *(i,j)* .

**Асимметричная задача коммивояжера:**

* ориентированный граф,
* дуги между одними вершинами могут иметь разный вес, т.е. .

**Симметричная задача коммивояжера является метрической**, если для весов ребер выполняется правило треугольника: . 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 не попадает в оптимальный маршрут и маршруты, близкие по длине к оптимальному маршруту.

Данная задача принадлежит к классу *NP*-трудных задач.

**Задача о рюкзаке**

Имеется:

* рюкзак объемом *b,*
* набор *n* различных товаров: объем *ai* и стоимость *ci*.

Требуется:

* упаковать рюкзак так, чтобы суммарная стоимость упакованных товаров была максимальной и их суммарный объем не превышал объем рюкзака *b*.



Если товар упакован в рюкзак, то *xi=1*, если нет *xi=0*.

Если все коэффициенты *ai* целые числа и *b* целое число, то задача является *NP*-трудной.

Если все коэффициенты *ai* вещественные, то задача может быть решена жадным алгоритмом сложности *O(n∙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-ai* 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 .

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.

Разработать генетический алгоритм и обосновать его свойства для решения:

* задачи о рюкзаке,
* симметричной задачи коммивояжера,
* асимметричной задачи коммивояжера,
* симметричной метрической задачи коммивояжера.

Разработать алгоритм имитации отжига и обосновать его свойства для решения:

* задачи о рюкзаке,
* симметричной задачи коммивояжера,
* асимметричной задачи коммивояжера,
* симметричной метрической задачи коммивояжера.

Разработать муравьиный алгоритм и обосновать его свойства для решения:

* задачи о рюкзаке,
* симметричной задачи коммивояжера,
* асимметричной задачи коммивояжера,
* симметричной метрической задачи коммивояжера.

Разработать алгоритм направленного случайного поиска для решения:

* задачи о рюкзаке,
* симметричной задачи коммивояжера,
* асимметричной задачи коммивояжера,
* симметричной метрической задачи коммивояжера.

**Лекция 2**

**Задачи построения расписаний и их классификация**

[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:

* моделью ресурсов
* моделью системы заданий (работ)
* моделью вычислений
* мерой оценки расписаний
* ограничениями на корректность расписания

**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 *Ri* задается целым положительным числом.

В ряде задач рассматривается среда обмена.

**Система работ определена через *T, , [τji], {Rj}*, где**:

* *T={T1*,*T2,…,Tn}* - набор работ подлежащих планированию;
* ** - заданное на *T* отношение частичного порядка. Если работы связаны отношением *TiTj* ,то работа *Ti* должна быть завершена раньше, чем начнется выполнение работы *Tj*.
* *[τji]* – матрица времен выполнения работ на процессорах, элемент которой *τji* задает время выполнения работы *j* на процессоре *i*;
* *R={R1(Tj)*,*…, Rs(Tj)}* – набор, *i-ая* компонента которого задает количество ресурса типа *Ri* необходимое для выполнения задания *Tj*.

**Расписание**может задаваться одним из трех способов:

* Временная диаграмма – для каждой работы задано время начала выполнения *s’(Tj)* и процессор на котором она выполняется.
* Привязка работ к процессорам и порядковый номер выполнения задания на процессоре.
* Привязка работ к процессорам и приоритет работы. 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:**

* 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:**

* пустое,
* лес,
* произвольное.

**Времена выполнения работ:**

* различными,
* равными,
* равными 1 или 2.

**Директивные интервалы работ:**

* *f*- общий директивный интервал для всей системы работ,
* *[sj,fj)* – индивидуальные директивные интервалы для каждой работы,
* *[0,fj)* – индивидуальные директивные интервалы с общей левой границей,
* *rj* – требуемая частота выполнения работы.

**Модель вычислений**:

* дисциплина обслуживания работ,
* учитываемые временные задержки.

**Дисциплина обслуживания работ**:

* 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).

Функции штрафа определены для работ *i= <s'i, si, fi, ti>* и имеют вид *Fi = Fi(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;
  + c1, c2 - константы, не зависящие от конкретной работы.

| **Название штрафной функции** | **Математическое представление:**  **Fi =** |
| --- | --- |
| *Отставание [1,2,3,11,13,14,15]* | *max{0, f'i – fi}* |
| Смещение [1,2,4,5,6,11,13,14] | *f'i – fi* |
| *Завершение* [*2, 4,5,6,7,14]* | *f'i* |
| Дискретное запаздывание [1,4,15] | *1, если f'i > fi, иначе 0* |
| Отклонение [5] | *|fi' – fi|* |
| Опережение [5,13] | *max{0, fi – fi'}* |
| Непопадание [7] | *0, если (si', fi')  (si, fi), иначе 1* |
| Простых элементов [8,9] | *U\* (U+ 1) / 2, где*  *U = max(fi' – fi,0) + max(si'– si,0)* |
| Красильщика [11] | *fi' + consti* |
| Длительность [13] | *fi' – si'* |
| «Жесткие сроки» [12] | *-c1, если fi' ≤ fi, иначе c2(fi'/fi – 1)* |
| Крепкие сроки [12] | *-c1, если fi' ≤ fi, иначе c2* |
| Мягкие сроки [12] | *-c1, если fi' ≤ fi, иначе (c2-c1)fi'/fi – c2* |
| Неубывающая кусочно-неперывная (общий вид) [15] | *0, если fi' ≤ fi, иначе E = φ, где φ > 0, φ – неубывающая кусочно-непрерывная функция* |
| Ступенчатая [15] | *0, если fi' ≤ fi, иначе ai, где ai > 0, ai – некоторая константа для i-й работы* |
| *Длительность прохождения (Flow time) [16, 17]* | *f'i – si* |
| Функция специального вида (1) [1] | *φ(fi') + αi, где φ(fi') не убывает в полуинтрвале (0, f], α1 ≥ α2 ≥ … ≥ αn* |
| Функция специального вида (2) [1] | *αi \* φ(fi'), где φ(fi') не убывает в полуинтрвале (0, f], α1 ≥ α2 ≥ … ≥ αn* |
| Функция специального вида (3) [1] | *φ(fi' + αi), где φ(fi') не убывает в полуинтрвале (0, f], α1 ≥ α2 ≥ … ≥ αn* |
| Функция специального вида (4) [1] | *aiui(fi'), ui(fi') = 0, если fi' ≤ f, иначе ui(fi') = 1; ai > 0* |
| Произвольная неубывающая [1] | *Произвольная неубывающая функция* |

**Критерии оценки качества расписаний** построенные на основе функций штрафа.

|  |  |
| --- | --- |
| **Название критерия** | **Математическое представление** |
| Максимум [1,4,5,6,11,13,14,15] | **max** *Fi*, i = 1..n |
| Взвешенный максимум [2,14] | **max** *w*i*Fi*, I = 1..n |
| Сумма [9,10,13,14,15,16] |  |
| Взвешенная сумма [8,10,11] |  |
| Средняя сумма [13] |  |
| Взвешенная средняя сумма [14] |  |

**Задача построения расписания обменов по шине с централизованным управлением** (на примере стандарта MIL-STD 1533В)

ОУ1

ОУ2

ОУ*N*

шина

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

…

КС

КС

ОС

СД

СД

СД

КС

Следующее соб.

*t*1

*t*2

ОС

*t*1

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.

Шина в данной системе может рассматриваться как одноприборное устройство, обслуживающее исходно заданный набор работ без прерываний.

Дано:

* Множество работ, которые должны выполняться на системе . Для каждой работы заданы - время выполнения, - директивный интервал выполнения и выполняется условие ;
* Дополнительные ограничения на корректность расписания ;
* Вектор значений технологических требований .

Расписание выполнения работ, которое представляет собой упорядоченное множество



Здесь *k* - порядковый номер *j*-ой работы в расписании,

 - время начала выполнения *j*-ой работы в расписании ,

 - время завершения выполнения *j*-ой работы.

Множество корректных расписаний  (т.е. множество *S* в задаче (1.1.)) определим набором ограничений:



Задача:



известна в теории расписаний как задача о выборе максимального числа совместимых заявок и является *NP*–трудной.

Для частной задачи:



известен оптимальный жадный алгоритм сложности *O(n∙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:



**Задача для схемы с подциклами**

s1

f1

s2

f2

s3

f3

φ1

φ1

φ1

...

φ2

φ2

φ2

**2\*T**

**3\*T**

0

**T**

число работ в цепочке

цепочка работ

резерв времени в конце подцикла

резерв для сдвига

расписания

резерв времени в начале подцикла

1. в каждом подцикле может находиться не более 1 цепочки работ и работы в цепочке следуют друг за другом без пауз;
2. время выполнения работ не должно пересекать границы подцикла;
3. время начала цепочки работ относительно начала соответствующего подцикла не должно быть меньше заданного значения;
4. в конце подцикла должен быть зарезервирован интервал времени, длительность которого не меньше, чем заданная доля длительности подцикла;
5. число работ в цепочке не должно превышать заданного значения;
6. 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.

**Задача для схемы без подциклов**

1. число работ в цепочке не должно превышать заданного значения;
2. суммарная длительность выполнения работ цепочки не должна превышать заданного значения;
3. интервал времени между последовательными цепочками должен быть не меньше заданного значения;
4. 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.

**Задача построения статико-динамических расписаний**

Пусть задано множество работ:

*SW=*{*ai=<si,fi,ti,pi>|i*[1*..n*]}, где

* [*si,fi*) – директивный интервал;
* *ti* – время выполнения работы;
* *pi* – номер раздела работы;
*  и , определяющие, временные затраты на переключение контекста между окнами и резерв свободного времени внутри каждого окна.

Требуется построить расписание, представляющее собой набор окон:

*SP=*{*wi*=*<Si,Fi,SWi>|i*[1*..m*]}, где

* *Si* – время открытия окна;
* *Fi* – время закрытия окна;
* *SWi=*{*aij*}*SW* – множество работ, выполняемых внутри окна.

**Ограничения корректности расписания:**

1. *(i,j)*[1*..m*],*i**j*:*SWi**SWj*= – множества работ, размещенных внутри разных окон, не пересекаются;
2. *i*[1*..n*],*j*[1*..m*],*aiSWj*:*si**Sj*<*Fj**fi* – временной интервал окна лежит внутри директивных интервалов работ, выполняющихся в окне;
3. *(i,j)*[1*..n*],*k*[1*..m*],*ai,ajSWk*:*pi=pj* – разделы работ, размещенных внутри одного окна, совпадают;
4. *(i,j)*[1*..m*],*i<j*:*Sj**Fi+* – окна не пересекаются и между любыми двумя соседними окнами есть промежуток не короче времени, необходимого на переключение контекста;
5.  – суммарное время выполнения всех работ из одного окна с учетом резерва времени не больше, чем длина окна.

Математическая формулировка задачи построения расписания обменов по каналу с централизованным управлением для схемы с подциклами.

Математическая формулировка задачи построения расписания обменов по каналу с централизованным управлением для схемы без подциклов.

**Лекция 3**

См презентацию

**Лекция 4**

**Л.А. Растригин. Статистические методы поиска.- М.: Наука, 1968.**

Основой методов случайного поиска служит итерационный процесс:



*αk* – величина шага,

*ξ = (ξ1,…,ξn)* – некоторая реализация *n*-мерного случайного вектора *ξ*.

*min f(X)*

*gi(X) ≤ 0, i=1,…,m* (1.1.)

*X ∈ S*

Ненаправленный случайный поиск (метод Монте-Карло)



Алгоритмы направленного случайного поиска без самообучения определяются двумя элементами:

1. Алгоритмом выбора пробных состояний на текущем шаге.
2. Решающим правилом, по которому на каждом шаге выбирается новое текущее приближение решения.

Алгоритмы направленного случайного поиска:

* алгоритм с парной пробой,
* алгоритм с возвратом при неудачном шаге,
* алгоритм с линейной экстраполяцией,
* алгоритм наилучшей пробы,
* алгоритм статистического градиента.

X\*

=C1



f(X)=C2

f(X)=C3

C1<C2<C3

Построить алгоритм с возвратом при неудачном шаге.

*min f(X)*

*gi(X) ≤ 0, i=1,…,m* (1.1.)

*X ∈ S*

**Шаг 1.** Выбрать:

* параметр точности ε > 0,
* начальный шаг α > 0,
* коэффициент уменьшения шага γ > 1,
* предельное число неудачных попыток *K*,
* начальное приближение .

Вычислить:

целевую функцию .

**Шаг 2.** *построить алгоритм*

**Шаг 1.** Выбрать:

* параметр точности ε > 0,
* начальный шаг α > 0,
* коэффициент уменьшения шага γ > 1,
* предельное число неудачных попыток *K*,
* начальное приближение .

Вычислить: целевую функцию .

**Шаг 2.** Установить счетчик неудачных попыток: *j=1.*

**Шаг 3.** Получить реализацию случайного вектора *ξ.*

**Шаг 4.** Найти пробную точку: .

**Шаг 5.** Проверить выполнение ограничений и *Y∈S.* Если ограничения выполняются, то перейти к шагу 6, иначе к шагу 3 (возможно также считать это неудачной попыткой – переход к шагу 7).

**Шаг 6.** Вычислить *f(Y)*. Если *f(Y) < f(X)*, то *X = Y, f(X) = f(Y)*, и перейти к шагу 3. Иначе, перейти к шагу 7.

**Шаг 7.** Увеличить счетчик попыток *j = j + 1*. Если *j ≤ K*, то перейти к шагу 3, иначе к шагу 8.

**Шаг 8.** Проверка условия достижения точности. Если *α < ε*, то поиск завершить, полагая , . Иначе – положить

α = α/γ и перейти к шагу 2.

Алгоритмы случайного направленного поиска с самообучением заключаются в перестройке вероятностных характеристик поиска, т.е. в определенном целенаправленном воздействии на случайный вектор *ξ*.

Это достигается введением вектора памяти



- вероятность выбора положительного направления по *j*-ой координате на *k*-ом шаге.

Алгоритмы направленного случайного поиска с самообучением определяются тремя элементами:

1. Алгоритмом выбора пробных состояний на текущем шаге.
2. Решающим правилом, по которому на каждом шаге выбирается новое текущее приближение решения.
3. Алгоритмом самообучения, которые корректирует вектор предыстории с точки зрения информации, полученной на текущем шаге.

**Самообучение методом исключения.**

Ограничение – число возможных направлений конечно.

Исключаются из рассмотрения «неудачные» направления → повышается вероятность отыскания «удачных» направлений.

Недостаток - большой объем памяти.

**Покоординатное экспоненциальное обучение.**

Элементы вектора *Х* изменяются на строго определенную по модулю величину. Она одинакова для всех элементов. Разница лишь в направлении изменения.

Число возможных направлений: 

**Алгоритм покоординатного самообучения с произвольным законом изменения вероятности.**

Пусть 

Вид функции  может быть различным, но она должна быть:

* монотонной,
* неубывающей.

Линейная зависимость:



Экспоненциальная зависимость:





- шаг, определяющий скорость обучения.

X1

Избежать передетерминирования можно наложив ограничения на область изменения *w*:





Чувствительность алгоритма к выбору наилучшего направления поиска можно улучшить, учитывая степень участия каждого элемента вектора *Х* в изменении целевой функции:



Для задач с существенно нерегулярным рельефом целевой функции актуально достаточно быстрое забывание сведений, полученных ранее:



*k* – параметр забывания 

При отсутствии опыта  и  алгоритм вырождается в равновероятностный поиск:





|  |  |  |
| --- | --- | --- |
|  |  |  |
| **+** | **+** | **↓** |
| **+** | **-** | **↑** |
| **-** | **+** | **↑** |
| **-** | **-** | **↓** |

Рассмотренные алгоритмы (за исключением алгоритма с забыванием) однажды обнаружив хорошее направление, будут стараться фиксировать движение в этом направлении. Причиной этого является в основном положительная обратная связь.

Алгоритм обучения строится так, чтобы вообще не учитывать положительный опыт работы. Любой результат шага поиска воспринимается как отрицательный, но в разной мере:



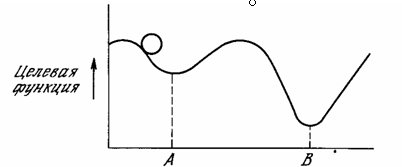
*d* выбирается так чтобы всегда обеспечивалось неравенство:



**Лекция 5**

**Алгоритмы имитации отжига**

*Концепция построения алгоритмов*



*Общая схема алгоритмов*

Шаг 1. Задать начальное корректное решение *X0∈S* и считать его текущим вариантом решения (*X=X0*).

Шаг 2. Установить начальную температуру *Т0*, приняв ее текущей (*Т=Т0*).

Шаг 3. Применить операции преобразования решения к текущему решению *X* и получить новый корректный вариант решения *X’∈ S*, если это решение является лучшим из ранее найденных решений, то запомнить его.

Шаг 4. Найти изменение функционала оценки качества решения :

* + если  (решение улучшилось), то новый вариант решения считать текущим (*X=X'*);
  + если (решение ухудшилось), то принять с вероятностью  в качестве текущего решения новый вариант решения *X'*.

Шаг 5. Повторить заданное число раз шаги 3 и 4 без изменения текущей температуры.

Шаг 6. Если критерий останова выполнен, то завершение работы алгоритма.

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

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 его изменять.
3. Выбрать закон понижения температуры на шаге 7.
4. Определить функционал , используемый для оценки качества текущего решения на шаге 4.
5. Выбрать критерий останова алгоритма, используемый на шаге 6.

Закон Больцмана: .

* Закон Коши: .
* .

*T* - текущая температура алгоритма,

*T0* - начальная температура,

*x* - номер итерации алгоритма.

,

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

***Методы распараллеливания алгоритмов имитации отжига***

*Схема параллельного асинхронного алгоритма.*

Initial\_solution1

. . .

. . .

New\_local

solution1

Best\_global

solution

Best\_local

solution1

Initial\_solutioni

New\_local

solutioni

Best\_local

solutioni

Initial\_solutionn

New\_local

solutionn

Best\_local

solutionn

*Схема параллельного алгоритма с синхронизацией*

**Stop**

**condition**

Initial\_solution

. . .

. . .

New\_local

solution1

Best\_local

solution1

Initial\_solution

New\_local

solutioni

Best\_local

solutioni

Initial\_solution

New\_local

solutionn

Best\_local

solutionn

Gather best solutions

Broadcast best solution

Best

solution

**yes**

**no**

*Способы обмена полученными решениями и стратегии принятия решения в качестве текущего:*

* Процесс *i* посылает процессу *i+1* своё текущее решение. Процесс *i+1* принимает это решение в качестве текущего, если *F(Xi) < F(Xi+1)*. Где *F(Xi)* – 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(Xi) < F(Xj)*  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.

*Алгоритм, основанный на декомпозиции целевой функции*

Большую часть времени алгоритм имитации отжига затрачивает на вычисление целевой функции.

Если целевая функция является декомпозируемой *F(x1, x2, … , xi, … , xn) = F(x1) + F(x2) + … + F(xi) + … +F(xn)*,

то возможно параллельное её вычисление на *m* (*m≤n*) процессорах.

*Подходы, основанные на декомпозиции пространства решений*

**Декомпозиция на области**

**О1**

**О2**

**Оi**

**Оn**

Поиск решения в О1, О2

Поиск решения в Оi, Оi+1

Поиск решения в Оn-1, Оn

Выбор наилучшего решения

Распределение областей по процессам

…

…

**Лекция 6**

Поведение программы *Bh(PR)*:

*Bh(PR) = < S, {R ⎯ (PR)}, {R → (PR)}>*,

*S* – множество всех возможных шагов процессов (рабочих интервалов) в допустимом диапазоне входных данных программы,

*{R → (PR)}* - отношения, определяющие частичный порядок на множестве шагов каждого процесса,

{*R ⎯ (PR)}* - отношения взаимодействия между процессами.

Рабочий интервал процесса - внутренние действия процесса между двумя последовательными взаимодействиями с другими процессами.

Для задачи построения расписания будем использовать одну из историй поведения программы *H(PR)∈Bh(PR)*.

Для *H(PR)* отношение *{R → (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 соответствуют рабочие интервалы процессов,

дугам *={ik=(pi,pk)}(i,k)О(1...N)* - связи, определяющие взаимодействия между рабочими интервалами из множества *P* (определяются объединением отношений *{R⎯(PR)}, {R→(PR)}*).

Модель прикладной программы определим:

1. Множеством рабочих интервалов процессов, составляющих программу *PR*:



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

1. Частичным порядком на *P*:

*={ik=(pi,pk)}(i,k)∈(1...N)*;

1. Вычислительной сложностью рабочих интервалов:

;

1. Объемом данных обмена для каждой связи из **:

*{vik}(i,k)∈(1...N)*;.

*Расписание выполнения программы* определено, если заданы:

* привязка,
* порядок.

Модель расписания выполнения программы определим набором простых цепей и отношением частичного порядка *HP* на множестве *P*:

*HP=({SPi}i=(1...M) ,HP)*.

*{SPi}i=(1...M)* -набор простых цепей (ветвей параллельной программы).

Отношение частичного порядка *HP* на множестве *P* для *HP* определим как объединение отношений: *HP*=*c*∪*1*∪…∪*M*, *i* - отношение полного порядка на *SPi*, которое определяется порядковыми номерами рабочих интервалов в *SPi*; *c* - набор секущих ребер, которые определяются связями рабочих интервалов, распределенных на разные процессоры.

Если рабочие интервалы *pi* и *pj* распределены на разные процессоры и в  существует связь *ij*, то она определяет секущее ребро в *HP*.

На отношение *HP* накладываются условия ацикличности и транзитивности.

Расписание *HP* является корректным, если выполнены следующие ограничения:

1. Каждый рабочий интервал должен быть назначен на процессор (в *SPi*).
2. Каждый рабочий интервал должен быть назначен лишь на один процессор (в один *SPi*).
3. Частичный порядок, заданный в *H* сохранен в *HP*: 
4. Расписание *HP* должно быть беступиковым. Условием беступиковости является отсутствие контуров в графе *HP*: .
5. Все рабочие интервалы одного процесса должны быть назначены на один и тот же процессор (в один и тот же *SPi*).

В дальнейшем будем говорить, что расписание корректно , если оно удовлетворяет ограничениям 1-5.

*Дано:*

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

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



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 его изменять.
3. Выбрать закон понижения температуры на шаге 7.
4. Определить функционал , используемый для оценки качества текущего решения на шаге 4.
5. Выбрать критерий останова алгоритма, используемый на шаге 6.

*Бинарное представление расписания.*

Расписание задается:

* матрицей привязки *Y(HP)N×M*
* матрицей смежности *X(HP)N×N* графа *HP*.

Элементы матриц определяются:

*M* – число процессоров в ВС,

*N* – число рабочих интервалов в *H*.

Недостатком данного представления является большое число бинарных переменных *N2+N⋅M*.

*Целочисленное представление расписания*

1. Расписание задается матрицей *Y(HP)N×M*, где элемент матрицы определяется: ,

*c* – порядковый номер рабочего интервала *pi* в *SPj*.

При данном способе представления расписания число целочисленных переменных равно *N⋅M*.

2. Расписание задается:

* вектором привязки *Y(HP)K* (*i*-й элемент вектора *Y(HP)K* равен номеру списка в который назначены рабочие интервалы *i*-го процесса),
* вектором порядка *X(HP)N*, (*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 *SPj*;
* расписания *HP* и *HP′* отличаются привязкой рабочих интервалов.

Операция изменения порядка рабочих интервалов в одном списке:

* изменяет порядковый номер рабочего интервала *pi* в списке *SPm* (порядковый номер рабочего интервала становится равным *c*),
* корректирует порядковые номера соответствующих рабочих интервалов в данном списке (*NSm* – число рабочих интервалов в списке *SPm*).



Операция изменения привязки рабочих интервалов

* переносит рабочий интервал *pi* из списка *SPm* в список *SPk* (порядковый номер рабочего интервала становится равным *c*),
* корректирует порядковые номера соответствующих рабочих интервалов в списках *SPm* и *SPk*:



*Теорема.* Если *HP* и *HP′* - произвольные корректные расписания (), то существует конечная цепочка команд , переводящая расписание *HP* в *HP′*, такая, что все *K* промежуточных расписаний являются корректными и *K ≤ 2N*.

*Условия применимости операций не нарушающие ограничений на HP*

Получим интервал значений параметра *c* при применении операции *O1/O2* к рабочему интервалу . Расписания *HP* будем представлять в ярусной форме максимальной высоты.

 - множество непосредственных предшественников рабочего интервала  (всегда выполняется *k<i* и *l<s*),

 - множество непосредственных последователей рабочего интервала  (всегда выполняется *k>i* и *l>s*).

Операция  - получает максимальный номер яруса, на котором расположен один из непосредственных предшественников рабочего интервала , для рабочих интервалов, не имеющих предшественников *lin=0* (нулевой ярус всегда пуст).

Операция  - получает минимальный номер яруса, на котором расположен один из непосредственных последователей рабочего интервала , для рабочих интервалов, не имеющих последователей *lout=N*

Тогда, рабочий интервал  может быть размещен в любой из списков *SPj* на любой из ярусов из интервала *lin<s<lout*. Если выбранный ярус занят, то осуществляется коррекция ярусной формы путем соответствующего сдвига ярусов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | PIN | *PI* | *POUT* | *C* |
| 1 | *PIN=∅* | *PI=∅* | *POUT=∅* | *c=1* |
| 2 | *PIN=∅* | *PI=∅* | *POUT≠∅* | *c=1* |
| 3 | *PIN=∅* | *PI≠∅* | *POUT=∅* |  |
| 4 | *PIN=∅* | *PI≠∅* | *POUT≠∅* |  |
| 5 | *PIN≠∅* | *PI=∅* | *POUT≠∅* |  |
| 6 | *PIN≠∅* | *PI=∅* | *POUT=∅* |  |
| 7 | *PIN≠∅* | *PI≠∅* | *POUT=∅* |  |
| 8 | *PIN≠∅* | *PI≠∅* | *POUT≠∅* |  |

*Возможность снятия условий: каждый процесс имеет не более одного рабочего интервала или на расписание не 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 1-5.

**Стратегии применения операций преобразования текущего решения**

*Стратегия уменьшения задержек.*

Если время начала выполнения каждого рабочего интервала равно длине критического пути в графе *H* от истоков до рабочего интервала, то расписание будет оптимальным.

*Стратегия заполнения простоев.*

Эта стратегия основана на эмпирической гипотезе: чем меньше времени в сумме простаивают процессоры, тем лучше расписание.

*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.

*Схемы реализации стратегий*

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

Закон понижения температуры - закон с искусственным выходом из локальных оптимумов.

В качестве функционала  используется функция *T=f(HP,HW).*

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

**Параллельный алгоритм имитации отжига для построения статических многопроцессорных расписаний**

Множество всех возможных решений  задачи построения расписаний представляется совокупностью областей :

1. ;
2. ;
3. введённые в  операции преобразования должны быть замкнутыми на областях  и сохранять на них свойство полноты.

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.

**Метрика в пространстве *HP\****

Пусть *HP1*, *HP2HP\**- два произвольных корректных расписания. Расстоянием *(HP1,HP2)* между расписаниями *HP1* и *HP2* будем называть число, равное длине минимальной допустимой цепочки операций *O1* и *O2*, которая переводит расписание *HP1* в расписание *HP2*:

*(HP1,HP2)= L(S).*

*L(S)*- длина цепочки *S* (количество операций в этой цепочке).

*(HP) -* множество всех допустимых цепочек

Функция *(HP1,HP2)* является метрикой:

1. функция определена всюду на *HP\**;
2. для  выполнены аксиомы метрики.

*Свойство 1*. *HP1, HP2HP\** : *(HP1,HP2) 0 и* *(HP1,HP2) = 0  HP1=HP2.*

*Свойство 2*. *HP1,HP2HP\** : *(HP1,HP2) =* *(HP2,HP1)*

*Свойство3*(неравенство треугольника)*.* *HP1,HP2,HP3HP\**: *(HP1,HP3)**(HP1,HP2)+**(HP2,HP3)*.

**Разбиение исходного пространства решений на области**

4

5

6

7

8

1

2

3

Рабочие интервалы выполняются на разных процессорах

Рабочие интервалы выполняются на одном процессоре. Второй после четвертого

Рабочие интервалы выполняются на одном процессоре. Четвертый после второго

4

5

6

7

8

1

2

3

4

5

6

7

8

1

2

3

Выбор двух рабочих интервалов

Каждой из областей , можно поставить в соответствие граф .

*P -* множество вершин, соответствующих рабочим интервалам.

Дугам  отвечают связи, определяющие взаимодействия между рабочими интервалами. Все связи  присутствующие в *H* сохраняются в . Отношение  задает дуги, устанавливающие дополнительные ограничения на порядок выполнения рабочих интервалов в каждой области. Отношение  транзитивно и ациклично.

Отношения *J* и *K* соответствуют ограничениям на привязку рабочих интервалов:

* два рабочих интервала  и  связаны отношением *J*, , если они должны выполняться на одном процессоре;
* два рабочих интервала  и  связаны отношением *K*, , когда они распределены на разные процессоры.

Дополнительно к ограничениям 1-5 на корректность расписания вводятся еще два ограничения:

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

Алгоритм позволяет так осуществлять разбиение на области, что критические пути в графах областей разбиения будут существенно отличаться между собой.

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.

**Операции преобразования расписания**

 – множество рабочих интервалов, которые должны выполняться на одном процессоре с рабочим интервалом *p*;

- множество рабочих интервалов, для которых запрещено выполнение на одном процессоре с рабочим интервалом *p*;

*SP* – множество всех процессоров;

- процессор, на который распределён рабочий интервал *p* в расписании *HP*.

*Алгоритм выполнения операции O2:*

Шаг 1. Случайным образом выбирается рабочий интервал *p*.

Шаг 2. Строится множество процессоров , на которые возможно перенести рабочий интервал *p*: .

Шаг 3. Случайным образом выбирается процессор из и на него переносятся все рабочие интервалы из множества .

*Алгоритм выполнения операции O1:*

Шаг 1. Случайным образом выбирается рабочий интервал *p*.

Шаг 2. Определяется допустимый диапазон ярусов . Этот диапазон выбирается таким образом, что рабочий интервал *p* может быть перенесён на любой ярус из найденного диапазона.

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

**Распределение областей по узлам вычислительной системы**

Пусть *n* – количество областей разбиения,

*m* – количество узлов вычислительной системы, осуществляющих поиск решений в областях *(n > m)*.

Области упорядочиваются по критическому пути в графах.

В *i*-й узел *(1im)* будут распределены области с номерами



*j : j mod m = i (1jn)*.



**1**

**2**

**3**

**m**

**m+1**

**m+2**

**m+3**

**2m**

**n-m-1**

**n-m**

**n-m**

**n-2**

**n**

**n-1**

Proc 1

Proc 2

Proc 3

Proc m

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

Рис. Отсечение областей в результате обмена с другими узлами.

Рис. Сравнение времени работы алгоритмов.

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

**Лекция 8**

**Простой генетический алгоритм (алгоритм Холланда)**

Holland J.N. Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan: Univ. of Michigan Press, 1975.

*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.

Основные операции алгоритма: селекция, скрещивание и мутация выполняются над элементами популяции.

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.

*Кодирование решений*:

1. Однозначность, т.е. каждая закодированная строка должна соответствовать единственному решению исходной задачи.
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.

Для задач непрерывного и целочисленного математического программирования, оптимизируемые параметры задаются:

* двоичным кодом числа,
* 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.

1. Выделять каждой строке количество потомков в соответствии со значением её функции выживаемости:

* если *f*(*x1*)*>f*(*x2*)*,* то вероятность того, что *Ch*(*x1*)*≥Ch*(*x2*)должна быть достаточно велика. Здесь *f*  – целевая функция, *x1* и *x2* – строки популяции, *Ch*(*x*) – количество потомков, присваиваемых строке *x*.

1. Обеспечивать сохранение в популяции лучших решений.
2. 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.

Схема пропорциональной селекции:

1. Вычислить среднее значение функции выживаемости  для популяции.
2. Для *i*-ой строки популяции выделить  потомков, где *Fi* – значение функции выживаемости *i*-ой строки.
3. Из полученных строк сформировать новую популяцию.

Схема рулетки:

1. Вычислить среднее значение функции выживаемости  для популяции.
2. Для *i*-ой строки популяции выделить сектор рулетки с центральным углом , где *Fi* – 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.

*Операция скрещивания*:

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 ().

Одноточечное скрещивание:

1. Вся популяция случайным образом разбивается на пары.
2. Для каждой пары случайным образом генерируется число ,
   * если , 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.

1 … L

Fi



Fj

*Операция мутации* заключается в инвертировании каждого бита с заданной вероятностью:

1. Для каждого бита генерируется случайное число .
2. Если , то бит инвертируется.

(бит ивертируется, если )

Вместо того, чтобы вычислять мутацию для каждого бита, вычислялось расстояние от текущего бита по формуле (\*). Бит, отстоящий на вычисленное расстояние обязательно инвертировался.

, (\*)

*m=*, *pm* - вероятность мутации,

*xi* - равномерно распределённая в промежутке [0, 1] случайная величина,

,

*n* - целое число.

*Критерий останова*:

1. выполнение алгоритмом априорно заданного числа итераций;
2. выполнение алгоритмом априорно заданного числа итераций без улучшения функции выживаемости;
3. достижение некоторого априорно заданного значения функции выживаемости.

*Эволюционные алгоритмы* используют целочисленное или вещественное кодирование решений.

Задачи возникающие при построении ГА и ЭА:

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.

1. Задание целевой функции и критерия останова.

**Теория схем. Гипотеза строительных блоков**

*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.

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

* у схемы \*\*000 определяющая длина равна 2,
* у схемы 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*.

Представим схему c *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.

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:

,

 - ожидаемое количество примеров схемы *h* на шаге *t+1*,

 - среднее значение целевой функции схемы *h* на шаге *t*,

 - средние значение целевой функции в популяции на шаге *t*,

 - определяющая длина схемы *h*,  - порядок схемы *h*,

 - вероятность мутации,  - вероятность скрещивания,

*l* - количество бит в строке.

Выражение  определяет ожидаемое число примеров схемы *h* в новой популяции.

Выражение  определяет вероятность выживания схемы *h* после выполнения операций скрещивания и мутации:

* слагаемое  определяет вероятность того, что пример схемы *h* будет разрушен скрещиванием,
* слагаемое  определяет вероятность того, что пример будет разрушен мутацией.
* Мутация с большей вероятностью разрушает схемы высокого порядка.
* 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 ,  или уменьшение меры давления селекции увеличивает разнообразие популяции, но не позволяет использовать все хорошие схемы, имеющиеся в популяции.
* Уменьшение ,  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”.

С*троительные блоки* это схемы:

* с короткими определяющими длинами,
* низким порядком,
* высокими значениями средних целевых функций.

*Гипотеза строительных блоков* утверждает, что комбинирование хороших строительных блоков даёт хорошую 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.

**Описание динамики работы ГА с помощью цепей Маркова.**

Данный подход предполагает, что строится система детерминированных уравнений, которая определяет вероятность того, что каждая конкретная строка будет присутствовать в популяции в момент времени *t* c 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):

* мутация является линейной операцией, она описывается матрицей размера 2*l*×2*l*,
* операция скрещивания является двухместной и описывается тензором размера 2*l*×2*l*×2*l*.

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.

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.

**Лекция 9**

См. презентацию

Модель прикладной программы определим:

1. Множеством рабочих интервалов процессов, составляющих программу *PR*:



1. Частичным порядком на *P*:

*={ik=(pi,pk)}(i,k)∈(1...N)*;

1. Вычислительной сложностью рабочих интервалов:

;

1. Объемом данных обмена для каждой связи из **:

*{vik}(i,k)∈(1...N)*;.

*Расписание выполнения программы* определено, если заданы:

* привязка,
* порядок.

Модель расписания выполнения программы определим набором простых цепей и отношением частичного порядка *HP* на множестве *P*:

*HP=({SPi}i=(1...K) ,HP)*.

*{SPi}i=(1...M)* -набор простых цепей (ветвей параллельной программы).

Отношение частичного порядка *HP* на множестве *P* для *HP* определим как объединение отношений: *HP*=*c*∪*1*∪…∪*M*, *i* - отношение полного порядка на *SPi*, которое определяется порядковыми номерами рабочих интервалов в *SPi*; *c* - набор секущих ребер, которые определяются связями рабочих интервалов, распределенных на разные процессоры.

Расписание *HP* является корректным, если выполнены следующие ограничения:

1. Каждый рабочий интервал должен быть назначен на процессор (в *SPi*).
2. Каждый рабочий интервал должен быть назначен лишь на один процессор (в один *SPi*).
3. Частичный порядок, заданный в *H* сохранен в *HP*: 
4. Расписание *HP* должно быть беступиковым. Условием беступиковости является отсутствие контуров в графе *HP*: .
5. Все рабочие интервалы одного процесса должны быть назначены на один и тот же процессор (в один и тот же *SPi*).

*Дано:*

* 1. *H(PR)=(P,)*- модель программы,
  2. *T=f(HP,HW)*- функция вычисления времени выполнения расписания *HP* на архитектуре *HW*,
  3. - директивный срок выполнения программы.

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



 - число процессоров, на котором выполняется расписание.

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.

*Параметрическое представление расписаний с использованием приоритетов*

Расписание задается вектором *YN+K* :







*K* - число процессов в *H*,

*Ni* – число рабочих интервалов в *i*-ом процессе,

 - операция построения вектора  из параметров  и .

 содержит номер процессора, на котором выполняются рабочие интервалы *i*-го процесса.

 используются алгоритмом восстановления расписания (определение отношения полного порядка в каждом *SPi*) в качестве приоритетов рабочих интервалов.

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

Mножество рабочих интервалов разобьем на три подмножества: :

* *P1* - множество рабочих интервалов распределенных в *SP* 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*.

1. Начальное разбиение множества *P*:

* *P1=∅*;
*  - множество рабочих интервалов в *H* без предшественников;
* - множество рабочих интервалов в *H*, у которых имеются предшественники.

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

* размещаем его в конец соответствующего списка *SP* (номер списка определяется значением параметра );
* 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≠∅*, то к п.2, иначе завершить работу.

*Утверждение.* Алгоритм *A1* восстановления расписания по его параметрическому представлению *YN+K* получает расписание , и расписание восстанавливается однозначно.

*Теорема.* Любое допустимое расписание  может быть задано параметрическим представлением с использованием приоритетов (*YN+K*) и однозначно восстановлено алгоритмом *A1*, если допустимая верхняя граница *L* значений параметров  больше или равна числу рабочих интервалов (*L≥N*).

*Кодирование решений*

 – закодированное решение,

 – привязка,

 – приоритеты,

 – оператор конкатенации строк.

П*реобразование решения*

1. Параметры  могут принимать целочисленные значения в диапазоне *[1,…,K]*,
2. Параметры  могут принимать целочисленные значения в диапазоне *[1,…,N]*,
3. количество изменяемых параметров может быть 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 операции преобразования решения, удовлетворяющие условиям 1-3, и алгоритм восстановления *A1* позволяют получить любой допустимый вариант решения.

*Операции мутации и скрещивания*

Поскольку ограничения на изменения значений параметров расписаний задаются в виде , то могут быть использованы известные операции мутации и скрещивания для целочисленного кодирования решений из Приложения 1.

*Операция селекции*

Для выполнения селекции используется комбинация схемы пропорциональной селекции и схемы рулетки:

* если  или это первая итерация алгоритма, то запоминаем  как лучшую строку  и переносим ее в новую популяцию,
* для вычисления целого числа потомков используется схема пропорциональной селекции,
* для распределения остатка – схема рулетки.

*Функция выживаемости*

*F*(*kt, ke*)*=C1kt+C2ke*,

*С1+С2=*1, *Ci*≤0 (*i*=1,2)



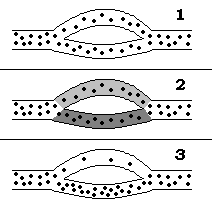


*Критерий останова*

Выполнение алгоритмом априорно заданного числа итераций без улучшения функции выживаемости.

**Лекция 10**

**Концепция построения алгоритмов (биологическая модель)**



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.

**Общая схема работы муравьиных алгоритмов**

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.

**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:



*ij*(*t*) - количество феромона на ребре (*i,j*),

 - значение локальной целевой функции на ребре (*i,j*),

*Lk* – множество вершин, включенных в табу-список муравья *k,*

** и ** - параметры алгоритма, определяющие важность феромонного следа и локальной целевой функции.

**Операция обновление количества феромона на ребрах**





*Tk*(*t*) – путь, построенный *k*-м муравьем,

*F*(*T*) – целевая функция, определяющая качество пути,

*m* – количество муравьев,

*p*[0,1] – коэффициент испарения феромонов.

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).

Модификации муравьиных алгоритмов были разработаны с целью устранить 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.

**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: ;
3. Изначально количество феромона на всех ребрах равно *max*.

Изначально одинаковое количество феромона на всех ребрах уменьшает вероятность выбора одного и того же маршрута разными муравьями.

Условие  не позволяет одному относительно хорошему решению доминировать над другими.

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 *.*

**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:

, где *d* – доля поглощаемого феромона.

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

**Усиление сходимости к оптимуму в его окрестности.**

***Элитные муравьи.***

Элитные муравьи откладывают феромоны только на ребрах наилучшего маршрута найденного сначала работы алгоритма.

***Ранговый муравьиный алгоритм.***

Найденные на каждой итерации решения ранжируются.

Только *(w-1)* лучших муравьев и один элитный откладываю феромоны.

**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.

**Список кандидатов**

Используется для решения задач большой размерности.

Список кандидатов представляет собой небольшой список предпочтительных вершин, в которые может перейти муравей из очередной вершины.

Муравей выбирает вершину не из списка кандидатов только тогда, если он прошел его уже полностью.

**Штовба С.Д. Муравьиные алгоритмы: теория и применение// Программирование. 2005. №4.**

**Задача построения статико-динамических расписаний**

Пусть задано множество работ:

*SW=*{*ai=<si,fi,ti,pi>|i*[1*..n*]}, где

* [*si,fi*) – директивный интервал;
* *ti* – время выполнения работы;
* *pi* – номер раздела работы;
*  и , определяющие, временные затраты на переключение контекста между окнами и резерв свободного времени внутри каждого окна.

Требуется построить расписание, представляющее собой набор окон:

*SP=*{*wi*=*<Si,Fi,SWi>|i*[1*..m*]}, где

* *Si* – время открытия окна;
* *Fi* – время закрытия окна;
* *SWi=*{*aij*}*SW* – множество работ, выполняемых внутри окна.

**Ограничения корректности расписания:**

1. *(i,j)*[1*..m*],*i**j*:*SWi**SWj*= – множества работ, размещенных внутри разных окон, не пересекаются;
2. *i*[1*..n*],*j*[1*..m*],*aiSWj*:*si**Sj*<*Fj**fi* – временной интервал окна лежит внутри директивных интервалов работ, выполняющихся в окне;
3. *(i,j)*[1*..n*],*k*[1*..m*],*ai,ajSWk*:*pi=pj* – разделы работ, размещенных внутри одного окна, совпадают;
4. *(i,j)*[1*..m*],*i<j*:*Sj**Fi+* – окна не пересекаются и между любыми двумя соседними окнами есть промежуток не короче времени, необходимого на переключение контекста;
5.  – суммарное время выполнения всех работ из одного окна с учетом резерва времени не больше, чем длина окна.

**Способ сведения задачи построения статико-динамического расписания к задаче нахождения на графе маршрута**

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

* *N=*{*ni|i*[1*..n*]}{*O*} – множество вершин;
* *A=*{(*ni,nj*)*|i,j*[1*..n*],*i**j*}{(*O,ni*)*|i*[1*..n*]}*>* – множество ребер.

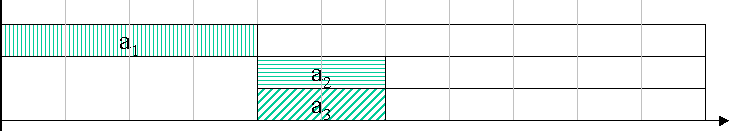
Каждой вершине графа соответствует одна из размещаемых работ. Кроме того, добавляется еще одна вершина *О*, соответствующая началу расписания.

**Алгоритм для построения расписания по такой последовательности работ**

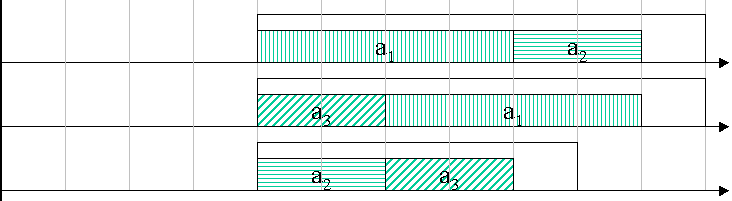
Пусть дана последовательность работ *SL*={*aik|aik**SW;i,k*[1*..n*]}. Первый индекс означает номер работы, второй - номер места работы в последовательности. Построим расписание *SP* по следующему алгоритму:

1. Инициализация расписания: *Time*=0 – текущая длина расписания; *k*=1 – номер места размещаемой работы в *SL*; *SW0*= – список работ в добавляемом окне;
2. Установка начальных параметров окна: *S*=max(*Time,sik*), *F*=*fik* – границы добавляемого окна; *T*=2 – минимальная необходимая длина окна с учетом добавленных работ; *R*=*rik* – раздел для работ в окне;
3. Обновление значений параметров окна с учетом новой добавляемой работы: *S’=*max(*S,sik*), *F’=*min(*F,fik*)*, T’=T+tik*;
4. Добавление работы в текущее окно: если *rik=R , T’**F’–S’* (условия корректности не нарушаются), то: *S=S’, F=F’, T=T’, SW0=SW0*{*aik*}*, k=k+*1, если *k**n* (список *SL* еще не пройден), переход к п.3;
5. Проверка возможности добавления работы в новое окно: если *fik–F–*<*tik ,* то *k=k+*1, если *k**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);
6. Закрытие окна: *F=S+T –* устанавливаем время закрытия окна минимально возможным;
7. Добавление данного окна в расписание: *SP=SP*{*<S,F,SW0>*};
8. Пересчет длины расписания: *Time=F+**, k=k+*1;
9. Если *k*  *n* (список еще не пройден), переход к п.2.
   * Условие корректности 1 выполняется, т.к. каждая вершина встречается в построенном маршруте ровно один раз, и соответствующая ей работа размещается лишь в одно окно;
   * Условие 2 выполняется, т.к. *S’=*max(*S,sik*)*sik*; *F’=*min(*F*,*fik*)*fik* (п.3);
   * Выполнение условий 3 и 5 обеспечивается проверками в п.4 алгоритма – если ограничения нарушаются, работа не размещается в данное окно;
   * Условие 4 выполняется, т.к. *Si+*1*Time* (п.2), где *Time=Fi+*п.8

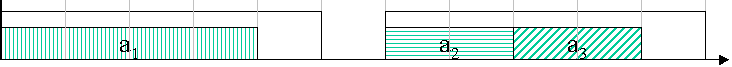
*SW=*{*a*1=<0,11,4,1>,*a*2=<4,11,2,1>,*a*3=<4,11,2,1>}, ==1



Построенные расписания по различным маршрутам.



Оптимальное расписание.



Можно выдвинуть гипотезу, что не существует детерминированного полиномиального алгоритма *A* построения расписания по маршруту в графе *G* такого, что для *(SW,,):SL:A(SL)=SP0,* 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.

**Локальная целевая функция на ребрах графа *G***

,

 – разница во времени между минимальным временем завершения *i*-й работы и максимальным временем начала *j*-й работы.

* Если *θ* < 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.

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

Построим граф *G’*=*<N’,A’>*,

*N’=*{*ni+,ni*–*|i*[1*..n*]}{*O*} – множество вершин; вершина *ni+* соответствует размещению работы без закрытия текущего окна, *ni*– – размещению работы с закрытием окна. Вершина *О* является началом всех маршрутов.

*A’=*{(*ni+,nj*–),(*ni+,nj+*),(*ni*–*,nj*–)*|i,j*[1*..n*]*,i**j*}{(*O,ni+*), (*O,ni*–)*|i*[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 *nj*– текущее окно закрывается принудительно.

Если *SP0* 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 *SP0*

* по количеству размещенных в нем работ,
* по количеству окон,
* по множеству работ выполняемых внутри каждого окна,

*но может отличаться* по времени открытия и закрытия окон.

**Алгоритм построения расписания обменов по каналу с централизованным управлением (схема с подциклами)**

Канал в данной системе может рассматриваться как одноприборное устройство, обслуживающее исходно заданный набор работ без прерываний.

Дано:

* Множество работ, которые должны выполняться на системе . Для каждой работы заданы - время выполнения, - директивный интервал выполнения и выполняется условие ;
* Дополнительные ограничения на корректность расписания ;
* Вектор значений технологических требований .

Расписание выполнения работ, которое представляет собой упорядоченное множество



Здесь *k* - порядковый номер *j*-ой работы в расписании,

 - время начала выполнения *j*-ой работы в расписании ,

 - время завершения выполнения *j*-ой работы.

Множество корректных расписаний  определим набором ограничений:



1. Построение муравьями путей в графе.
2. Применение алгоритма размещения работ в расписание, на вход которому подаются последовательности вершин, построенные муравьиным алгоритмом.
3. Вычисление целевой функции.
4. Обновление количества феромона на ребрах графа.
5. Если условие останова не выполнено, переход к п.1.

Сопоставим каждой работе из *J* вершину *ri.*

Введем специальную вершину *О*, с которой будет начинаться маршрут каждого муравья.

Получим полносвязный ориентированный граф 

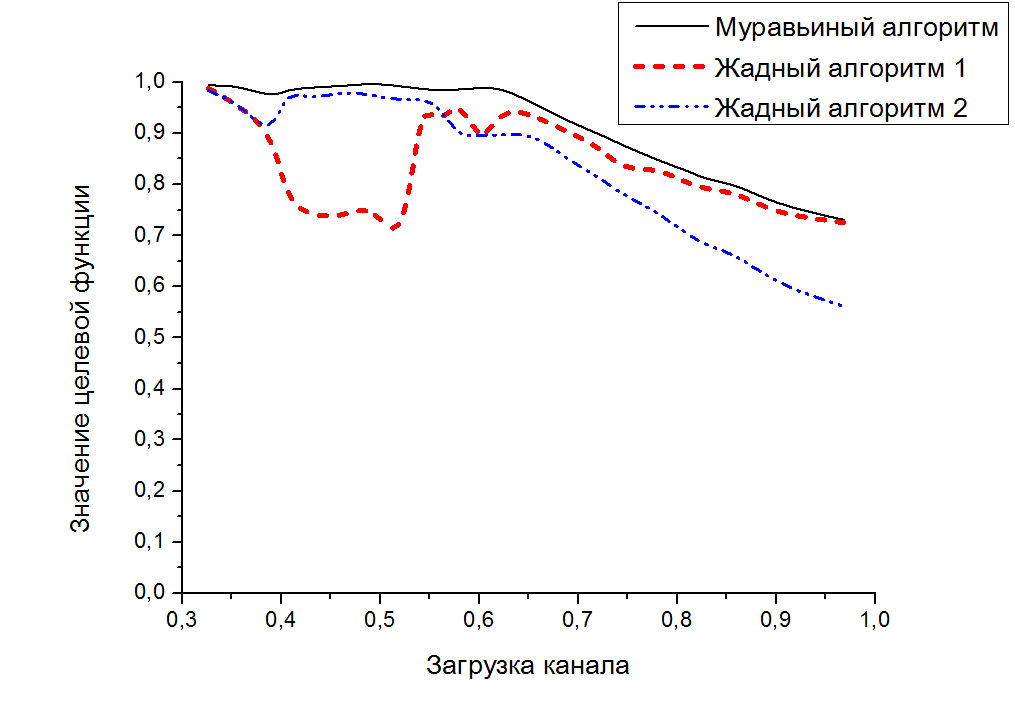
* – множество вершин,
* – множество ребер (между любыми двумя вершинами есть два ориентированных ребра).

**Утверждение.** Каждому пути в графе *G* соответствует ровно одно расписание и для одного и того же расписания может существовать более одного пути, по которому его можно построить.

Локальные эвристические функции на ребрах графа:

1.  – чем ýже директивный интервал работы, тем раньше её предпочтительно разместить;
2. , - чем ýже директивный интервал работы и чем больше времени нужно на её выполнение, тем раньше её предпочтительно разместить;
3. , где *NS –* количество подциклов, в которые может быть размещена *j-*я работа – чем меньше это число, тем раньше работу предпочтительно разместить;

Сравнение алгоритмов на классе исходных данных А.



Сравнение алгоритмов на классе исходных данных Б.

