МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ЗАПОРОЖСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ

И.В. Козин

Эволюционные модели в дискретной оптимизации

Монография

Запорожье, 2019

ББК

УДК 519.8

Рецензенти:
Чл.кореспондент АН України, доктор фізико-математичних наук, професор
Кісельова О.М.

Доктор фізико-математичних наук, професор
Перепелиця В.О.

Видається за рішенням Вченої ради Запорізького національного
Університету (пр.№___ від _____р.)

Козін І.В.

Еволюційні моделі в дискретній оптимізації

Запоріжжя, 2017 - 128(рос)

Исследованы эволюционные модели для поиска приближенных решений оптимизационных задач. Установлены свойства моделей, связанные с понятием наследственности. Показано связь между наследственностью и геометрическими свойствами кроссовера в эволюционных моделях. Исследованы фрагментарные(мозаичные) модели дискретных оптимизационных задач. На основе понятия фрагментарной структуры предложена универсальная эволюционно-фрагментарная модель для поиска приближенных решений задач дискретной оптимизации.

Для специалистов в области математического моделирования и теории принятия решений, а также преподавателей, аспирантов и студентов математических специальностей.

Досліджено еволюційні моделі для пошуку наближених розв’язків оптимізаційних задач. Встановлено властивості моделей, пов'язані з поняттям спадковості. Показано зв'язок між спадковістю і геометричними властивостями кросоверу в еволюційних моделях. Досліджено фрагментарні(мозаїчні) моделі дискретних оптимізаційних задач. На основі поняття фрагментарної структури запропонована універсальна еволюційно-фрагментарна модель для пошуку наближених розв’язків задач дискретної оптимізації.

Для фахівців в області математичного моделювання й теорії прийняття рішень, а також викладачів, аспірантів і студентів математичних спеціальностей .

УДК 519.8

ISBN

Содержание

Список условных обозначений

$\forall, \exists$ - кванторы всеобщности, существования

$\Leftrightarrow$ - необходимо и достаточно, тогда и только тогда

$\Rightarrow$ - следует, вытекает, с необходимостью

$\rightarrow$ - отображение множеств

$\in$ - принадлежность множеству

$\varnothing$ - пустое множество

$\subseteq$ - подмножество (включение)

$\cup$ - объединение

$\cap$ - пересечение

$|X|$ - мощность (число элементов) множества X

R^n - эвклидово n-мерное пространство

$\times$ - прямое(декартово) произведение

S_n - группа подстановок n элементов (симметрическая группа)

$[x_1,x_2]$ - отрезок метрического пространства

K_n - полный n-вершинный граф

Введение

Одним из новых направлений математического моделирования является создание и исследование математических методов поиска оптимальных решений на основе механизмов, которые использует природа [20]. Исследования в области естественных(природных) алгоритмов оказались настолько перспективным, что сегодня можно указать целые теории, которые созданы в рамках этого направления. Нейронные сети, эволюционное моделирование, методы отжига, метод муравьиной колонии, имитационное моделирование и др.

Поразителен тот факт, что на основании очень простых принципов природе удается находить достаточно удачные решения для ряда задач. Причем некоторые технические решения обладают настолько высокой эффективностью, о которой человек сегодня может только мечтать. Великолепными примерами природной оптимизации является живые организмы. Каждый живой организм великолепно приспособлен к среде обитания. Таких замечательных результатов природа добивается благодаря естественным механизмам эволюции. Это механизмы наследственности и естественного отбора.

В предлагаемой работе рассматривается одно из направлений математического моделирования, которое за основу поиска оптимальных решений берет принципы природной эволюции. В рамках этого направления развивается теория генетических алгоритмов, эволюционные вычисления, эволюционное программирование.

Главные задачи книги: показать общие принципы эволюционного моделирования, выявить основные свойства и классы эволюционных моделей, применить методы эволюционного моделирования для задач дискретной оптимизации.

По-видимому, началом эволюционного моделирования в математике следует считать работы различных групп кибернетиков в 60-х годах XX-го века, которые независимо друг от друга исследовали возможности применения принципов биологической эволюции к решению различных технических проблем.

Одной из первых работ в этой области была работа Холланда [39,64], в которых рассматривались модели естественной эволюции и использовался генетический алгоритм для решения некоторых оптимизационных задач. Впоследствии в многочисленных работах [4,31,46,49,56,58,59,72,78,79] области применения генетических алгоритмов были значительно расширены.

Однако в середине XX-го века в математике господствовала идея о создании эффективных алгоритмов поиска оптимальных решений с оценками сходимости. Поэтому работам Холланда и его последователей в то время не было уделено достаточного внимания.

В конце XX-го века интерес к эволюционным моделям, в частности к генетическим алгоритмам, просыпается с новой силой. Наверное, это объясняется тем, что наука вышла на рубежи решения сложных задач [9], для которых неизвестны эффективные численные методы поиска оптимальных решений, либо реализация этих методов очень сложна. С другой стороны, благодаря развитию вычислительных средств, появилась возможность создания адекватных моделей очень сложных систем. Эволюционные модели в этом смысле достаточно универсальны и подходят для большинства прикладных задач. Поэтому сегодня наблюдается экспоненциальный рост числа работ, посвященных эволюционному моделированию. Правилом «хорошего тона» во многих прикладных исследованиях стало построение соответствующей эволюционной модели для решения конкретной оптимизационной задачи.

К сожалению, методы поиска оптимальных решений в эволюционных моделях являются метаэвристиками. Получить какие-либо гарантированные результаты с их помощью не представляется возможным. Однако пример живой природы показывает, что подобные модели содержат в себе определенную «изюминку» - свойства, которые позволяют рассчитывать на очень хорошие результаты при сравнительно небольших вычислительных затратах. Именно поэтому интерес к эволюционному моделированию не ослабевает и в ближайшие годы следует ожидать лишь активизации исследований в этом направлении.

Что же привлекает в эволюционных моделях?

Во-первых, это относительная простота построения модели. Фактически, для того чтобы построить математическую модель для поиска решения оптимизационной задачи достаточно представить пространство допустимых решений в какой-либо стандартной кодировке. Во-вторых, универсальность модели. Эволюционная модель может применяться практически для любой оптимизационной задачи. В-третьих, небольшое количество составляющих элементов позволяет легко создавать программные продукты для реализации эволюционных моделей. В-четвертых, возможность решения задач большой размерности. В-пятых, методы эволюционного моделирования позволяют вести поиск глобального оптимума на всем пространстве допустимых решений. Эти и ряд других свойств обуславливают широкую применимость эволюционных моделей для поиска решений сложных прикладных задач.

В конце прошлого столетии методы эволюционного моделирования вошли в практику дискретной оптимизации [16,35,44,58,59,80]. С тех пор число работ, связанных с дискретными задачами, в которых используются принципы эволюционного моделирования, непрерывно растет. Тем более, что для ряда прикладных задач эволюционная модель практически единственный способ добиться приемлемых результатов при поиске оптимальных решений.

Методы эволюционного моделирования могут успешно применяться и при исследовании многокритериальных задач дискретной оптимизации [26,39,56,78]. В этом случае задача заключается в поиске подмножества множества Парето ограниченной мощности.

Хорошие результаты можно ожидать при комбинировании эволюционного моделирования с другими методами поиска оптимальных решений. Перспективным является использование гибридных алгоритмов, которые соединяют в себе преимущества генетических алгоритмов и методов локального поиска [44,46,63].

Можно сказать, что исследования в области эволюционного моделирования сегодня переживают стадию бурного роста и накопления экспериментального материала.

В настоящей работе предпринята попытка объяснить действие эволюционных моделей на основе принципов наследственности. Это дает возможность значительно расширить класс алгоритмов, которые можно отнести к разряду эволюционных и, соответственно, увеличить число задач, для которых оправдано применение эволюционных моделей.

Другим важным результатом настоящего исследования является подход к решению ряда дискретных оптимизационных задач на основе эволюционно-фрагментарной модели [24]. С помощью теории фрагментарных структур удалось построить эволюционную модель, которая является универсальной для многих классов дискретных оптимизационных задач. А именно: задач, допускающих фрагментарную структуру решений.

В работе автор старался избегать слишком большой «биологизации» изложения, учитывая, что методы и модели, которые исследуются, все-таки являются математическими и аналогия с биологическими конструкциями весьма условна. Поэтому здесь вы не найдете терминов «генотип», «фенотип», «аллель», «ген», «хромосома» и многих других, которыми изобилуют работы по эволюционному моделированию. Не приводятся подробные описания классических эволюционных моделей, с которыми легко можно ознакомиться в литературе[5,15,46,72].

Предполагается, начальное знакомство читателя с основными понятиями эволюционного моделирования, которым посвящены многочисленные публикации[2,4,5,15,46,72].

Хочется поблагодарить многих коллег, без постоянного внимания которых работа над книгой была бы невозможной. Особая благодарность профессору Перепелице В.А. Беседы с ним породили целый ряд идей, реализованных в этой книге. Многочисленные обсуждения в институте кибернетики АН Украины, на кафедре Экономической кибернетики Запорожского национального университета (зав. д.э.н. Максишко Н.К.), доклады на многочисленных конференциях и семинарах позволяют надеяться на хорошую апробацию результатов исследований. Большую помощь в подготовке книги оказали мои аспиранты.

Надеюсь, что это труд окажется полезным и нужным как для развития теории эволюционного моделирования, так и для создания приложений, которые используют идеи и методы эволюционного моделирования в решении практических задач.

И.Козин

1 ОБОБЩЕННАЯ ЭВОЛЮЦИОННАЯ МОДЕЛЬ ДЛЯ ЗАДАЧИ ОПТИМИЗАЦИИ

На сегодня уже сформировался единый подход к понятию эволюционной модели, который обобщает концепции, заложенные в работах основоположников эволюционного моделирования. Формализация этого подхода предложена ниже.

1.1 Эволюционная модель

Рассмотрим классическую постановку однокритериальной задачи оптимизации. На множестве V найти точку v^*, в которой достигает максимального(минимального) значения функция $\Phi : V \to R^1$.

Для поиска приближенного решения задачи оптимизации переходят к математической модели задачи. В модели исходное множество V заменяется некоторым множеством достаточно простой природы – базовым множеством решений, а целевая функция, соответственно, заменяется функцией, заданной на базовом множестве решений.

Обобщая подходы к эволюционному моделированию, которые использовались в различных работах [57,72,78], определим эволюционную модель следующим образом.

Определение 1.1 Эволюционная модель оптимизационной задачи включает следующие составляющие.

1. Базовое множество решений X, на котором осуществляется поиск оптимального решения. Конечные непустые подмножества множества X будем называть популяциями.

2. Критерий – функция $f : X \to R^1$, заданная на базовом множестве X со значениями в R^1 (возможны и многокритериальные модели с целевыми функциями $f : X \to R^n$).

3. Правило построения начальной популяции: оператор, который выделяет на множестве X его непустое подмножество $Y_0 \subseteq X$.

4. Правило селекции – правило выбора, которое позволяет выделить для любой популяции $Y \subseteq X$ множество пар элементов-родителей в этой популяции для последующего выполнения операции кроссовера.

5. Правило кроссовера(скрещивания) задается оператором $Kr : X \times X \to X$, который по двум решениям-родителям строит новое решение-потомок (или несколько таких решений) из множества *X*.

6. Правило мутации задается оператором мутации $M : X \to X$ и числом $\alpha_{mut} \in [0,1]$ - вероятностью мутации. Для любой популяции *Y* множество потомков этой популяции получается путем последовательного применения правил селекции, кроссовера и мутации.

7. Правило отбора – правило выбора, позволяющее получить новую популяцию из объединения множеств родителей и потомков. Обычно это правило задается с помощью указания размера (числа элементов) популяции и линейного порядка или функции – критерия отбора. Элементы объединения множеств родителей и потомков упорядочиваются и выбираются те, у которых значение критерия наибольшее.

8. Правило остановки - правило, определяющее условие остановки алгоритма.

Поиск приближенного оптимального решения в рамках эволюционной модели осуществляется с помощью эволюционного алгоритма, общая схема которого приводится ниже.

1.2 Эволюционный алгоритм

Приведем описание эволюционного алгоритма для поиска приближенных решений задачи оптимизации в рамках рассматриваемой эволюционной модели. Шаг эволюционного алгоритма, который будем называть в дальнейшем шагом эволюции, состоит в следующем:

Пусть на *k*-м шаге определена текущая популяция(k-е поколение) $Y_k \subseteq X$. На начальном шаге с номером 0 это начальная популяция Y_0, которая находится по правилу построения начальной популяции.

С помощью правила селекции выбирается множество пар-родителей $P_k \subseteq Y_k \times Y_k$ в текущей популяции Y_k. К каждой родительской паре применяется оператор кроссовера и строится множество $C_k = Kr(P_k)$. Ко всем элементам множества C_k применяется правило мутации. А именно: каждый элемент $c \in C_k$ с вероятностью α_{mut} заменяется на элемент $c' = M(c)$ или с вероятностью $1 - \alpha_{mut}$ остается без изменений. Таким путем строится множество потомков C'_k.

Далее применяется правило отбора. Элементы объединения множеств $Y_k \cup C'_k$ упорядочиваются по убыванию (или возрастанию) значений критерия. Первые *m* из них (*m* размер популяции) образуют популяцию Y_{k+1}.

Проверяется условие остановки. Если условие не выполнено, то осуществляется переход к очередному шагу алгоритма с начальной популяцией Y_{k+1}. При выполнении условия остановки алгоритм заканчивает работу. Приближенным решением задачи считается «лучшее» по значению целевой функции решение в последней популяции.

Эволюционный алгоритм работает следующим образом. На *k*-м шаге определена текущая популяция $Y_k \subseteq X$. На начальном шаге это начальная популяция Y_0, которая находится по правилу построения начальной популяции.

Часто в эволюционных моделях в качестве базового множества рассматривается конечное множество слов фиксированной длины над некоторым алфавитом. В частности, в классической модели Холланда [64] базовое множество - множество бинарных последовательностей фиксированной длины. Во многих более поздних моделях базовое множество – множество целочисленных (а иногда и нецелочисленных) векторов заданной размерности.

1.3 Пример. Знаковый кроссовер

В качестве примера рассмотрим часто используемое правило кроссовера на множестве слов длины *N* в некотором конечном алфавите. Пусть заданы два слова – родителя $a = a_1 a_2 ... a_N$ и $b = b_1 b_2 ... b_N$. Выберем некоторый набор индексов $\{i_1, i_2, ..., i_s\} \in \{1, 2, ..., N\}$. Каждую букву a_{i_k} в слове *a* заменим соответствующей буквой b_{i_k} из слова *b* и, наоборот, каждую букву b_{i_k} в слове *b* заменим соответствующей буквой a_{i_k} из слова *a*. Полученные слова являются результатом применения правила кроссовера.

Правило мутации в этой модели – в слове случайно выбранная буква заменяется другой (тоже случайно выбранной) буквой алфавита.

В многочисленных исследованиях [46,72] в области эволюционного моделирования рассмотрены различные модификации приведенных выше правила кроссовера и правила мутации и приводятся рекомендации по их практическому применению и выбору конкретных значений параметров (размер популяции, вероятность мутации и т.д.)

Естественно, эволюционные алгоритмы являются лишь метаэвристиками. Вопрос о сходимости, и об оценке качества этих алгоритмов на сегодняшний день открыт. Поэтому для оценки качества эволюционных моделей с заданными свойствами используют методы имитационного моделирования на основе генерации многочисленных индивидуальных задач и сравнения методов эволюционного моделирования с результатами применения других методов для этих задач. Другим широко распространенным способом проверки качества моделей является апробация этих моделей на различных классах тестовых примеров, которых на сегодняшний день наработано достаточно много[53,80].

2 ЭВОЛЮЦИОННАЯ МОДЕЛЬ ХОЛЛАНДА И КЛАССИЧЕСКИЙ ГЕНЕТИЧЕСКИЙ АЛГОРИТМ

Одной из первых эволюционных моделей была модель Холланда, которая построена на определенной аналогии с природными процессами эволюции. Впоследствии эта модель видоизменялась, совершенствовалась. Однако базовое множество модели оставалось прежним – это пространство бинарных последовательностей заданной длины $\{0,1\}^n$.

2.1 Модель Холланда

В классической модели Холланда [64] допустимые решения оптимизационной задачи представляются бинарными последовательностями фиксированной длины *N*. Пусть U – множество допустимых решений задачи $F(u) \to \max\limits_{u \in U}$. Кодом будем называть инъективное отображение W этого множества во множество $\{0,1\}^N$. Кодом элемента $u \in U$ называется его образ $W(u)$. Образ $X=W(U)$ множества допустимых решений будем называть множеством допустимых кодов. Именно это множество является базовым для эволюционной модели оптимизационной задачи. Для простоты суперпозицию целевой функции и кода будем также обозначать $F : X \to R^1$. Кроме того, будем предполагать также, что $\forall x \in X \quad F(x) > 0$.

Опишем другие составляющие классической модели.

Начальная популяция Y_0 формируется случайным образом. Вероятность вхождения элемента из X в начальную популяцию равна $1/|X|$.

Правило селекции – пропорциональное, на *k*-м этапе вероятность выбора элемента для скрещивания пропорциональна значению целевой функции на этом элементе, то есть вероятность $P(y)$ выбора элемента $y \in Y_k$ для скрещивания определяется формулой

$$P(y) = \frac{F(y)}{\sum_{z \in Y_k} F(z)}$$

Опишем теперь одноточечное правило кроссовера в модели Холланда. Пусть заданы два допустимых решения в виде слов $x = (x_1, x_2, ..., x_n)$ и $y = (y_1, y_2, ..., y_n)$. Случайным образом выбирается номер $i \in \{1, 2, ..., n\}$. Из двух решений-родителей строятся два новых решения-потомка $x' = (x_1, x_2, ..., x_i, y_{i+1}, ..., y_n)$ и $y' = (y_1, y_2, ..., y_i, x_{i+1}, ..., x_n)$, которые замещают решения-родителей в популяции (рис.1).

Оператор мутации изменяет в решении $x = (x_1, x_2, ..., x_n)$ случайно выбранный элемент *j* по правилу $x_j \to 1 - x_j$.

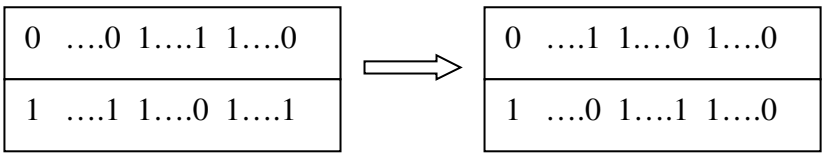


Рис.1 Кроссовер на бинарных последовательностях

Правило отбора удаляет из популяции решения-родители после выполнения процедуры кроссовера. Правило остановки является стандартным – или по достижению определенного числа поколений, или по достижению предельного времени работы алгоритма.

Приведенную выше эволюционную модель Холланда будем обозначать G_1. Генетический алгоритм, соответствующий модели G_1, получил название классического генетического алгоритма на бинарных последовательностях (КГА).

Как правило, модель G_1 используется при решении задач многомерной безусловной оптимизации в эвклидовом пространстве.

2.2 Недостатки модели Холланда

Практически сразу после опубликования модели Холланда был обнаружен ряд существенных недостатков модели. Первая проблема, которая появляется при

использовании модели Холланда – эта проблема кодирования. Дело в том, что при кодировке координат точек эвклидова пространства с помощью их двоичных представлений происходит нарушение соответствия расстояний. А именно: расстояние между точками в эвклидовом пространстве не соответствует расстоянию Хемминга между бинарными представлениями этих точек. Например, числа 8,7 и 1 имеют бинарные представления соответственно 1000, 0111 и 0001. Расстояние на числовой прямой между числами 7 и 8 равно 1, а расстояние между числами 1 и 8 равно 7. С другой стороны, расстояние Хемминга между бинарными последовательностями 1000 и 0111 равно 4, а между последовательностями 1000, 0001 соответственно 2. Таким образом, малое изменение двоичного кода в расстоянии Хемминга отнюдь не соответствует малому изменению решения в эвклидовом пространстве.

Для преодоления этой трудности рекомендуется при двоичном кодировании использовать бинарную кодировку Грея [77]. В этой кодировке соседние по порядку двоичные числа отличаются лишь в одном двоичном разряде.

2.3 Алгоритмы прямого и обратного преобразования в код Грея

Приведем простой алгоритм преобразования двоичного числа в код Грея. Пусть $b_n b_{n-1}...b_1$ - двоичная запись числа. Тогда код Грея $g_n g_{n-1}...g_1$ для этого числа определяется рекуррентной формулой

$$g_i = b_i + b_{i+1} \quad i = 1,2,...,n-1 \text{ и } g_n = b_n$$

Здесь операция «+» - сложение по модулю 2.

Обратный переход от кода Грея к двоичному числу осуществляется также просто по формуле

$$b_k = \sum_{i=k}^{n} g_i \quad k = 1,2,...,n.$$

Таким образом, при использовании указанных алгоритмов преобразования при переходе к базовому пространству модели решается проблема несоответствия расстояний в эволюционной модели Холланада.

2.4 Условная оптимизация

При отыскании условных оптимумов возникает другая проблема, которая состоит в том, что результат кроссовера и мутации может не быть допустимым кодом даже, если аргументы были допустимыми. Исключением является случай, когда все слова допустимы, или $W(U)=\{0,1\}^N$. Следовательно, для того, чтобы эволюционная модель Холланда была применима к рассматриваемой оптимизационной задаче необходимо каким-то образом видоизменить правила кроссовера и мутации, чтобы гарантировать допустимость результата. Например, брать допустимый код, наиболее близкий к полученному в результате кроссовера и мутации. (относительно метрики Хэмминга). Другим приемом при использовании эволюционной модели в задачах условной оптимизации является переход к безусловной оптимизации, Такой переход можно осуществить с помощью метода штрафных функций [1].

Например, задача минимизации $F(x)\to\min$ с соответствующими ограничениями $c_j(x)\le\varepsilon_j$, $\varepsilon_j>0$, $j=1,2,...,m$, наложенными на x, преобразовывается в задачу поиска минимума без ограничений $Z(x)\to\min$, где $Z(x)=F(x)+P(x)$.

Функция $P(x)$ является штрафной. Необходимо, чтобы при нарушении ограничений она «штрафовала» функцию Z, т.е. увеличивала её значение. В этом случае минимум функции Z будет находиться внутри области ограничений. Функция $P(x)$, удовлетворяющая этому условию, может быть не единственной.

Функцию $P(x)$ удобно записать следующим образом:

$$P(x)=r\sum_{j=1}^{m}p_j(x),$$

где r – положительная величина и

$$p_j(x) = \begin{cases} 0, & \text{если } c_j(x) \le \varepsilon_j \\ 1, & \text{если } c_j(x) > \varepsilon_j \end{cases}.$$

Тогда задача оптимизации для функции $Z = Z(x, r)$ принимает вид $Z(x, r) = F(x) + r\sum_{j=1}^{m} p_j(x) \to \min$.

2.5 Правило отбора

Удаление решений-родителей из популяции может привести к потере «хороших» решений. Чтобы этого не произошло можно использовать несколько модификаций классической модели. В одном случае родители не убираются из популяции, а также участвуют в процедуре отбора. В некоторых моделях участие родителей в эволюции ограничивается определенным числом поколений (время жизни). Иногда для элементов популяции устанавливается предельное число участия в операции кроссовера (ограничения селекции).

В ряде работ [5,15,59] предлагались различные модификации классических правил кроссовера и мутации. Однако все эти модификации носят, как правило, искусственный характер и являются индивидуальными для каждого конкретного класса задач. Это, конечно, значительно ограничивает область применимости эволюционных моделей.

3 ЭВОЛЮЦИОННЫЕ МОДЕЛИ G_2 , G_3 И G_4

Кодировать можно не все множество допустимых решений, а лишь некоторую его часть. Естественно, в этом случае поиск оптимума будет вестись именно на этой части множества допустимых решений. Разумно иметь определенные гарантии, что настоящее оптимальное решение не отброшено. Например, при поиске точки максимума многомерной задачи оптимизации можно ограничиться лишь той частью множества допустимых решений, для которой значение целевой функции больше чем ее значение в некоторой фиксированной точке. При поиске оптимума для задачи линейного программирования можно ограничиться лишь множеством базисных допустимых решений задачи.

Для кодирования можно использовать и неоднозначные коды. То есть при кодировании можно отказаться от требования инъективности. Разным кодам может отвечать одно и то же решение. Основное требование к кодированию при построении эволюционной модели – по коду можно вычислить: а) решение, которое за этим кодом скрывается, б) значение целевой функции на этом решении.

Приведем наиболее известные, наряду с бинарными последовательностями, коды, используемые в эволюционных моделях.

3.1 Целочисленное кодирование

При целочисленном кодировании множество кодов является множеством целочисленных векторов Z^n. В этом случае каждое допустимое решение представляется целочисленным вектором $x = (x_1, x_2, ..., x_n)$

Операция кроссовера может быть определена следующим образом: по двум решениям-родителям $x = (x_1, x_2, ..., x_n)$ и $y = (y_1, y_2, ..., y_n)$ строится решение-потомок $z = (z_1, z_2, ..., z_n)$, для координат которого выполняются неравенства

$$\forall k = 1, 2, ..., n \quad \min\{x_k, y_k\} \le z_k \le \max\{x_k, y_k\}$$

Операция мутации состоит в замене одной из координат вектора z произвольным целым числом.

Эволюционную модель на целочисленных векторах с определенными выше правилами кроссовера и мутации будем обозначать G_2.

3.2 Вещественное кодирование

Довольно часто в эволюционном моделировании используется кодирование с помощью последовательностей вещественных чисел [57,61,62,82]. При кодировании вещественными последовательностями множество кодов является эвклидовым пространством R^n.

Оператор кроссовера двум вещественным векторам x и y ставит в соответствие точку на отрезке, соединяющем x,y. Оператор мутации заменяет произвольно выбранную координату (или несколько координат) случайно выбранным вещественным числом.

Эволюционную модель на вещественных векторах в пространстве R^n с определенными выше правилами кроссовера и мутации будем обозначать G_3.

3.3 Эволюционная модель G_4.

Пусть теперь базовое множество модели – множество S_N всех перестановок заданного порядка N. Перестановка s из N элементов представляется словом $i_1 i_2 ... i_N$, в котором каждая буква $i_k \in \{1, 2, ..., N\}$, причем буквы не повторяются.

Определим оператор кроссовера $Kr : S_N \times S_N \to S_N$. Пусть заданы две перестановки $s_1 = i_1 i_2 ... i_N$ и $s_2 = j_1 j_2 ... j_N$. Будем просматривать эти перестановки слева направо и выполнять следующие операции: из первых букв двух слов-перестановок выбирается одна буква (по заданному правилу выбора) и записывается как буква нового слова-перестановки. Эта буква вычеркивается из слов-родителей. Буквы слов-родителей, которые следуют за вычеркнутой буквой,

сдвигаются влево на один символ. Далее процедура выполняется, пока все буквы родительских слов не будут исчерпаны.

Примером подобного оператора кроссовера может служить «минимальный кроссовер». Из двух первых букв перестановок родителей выбирается минимальная по значению. Результат выполнения такого оператора кроссовера представлен на рис.2.

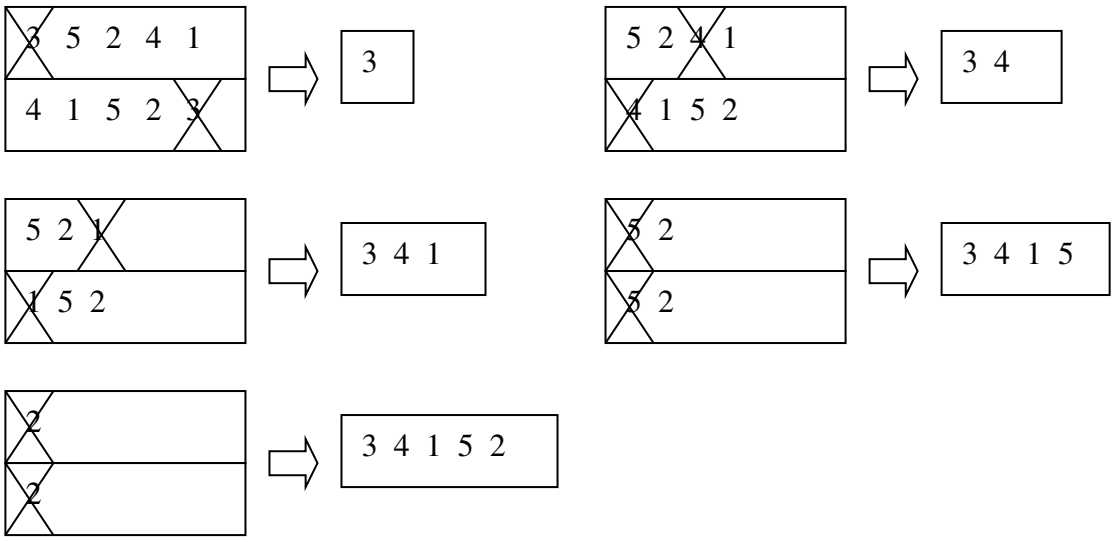


Рис.2 Выполнение оператора кроссовера

Единичная мутация в слове заключается в случайной перестановке двух наугад выбранных букв. Рассмотренную эволюционную модель на перестановках[24] будем обозначать G_4.

Правило кроссовера, которое было описано выше, обладает одним замечательным свойством. Это правило сохраняет отношение порядка между буквами слов-перестановок.

Теорема 3.1. Пусть заданы две перестановки $s_1 = i_1 i_2 ... i_N$ и $s_2 = j_1 j_2 ... j_N$ и пусть в каждой из этих перестановках число i предшествует числу j. Тогда результат кроссовера этих перестановок $Kr(s_1, s_2)$ также будет обладать этим свойством.

Доказательство. Воспользуемся определением операции кроссовера $Kr(s_1, s_2)$ на перестановках. Пусть число i предшествует числу j в перестановках $s_1 = i_1 i_2 ... i_N$ и $s_2 = j_1 j_2 ... j_N$. На каждом шаге операции кроссовера имеет место

следующее свойство. Если не выбрано ни одно из чисел i, j на очередном шаге операции, то в оставшихся после этого шага частях перестановок число i предшествует числу j. Таким образом, число j не может занять первую позицию в оставшихся частях двух перестановок до тех пор, пока число i на очередном шаге операции не перейдет в строку результата. ■

4 НАКОПЛЕНИЕ СВОЙСТВ В ЭВОЛЮЦИОННЫХ МОДЕЛЯХ

Актуальным является вопрос о сходимости эволюционного алгоритма и оценки качества приближенных решений, получаемых с его помощью. Одним из немногих значимых результатов в этой области является классическая теорема схем (шим) [64] для модели Холланда G_1. Но оказывается подобная «сходимость в среднем» присутствует не только в модели Холланда, но и в любой эволюционной модели, в которой кроссовер сохраняет определенные «наследственные» свойства.

4.1 Теорема схем

Определение 4.1. Схемой называется слово вида $s = s_1 s_2 ... s_N$, где $s_i \in \{0,1,*\}$. Каждая схема определяет некоторое подмножество $A_s = \{a_1, a_2, ..., a_N\} \subseteq \{0,1\}^N$,

$$\text{где } a_i = \begin{cases} 0, \text{ если } s_i = 0 \\ 1, \text{ если } s_i = 1 \\ 0 \text{ или } 1, \text{ если } s_i = * \end{cases}.$$

Это подмножество также будем называть схемой. Таким образом, схема – это определенный шаблон для бинарных последовательностей. Если N=6, то примерами схем будут следующие строки: 1**1**, **101*, 1001*0 и так далее. Знаки схемы, отличные от «*» называются значащими. Порядком $o(s)$ схемы $s = s_1 s_2 ... s_N$ называется число знаков схемы, отличных от «*». Определенной длиной $l(s)$ называется расстояние(количество знаков) между крайними значащими знаками схемы.

Обозначим через $S(s,k)$ - множество элементов на k-м шаге, генетического алгоритма, принадлежащих схеме s, а через $|\overline{S}(s,k+1)|$-математическое ожидание(среднее значение) количества элементов, принадлежащих схеме s на шаге k+1. Тогда теорема схем для эволюционной модели G_1 утверждает, что:

$$|\overline{S}(s,k+1)| \geq |S(s,k)| \frac{F_{ср}(s,k)}{F_{ср}(k)} (1 - p_{cross} \frac{l(s)}{N-1})(1 - p_{ms})^{o(s)}, \qquad (4.1)$$

здесь p_{cross} - вероятность случайного выбора элемента для скрещивания, p_{ms} - вероятность того, что отдельный бит слова будет изменен при мутации. $F_{средн}(k)$ – среднее значение критерия отбора на *k*-й популяции, $F_{средн}(s,k)$ – среднее значение критерия отбора на подмножестве элементов *k*-й популяции, принадлежащих схеме *s* и, как обычно, $|X|$- мощность(количество элементов) множества *X*.

Доказательство этой теоремы приводится в многочисленных публикациях [46,64,72].

Теорема схем показывает, что при работе генетического алгоритма происходит накопление элементов тех схем, для которых среднее значение критерия отбора выше среднего значения критерия для всего базового множества. Это в какой-то степени дает теоретическое обоснование сходимости эволюционного алгоритма. Однако вопрос о более точных оценках и гарантиях сходимости остается открытым.

4.2 Наследственность

Постараемся обобщить теорему схем на произвольную эволюционную модель с наследственными свойствами.

Пусть для оптимизационной задачи построена эволюционная модель с базовым множеством *X* и критерием отбора *F*. Причем правило селекции построено на пропорциональном принципе, т.е. вероятность попадания элемента в пару прямо пропорциональна значению критерия для этого элемента (разумеется в качестве критерия используется функция со строго положительными значениями). Пусть *P* – свойство, которым могут обладать (или не обладать) элементы множества *X*. Будем обозначать через X_P – множество элементов из *X*,

которые обладают свойством *P*. Здесь и далее будем рассматривать лишь собственные свойства, то есть такие, для которых $X_P \neq \varnothing$ и $X \setminus X_P \neq \varnothing$.

Определение 4.2. Свойство *P* называется наследственным, если оно сохраняется при кроссовере. Другими словами, если оба элемента-родителя обладают этим свойством, то оно также имеет место у результата кроссовера этих элементов:

$$Kr(X_P \times X_P) \subseteq X_P . \qquad (4.2)$$

Например, свойство «принадлежать определенной шиме» является наследственным для классической эволюционной модели Холланда. Другой важный пример наследственных свойств будет исследован в следующем подразделе.

Пусть для рассматриваемой эволюционной модели применяется эволюционный алгоритм. Обозначим через $|Y_k^P|$- число элементов популяции на *k*-м шаге эволюционного алгоритма, которые обладают свойством *P*, $|\overline{Y}_{k+1}^P|$ - среднее значение числа элементов с указанным свойством на шаге с номером *k*+1. При пропорциональном независимом правиле отбора среднее количество выбранных для кроссовера пар, каждый элемент которых обладает свойством *P*, равно:

$$|Y_k^P| \left(\frac{F_{средн}(P,k)}{F_{средн}(k)} \right) \qquad (4.3)$$

Пусть p_{cross} - вероятность сохранения свойства при операции кроссовера, а p_m - вероятность исчезновения свойства *P* при мутации отдельного элемента, обладающего этим свойством. Тогда справедливо очевидное обобщение теоремы Холланда для произвольного генетического алгоритма, построенного по модели Холланда

Теорема 4.1. (*О накоплении свойств*)

$$|\bar{Y}_{k+1}^{P}| \geq |Y_k^P| \left(\frac{F_{средн}(P,k)}{F_{средн}(k)} \right) p_{cross}(1-p_m). \qquad (4.4)$$

Модифицируем теперь эту теорему для эволюционной модели, которая описана в разделе 1. Отличие от классической состоит в том, что все особи (как существующие, так и возникшие в результате действия оператора кроссовера) переходят в следующее промежуточное поколение, а затем отбираются по правилу отбора.

Теорема 4.2.

$$|\bar{Y}_{k+1}^{P}| \geq (|Y_k^P| + \Delta_{cross}(1-p_m)) \left(\frac{F_{средн}(P,k)}{F_{средн}(k)} \right), \qquad (4.5)$$

где Δ_{cross} - количество особей, со свойством P, которые появились в результате применения оператора кроссовера к выбранным для селекции парам.

Доказательство теоремы вытекает из того факта, что нижняя граница количества элементов промежуточной популяции, обладающих свойством P, определяется соотношением $|Y_k^P| + \Delta_{cross}(1-p_m)$.■

Пусть теперь свойство P является наследственным. Предположим, что элементы в пары для скрещивания из текущей популяции выбираются независимо и с равными вероятностями. Пусть теперь размер популяции ограничен числом N, число пар для скрещивания на каждом этапе равно M. Пары выбираются независимо и элементы в парах могут повторяться. Тогда справедлива *теорема наследственности*:

Теорема 4.3 При принятых выше предположениях имеет место соотношение:

$$|\bar{Y}_{k+1}^{P}| \geq (|Y_k^P| + M\left(\frac{|Y_k^P|}{N}\right)^2 (1-p_m))\left(\frac{F_{средн}(P,k)}{F_{средн}(k)}\right) \qquad (4.6)$$

Доказательство. Действительно, вероятность того, что в наугад выбранной паре элементов из *k*-й популяции окажутся оба элемента, обладающие свойством *P*, равна $\left(\frac{|Y_k^P|}{N}\right)^2$. Таким образом, среднее число элементов, которые будут обладать свойством *P* после *M* скрещиваний и мутаций, равно $M\left(\frac{|Y_k^P|}{N}\right)^2 (1-p_m)$. Учитывая пропорциональное правило селекции и утверждение теоремы 4.2, приходим к требуемому результату. ■

Замечание. В теоремах 4.2, 4.3 предполагается, что правая часть неравенства не превосходит числа *N* элементов популяции.

Благодаря утверждению теоремы 4.3 можно пояснить некоторые особенности эволюционных алгоритмов. В процессе эволюции (работы алгоритма) в текущей популяции накапливаются наследственные свойства, которые в той или иной степени связаны с увеличением значения целевой функции – критерия отбора.

Из теоремы 4.3, в частности, следует, что если $\left(\frac{F_{средн}(P,k)}{F_{средн}(k)}\right) \geq 1$, то количество решений в популяции, обладающих свойством *P* увеличивается экспоненциально. Этот эффект в эволюционных моделях получил название гиперсходимости. Благодаря этому эффекту при практическом применении эволюционных моделей очень быстро может наступить стабилизация популяции, то есть все элементы популяции будут обладать одним значением целевой функции. Поэтому необходимо вносить в модель механизмы практического противодействия гиперсходимости эволюционных алгоритмов, которые в большинстве своем являются индивидуальными для различных задач.

Будем называть наследственное свойство стимулирующим, если для него среднее значение целевой функции на подмножестве всех элементов базового множества, обладающих этим свойством, больше среднего значения целевой функции на всем базовом множестве.

Таким образом, эволюционный алгоритм находит приближенной оптимальное решение задачи в той части базового множества, где сравнительно высокая плотность элементов со стимулирующими свойствами. Конечно, это никоим образом не гарантирует то, что решение будет действительно оптимальным и даже не дает возможности оценить качество этого решения. Однако в какой-то степени обеспечивает «устойчивость» (наличие близких по значению целевой функции решений).

Количество шагов эволюционного алгоритма, вообще говоря, не ограничено. Для того чтобы при практических реализациях эволюционный алгоритм всегда останавливался, в условие остановки алгоритма включают ограничения по числу шагов или по времени поиска решения.

Пусть $Y_0 \subseteq X$ начальная популяция в эволюционной модели. Будем называть *k*-м поколением множество $Y_k = Y_{k-1} \cup Kr(Y_{k-1}, Y_{k-1})$.

Определение 4.3. Достижимым множеством для заданной начальной популяции $Y_0 \subseteq X$ будем называть объединение всех поколений $D(Y_0) = \bigcup_k Y_k$. Начальную популяцию Y_0 будем называть достаточной, если $D(Y_0) = X$.

Оператор кроссовера будем называть исчерпывающим, если для этого оператора любая непустая начальная популяция является достаточной.

Интересно, что свойство «быть исчерпывающим» для оператора кроссовера несовместимо со свойством наследственности. А именно: справедлива следующая

Теорема 4.4 Любое собственное свойство *P*, для исчерпывающего оператора кроссовера не является наследственным.

Доказательство. Предположим, что существует некоторое собственное свойство *P*, которое является наследственным для исчерпывающего оператора кроссовера Kr. Обозначим через $X_P \subseteq X$ собственное подмножество базового

множества, которое состоит из всех элементов, обладающих свойством P. Выберем в качестве начальной популяции любое непустое подмножество $Y_0 \subseteq X_P$. Тогда в силу наследственности свойства $D(Y_0) \subseteq X_P \neq X$. Соответственно, оператор кроссовера не является исчерпывающим и, следовательно, предположение неверно. ■

Подведем краткие итоги. С одной стороны, наследственность оператора кроссовера приводит к тому, что в рамках эволюционной модели происходит накопление «полезных» свойств и концентрация решений с достаточно «хорошим» значением целевой функции. Причем эта концентрация происходит достаточно быстро, то есть имеет место эффект «гиперсходимости» эволюционного алгоритма. С другой стороны, если кроссовер обладает свойствами наследственности, то результат оптимизации с помощью эволюционного алгоритма существенно зависит от выбора начальной популяции решений. Кроме того, понятна роль мутации, благодаря которой удается выйти за пределы достижимого множества. Именно мутация позволяет применять эволюционные модели для поиска глобальных (а не только локальных) оптимумов задачи.

Отметим еще раз, что применение эволюционных моделей оправдано лишь для тех оптимизационных задач, для которых на сегодняшний день не известны эффективные точные алгоритмы поиска решения или приближенные алгоритмы с гарантированной сходимостью. Некорректно проводить сравнение эволюционных алгоритмов с точными эффективными алгоритмами. Разумно сравнивать походы, основанные на эволюционном моделировании, с другими метаэвристиками.

5 ГЕОМЕТРИЯ И НАСЛЕДСТВЕННОСТЬ

Возникает вопрос, существуют ли априорные условия, которые гарантируют наличие свойств наследственности в эволюционных моделях. Оказывается, эти свойства тесно связаны с геометрией базового пространства.

5.1 Структуры с наследственностью

Определение 5.1. Структурой с наследственностью будем называть пару (X,Θ), где X – множество, а Θ- функция, которая каждой паре точек $x, y \in X$, ставит в соответствие подмножество $\Theta_{xy} \subseteq X$, причем:

$$
\begin{aligned}
&1)\ \forall x, y \in X,\ x \in \Theta_{xy}, y \in \Theta_{xy} \\
&2)\ \forall x, y \in X,\ \Theta_{xy} = \Theta_{yx} \\
&3)\ \forall x, y \in X,\ \forall u \in \Theta_{xy},\ \Theta_{xy} \supseteq \Theta_{xu} \cup \Theta_{uy}
\end{aligned}
\qquad (5.1)
$$

Множество Θ_{xy} называется множеством потомков точек $x, y \in X$. Таким образом, структура с наследственностью определяется совокупностью множеств потомков $\{\Theta_{xy}\}_{x,y\in X}$ со свойствами (5.1).

Приведем теперь некоторые свойства структур с наследственностью.

Теорема 5.1. Пусть задана структура с наследственностью (X,Θ). Тогда для любого точки $x \in X$ и любой точки $y \in \Theta_{xx}$ справедливо равенство $\Theta_{xx} = \Theta_{xy}$.

Доказательство. Пусть $y \in \Theta_{xx}$. Тогда, в соответствии с определением (5.1), получаем, что и $\Theta_{xx} \supseteq \Theta_{xy} \cup \Theta_{yx}$. С другой стороны, в соответствии с тем же определением, $\Theta_{xy} \supseteq \Theta_{xx} \cup \Theta_{xy}$. Следовательно, $\Theta_{xx} = \Theta_{xy}$ ■

В частности, $\forall y \in \Theta_{xx}$ выполняется равенство $\Theta_{yy} = \Theta_{xx}$.

Теорема 5.2. Пусть задана структура с наследственностью (X,Θ). Тогда для любого множества потомков Θ_{xy} и для любой точки $z\in\Theta_{xy}$ из условия $y\in\Theta_{xz}$ следует, что $\Theta_{xz}=\Theta_{xy}$.

Доказательство. Пусть $z\in\Theta_{xy}$. Тогда, в соответствии с определением (5.1), получаем, что и $\Theta_{xy}\supseteq\Theta_{xz}\cup\Theta_{zy}$ и, соответственно, $\Theta_{xz}\subseteq\Theta_{xy}$. Если точка $y\in\Theta_{xz}$, то $\Theta_{xy}\subseteq\Theta_{xz}$. Следовательно, $\Theta_{xz}=\Theta_{xy}$ ■

Заметим, что наличие структуры с наследственностью на множестве *X* позволяет определить на этом множестве структуры выпуклости [47].

Выпуклым подмножеством называется подмножество в *X*, которое вместе с любыми двумя своими точками содержит множество потомков этих точек. Пустое множество по определению считается выпуклым. Очевидно следующее утверждение: пересечение любого числа выпуклых подмножеств выпукло.

Определение 5.2. Пусть задана структура с наследственностью. Выпуклой оболочкой подмножества $Y\subseteq X$ называется пересечение всех выпуклых подмножеств множества *X*, содержащих подмножество *Y*.

Другими словами – выпуклая оболочка подмножества $Y\subseteq X$ это наименьшее по включению выпуклое подмножество в *X*, содержащее *Y*.

Один из тривиальных примеров структуры с наследственностью - это разбиение множества. Пусть множество *X* разбито на *k* подмножеств $X_1, X_2, ..., X_k$, причем

1) $\bigcup_{i=1}^{k} X_k = X$;

2) $\forall i, j,\ \ i\neq j\ \ X_i\cap X_j=\varnothing$;

3) $\forall i\ \ X_i\neq\varnothing$.

Каждый элемент $x\in X$ принадлежит ровно одному элементу разбиения, который будем обозначать X_x. Множество всевозможных объединений элементов разбиения образует структуру с наследственностью. Причем

$\forall x, y \in X \quad [x, y] = X_x \cup X_y$, где X_x - это элемент разбиения, который содержит элемент *x*.

Рассмотрим эволюционную модель, для которой базовое множество обладает структурой с наследственностью. Оператор кроссовер $Kr : X \times X \to X$ будем называть наследственным или выпуклым, если $\forall x, y \in X \quad Kr(x, y) \in \Theta_{xy}$. Для такого оператора кроссовера наследственными являются свойства «принадлежать какому-либо множеству выпуклой оболочки потомков».

Если кроссовер в эволюционной модели наследственный, то в этой модели присутствует эффект накопления свойств, который описан в предыдущем разделе.

5.2 Структуры с наследственностью и метрика

В метрическом пространстве $R^n = \{(x_1, x_2, ..., x_n)\}$ с канонической эвклидовой метрикой естественным образом определяется структура с наследственностью. А именно: множествами потомков являются обычные (геометрические) отрезки в этом пространстве. Оказывается, структуру с наследственностью можно определить на любом метрическом пространстве.

Напомним, что метрическим пространством называется пара (X, ρ), где X - множество, $\rho : X \times X \to R^1$ - функция, со следующими свойствами:

$$\begin{aligned} &1) \quad \forall x, y \in X, \ \rho(x, y) \geq 0; \ \rho(x, y) = 0 \Leftrightarrow x = y \\ &2) \quad \forall x, y \in X, \ \rho(x, y) = \rho(y, x) \\ &3) \quad \forall x, y, z \in X, \ \rho(x, y) \leq \rho(x, z) + \rho(z, y) \end{aligned} \qquad (5.2)$$

Метрическим отрезком (в дальнейшем отрезком), соединяющим две точки *x* и *y* метрического пространства *X*, будем называть множество $[x, y] = \{z \in X \mid \rho(x, y) = \rho(x, z) + \rho(z, y)\}$.

Теорема 5.3. Множество отрезков метрического пространства (X, ρ) образуют структуру с наследственностью.

Доказательство. Свойства (1) и (2) определения 5.1 для метрических отрезков с очевидностью вытекают из определения метрики. Докажем теперь свойство (3) структур с наследственностью. Пусть *x,y* – две произвольные точки пространства *X*. Возьмем произвольную точку $z \in [x, y]$. Покажем, что $[x, y] \supseteq [x, z] \cup [z, y]$. Пусть $u \in [x, z]$. В соответствии с определением отрезка $\rho(x, z) = \rho(x,u) + \rho(u, z)$. По определению расстояния $\rho(x, y) \le \rho(x,u) + \rho(u, y)$. С другой стороны, имеем соотношение $\rho(x, y) = \rho(x, z) + \rho(z, y) = \rho(x,u) + \rho(u, z) + \rho(z, y) \ge \rho(x,u) + \rho(u, y)$. Таким образом, $\rho(x, y) = \rho(x,u) + \rho(u, y)$ и, следовательно, $[x, z] \subseteq [x, y]$. Аналогично можно показать, что $[z, y] \subseteq [x, y]$ и потому $[x, z] \cup [z, y] \subseteq [x, y]$. ■

Таким образом, на метрическом пространстве всегда определена структура с наследственностью, состоящая из всего множества отрезков.

Рассмотрим примеры метрических пространств и отрезков в этих пространствах.

5.3 Эвклидово пространство с канонической метрикой

В векторном пространстве $R^n = \{(x_1, x_2, ..., x_n)\}$ стандартная(эвклидова) метрика задается следующим равенством:

$$\forall x = (x_1, x_2, ..., x_n), y = (y_1, y_2, ..., y_n) \in R^n$$

$$\rho_{эвкл}(x, y) = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + ... + (y_n - x_n)^n}$$

В этой метрике отрезок между точками выглядит как обычный отрезок прямой (рис.3.1).

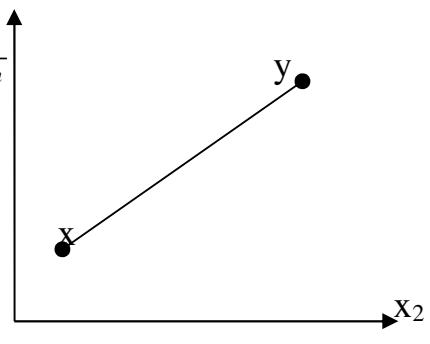


Рис.3.1 Отрезок между двумя точками в R^n

5.4 Пространство с манхэттенской метрикой

Приведем другой пример метрики в векторном пространстве R^n. Манхэттенская метрика(метрика городских кварталов) определяет расстояние между точками $x=(x_1,x_2,...,x_n)$ и $y=(y_1,y_2,...,y_n)$ формулой

$$\rho_{manh}(x,y)=|y_1-x_1|+|y_2-x_2|+...+|y_n-x_n|$$

Пример отрезка [*x*,*y*] в этой метрике изображен на рис.4. Интересно, что выпуклыми множествами в такой метрике являются прямоугольные параллелепипеды различных размерностей и только они.

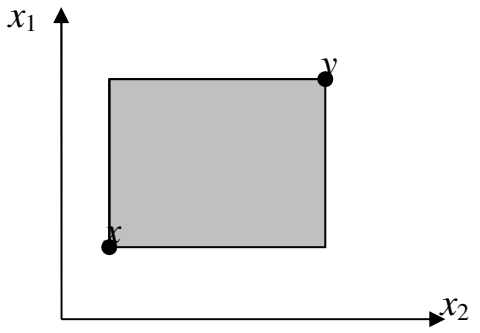


Рис.4 Отрезок между двумя точками в R^n в манхеттенской метрике

5.5 Бинарное n-мерное пространство и метрика Хэмминга

Рассмотрим теперь бинарное *n*-мерное пространство $B^n=\{(x_1,x_2,...,x_n)\}$, где $x_i\in\{0,1\}\ \ \forall i=1,2,...,n$. Стандартная метрика в этом пространстве – метрика Хэмминга. Расстояние $\rho_{Hem}(x,y)$ между точками $x,y\in B^n$ определяется, как количество позиций, в которых различаются координаты этих точек. Например, расстояние Хэмминга между точками (1,0,1,1,0,1) и (0,1,1,0,1,0) в шестимерном бинарном пространстве равно 5. Отрезок между двумя точками бинарного пространства – определяет схему в смысле Холланда, которая уже упоминалась выше. Например, для точек (1,0,1,1,0,1) и (0,1,1,0,1,0) отрезок между ними задается схемой (*,*,1,*,*,*). В бинарном пространстве выпуклыми множествами являются отрезки и только они.

5.6 Пространство перестановок

Следующий интересный пример метрического пространства это пространство перестановок S_n. Каждый элемент этого пространства описывается перестановкой $s = (s_1, s_2, ..., s_n)$, где все координаты попарно различные числа из множества $\{1,2,\ldots,n\}$.

Расстояние между точками пространства S_n может быть определено различными способами. Приведем некоторые из них [10].

1) расстояние Хэмминга - расстояние между перестановками определяется как количество позиций перестановок, в которых элементы перестановок различаются;

2) метрика Кэли – минимальное количество транспозиций элементов, которые необходимо выполнить, чтобы перевести одну перестановку в другую;

3) метрика Кэндалла – минимальное количество транспозиций соседних элементов, которые необходимо выполнить, чтобы перевести одну перестановку в другую.

Пусть $s = (s_1, s_2, ..., s_n) \in S_n$. Запись $i <_s j$ будет означать, что в перестановке s элемент i предшествует элементу j. Определим порядковую матрицу (a_{ij}^s) перестановки s следующим образом:

$$a_{ij}^s = \begin{cases} 1 & \text{если } i <_s j \\ -1 & \text{если } \mathrm{j} <_s i \\ 0 & \text{если } i = j \end{cases}.$$

Теорема 5.4 Пусть $u = u_1 u_2 ... u_n$ и $v = v_1 v_2 ... v_n$. Расстояние Кэндалла между перестановками u и v определяется формулой

$$\rho_{Kend}(u,v) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} | a_{ij}^v - a_{ij}^u |. \qquad (5.3)$$

Доказательство. Пусть $u \neq v$. Тогда в перестановке v найдутся два соседних элемента i,j такие, что

$$i <_u j \text{ и } j <_v j. \qquad (5.4)$$

Действительно, пусть таких элементов в перестановке v нет. Выберем в перестановке v наиболее близкие элементы i,j обладающие указанным свойством. Выберем элемент k между элементами j и i в перестановке v. Тогда имеем $i <_u j$, $j <_v k <_v i$. Но это означает, что в перестановке u либо $k <_u i$, либо $j <_u k$. В любом случае i,j не являются ближайшими, обладающими свойством (5.4), в перестановке v. Следовательно, обязательно найдутся соседние элементы i,j в перестановке v, обладающие свойством (5.4). Последовательно меняя местами в перестановке v соседние элементы, обладающие свойством (5.4), можно перейти к перестановке u за минимальное число шагов, равное $\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}|a_{ij}^{v} - a_{ij}^{u}|$. Это и будет расстояние Кэндалла между перестановками. ■

Попытаемся теперь описать отрезок межу двумя перестановками в метрике Кэндалла.

Теорема 5.5. В метрике Кэндалла отрезком, соединяющим перестановки, является множество всех перестановок, сохраняющих относительный порядок между элементами в u, и v, а именно: $[u,v] = \{w \in S_n : i <_u j \text{ и } i <_v j \Rightarrow i <_w j\}$

Доказательство. Обозначим через $W = \{w \in S_n : i <_u j \text{ и } i <_v j \Rightarrow i <_w j\}$. Тогда, в соответствии с (5.3), для всякой перестановки $w \in W$ имеет место равенство $\rho(u,v) = \rho(u,w) + \rho(w,v)$ и потому $W \subseteq [a,b]$. Пусть перестановка w принадлежит отрезку $[u,v]$, соединяющему перестановки $u = u_1 u_2 ... u_n$ и $v = v_1 v_2 ... v_n$. Тогда если $i <_u j$ и $i <_v j$, то этот же порядок элементов i,j присутствует и в перестановке w. В противном случае удалось бы уменьшить число транспозиций соседних элементов, позволяющих перейти от u к v. Следовательно, $[a,b] \subseteq W$ и теорема доказана. ■

Можно показать, что любое выпуклое множество в метрике Кэндалла является отрезком. Отметим одно интересное свойство: шар в этой метрике, т.е. множество элементов вида $D(u_0, r) = \{u \in S_n \mid \rho(u_0, u) \le r\}$, вообще говоря, не является выпуклым множеством [27]

5.7 Метрика прямого произведения пространств

Пусть заданы два метрических пространства (X_1, ρ_1) и (X_2, ρ_2). Напомним, что прямым произведением множеств X_1 и X_2 называется множество пар $X_1 \times X_2 = \{(x_1, x_2) \mid x_1 \in X_1, x_2 \in X_2\}$. На множестве $X_1 \times X_2$ естественным образом определеляется метрика ρ_{12}. А именно: для любых двух точек $(x_1, x_2), (y_1, y_2) \in X_1 \times X_2$ расстояние между ними задается формулой

$$\rho_{12}((x_1, x_2), (y_1, y_2)) = \rho_1(x_1, y_1) + \rho_2(x_2, y_2). \qquad (5.5)$$

Теорема 5.6. В метрике прямого произведения отрезок между любыми двумя точками $(x_1, x_2), (y_1, y_2) \in X_1 \times X_2$ содержит как подмножество прямое произведение отрезков $[x_1, y_1] \times [x_2, y_2]$ в соответствующих метриках пространств X_1 и X_2.

Доказательство. Действительно выберем произвольную точку $(z_1, z_2) \in [x_1, y_1] \times [x_2, y_2]$. В соответствии с определением отрезка имеем цепочку равенств:

$$\rho_{12}((x_1, x_2), (y_1, y_2)) = \rho_1(x_1, y_1) + \rho_2(x_2, y_2) =$$
$$\rho_1(x_1, z_1) + \rho_1(z_1, y_1) + \rho_2(x_2, z_2) + \rho_2(z_2, y_2) =$$
$$\rho_{12}((x_1, x_2), (z_1, z_2)) + \rho_{12}((z_1, z_2), (y_1, y_2))$$

Следовательно, $[x_1, y_1] \times [x_2, y_2] \subseteq [(x_1, x_2), (y_1, y_2)]$.■

Следует заметить, что рассмотренные выше определение и теорема легко могут быть обобщены на случай прямого произведения любого конечного числа метрических пространств.

5.8 Геометрический кроссовер

Пусть базовое множество X эволюционной модели является метрическим пространством с метрикой $\rho : X \times X \to R^1_+$. Как и выше, обозначим оператор кроссовера $Kr : X \times X \to X$.

Определение 5.3. Оператор кроссовера на метрическом пространстве (X, ρ) называется геометрическим, если для любых $x, y \in X$ результат операции кроссовера $Kr(x, y)$ принадлежит отрезку [x,y], соединяющему точки x и y. Другими словами, $\forall x, y \in X \quad \rho(x, y) = \rho(x, Kr(x, y)) + \rho(Kr(x, y), y)$.

Естественно, всякий геометрический кроссовер является выпуклым.

Напомним, что на метрическом пространстве определена структура с наследственностью, которая состоит из множества всех отрезков пространства. Таким образом, для геометрического кроссовера имеют место свойства наследования. Наследуется принадлежность элементов одному отрезку.

Заметим, что в каждой из моделей G_1-G_4, которые рассматривались выше, оператор кроссовера является геометрическим. Таким образом, во всех этих моделях присутствуют наследственные свойства – сохранение принадлежности отрезку в соответствующем метрическом пространстве.

В статье [73] доказан ряд необходимых условий геометричности кроссовера. Приведем их с небольшими изменениями в виде теорем.

Теорема 5.7. (Свойство чистоты) Для того, чтобы кроссовер $Kr : X \times X \to X$ был геометрическим необходимо, чтобы $\forall x \in X \quad Kr(x, x) = x$.

Теорема 5.8. (Свойство конвергенции) Для того, чтобы кроссовер $Kr : X \times X \to X$ был геометрическим необходимо, чтобы $\forall x, y, z \in X$ и $z = Kr(Kr(x, y), y)$ из условия $z = x$ следует, что $Kr(x, y) = x$.

Теорема 5.9. (Свойство делимости) Для того, чтобы кроссовер $Kr : X \times X \to X$ был геометрическим необходимо, чтобы из условий $z \in [x, y]$ и $w \in [x, z] \cap [z, y]$ следовало, что $z = x$ или $z = y$.

Великолепным исследованием, посвященным эволюционным моделям с геометрическим оператором кроссовера является работа [75].

5.9 Полные системы наследственных свойств

С каждым свойством P на базовом множестве X эволюционной модели можно связать его характеристическую функцию

$$\chi(x)=\begin{cases}1, & \text{если элемент } x \text{ обладает свойством } P\\ 0, & \text{если элемент } x \text{ не обладает свойством } P\end{cases}.$$

Определение 5.3. Система свойств $P_1,P_2,...,P_s$ называется полной системой свойств, если характеристические функции этих свойств попарно различны, то есть

$$\forall x,y\in X \quad \exists k\in\{1,2,...,s\} \quad \chi_k(x)\neq\chi_k(y),$$

где $\chi_k(x)$ -характеристическая функция свойства P_k, k=1,2,…,s. Заметим, что в соответствии с определением, полные системы наследственных свойств могут существовать лишь для тех моделей, у которых базовые множества конечны. Однако это понятие легко обобщается и на случай бесконечных базовых множеств.

Теорема 5.10 Для того чтобы существовала полная система наследственных свойств $P_1,P_2,...,P_s$ на конечном базовом множестве X с оператором кроссовера $Kr: X\times X\to X$, необходимо и достаточно, чтобы на базовом множестве X существовала метрика, в которой оператор кроссовера Kr был бы геометрическим.

Доказательство. Необходимость. Пусть $P_1,P_2,...,P_s$ - полная система наследственных свойств на множестве X. Определим метрику

$$\rho(x,y)=\sum_{k=1}^{s}|\chi_k(y)-\chi_k(x)|, \qquad (5.6)$$

где $\chi_k(x)$ - характеристическая функция свойства P_k $k=1,2,...,s$. Свойства метрики (5.2) проверяются непосредственно.

Для произвольных элементов $x,y\in X$ обозначим $z=Kr(x,y)$. Так как кроссовер является наследственным по отношению к свойствам P_k $k=1,2,...,s$, то $\chi_k(z)=\chi_k(x)$, при $\chi_k(x)=\chi_k(y)$ и $|\chi_k(x)-\chi_k(y)|=|\chi_k(x)-\chi_k(z)|+|\chi_k(y)-\chi_k(z)|$,

в том случае, когда $\chi_k(x) \neq \chi_k(y)$. Таким образом, $\rho(x,y) = \rho(x,z) + \rho(z,y)$ и, следовательно, кроссовер является геометрическим в метрике (5.6).

Достаточность. Для каждой пары точек $a,b \in X$ определим свойство P_{ab} - принадлежит выпуклой оболочке отрезка, соединяющему точки a и b. Эта система свойств является полной. Действительно, если $x \neq y$, то элемент x обладает свойством P_{xx}, а элемент y свойством P_{xx} не обладает.

Покажем, что свойство P_{ab} является наследственным для геометрического кроссовера. Пусть два решения $x, y \in X$ обладают свойством P_{ab}. Обозначим $z = Kr(x,y)$. В силу геометричности кроссовера, точка z принадлежит отрезку $[x,y]$ и, следовательно, точка z обладает свойством P_{ab}. Таким образом, любое из свойств P_{ab} является наследственным ■

Приведем примеры полных систем наследственных свойств для различных эволюционных моделей G_1 –G_4.

В модели G_1 свойство принадлежность различным схемам образует полную систему наследственных свойств.

В модели G_2 (G_3) полную систему наследственных свойств образуют свойства вида «точка x принадлежит выпуклой оболочке точек a,b» для произвольных точек $a,b \in X$.

В модели G_4 полную систему наследственных свойств образуют отношения предшествования вида «i предшествует j» в перестановке.

6 КАК ПОСТРОИТЬ ЭВОЛЮЦИОННУЮ МОДЕЛЬ

Теперь можно сформулировать общие принципы построения эволюционных моделей для поиска оптимальных решений оптимизационных задач. Влияние составляющих модели на эффективность эволюционного алгоритма рассматривалось в многочисленных работах [2,4,5,46,72]. Поэтому приведем здесь лишь основные общие правила построения эволюционных моделей, уделив особое внимание условиям наследования.

6.1 Базовое пространство

Первым этапом построения эволюционной модели является, безусловно, выбор базового пространства модели. Базовое пространство по возможности наиболее полно должно отвечать естественному описанию модели. В частности, должны быть определены функции кодирования и декодирования, которые переводят элементы моделируемых структур в элементы базового пространства и, наоборот, позволяют построить по элементу базового пространства решение исходной задачи.

6.2 Критерий

На целевую функцию(критерий отбора) не накладывается никаких ограничений, кроме принципиальной вычислимости. То есть по коду решения может быть легко найдено значение целевой функции. Однако иногда целевая функция может быть изменена. В задачах условной максимизации целевая функция доопределяется на всем базовом пространстве следующим образом. Если для рассматриваемого решения не выполняются условия задачи, то значение целевой функции – большое по модулю отрицательное число (-M для задач на максимум). Еще один случай, когда нужно видоизменить критерий – это случай наличия многих экстремумов с одинаковым значением целевой функции. В этом случае эволюционный алгоритм начинает работать неустойчиво, перепрыгивая от окрестности одного локального оптимума к другому. Рекомендуется дополнить

задачу еще одним критерием или группой критериев, для того чтобы исключить многоэкстремальность. Затем ведется поиск лексикографического оптимума. В измененной задаче главным критерием остается исходная целевая функция, а все остальные критерии относятся ко второстепенным..

6.3 Оператор кроссовера

На следующем шаге необходимо выбрать метрику в базовом пространстве. Наличие метрики позволит запустить механизм наследственности. Метрика определяет наследственную структуру в базовом пространстве, что в дальнейшем обеспечит эффект накопления «полезных» свойств при работе эволюционного алгоритма.

Выбор метрики на базовом пространстве и условие выпуклости налагает определенные требования на оператор кроссовера. Оператор кроссовера должен быть геометрическим в этой метрике. Именно это свойство кроссовера обеспечивает наследственность и позволяет в какой-то степени обосновать работу эволюционного алгоритма. Подход, основанный на геометрических свойствах кроссовера, часто применяется в гибридных алгоритмах поиска оптимальных решений [7,8].

6.4 Начальная популяция

От выбора начальной популяции существенно зависит результат, который будет получен в рамках эволюционной модели. Будем называть достижимым элементом в эволюционной модели тот элемент, который может появиться на каком-либо шаге эволюции.

Пусть кроссовер в рассматриваемой модели геометрический и, следовательно, выпуклый. Тогда, в силу определения выпуклого оператора кроссовера, справедливо следующее утверждение:

Теорема 6.1. В рамках эволюционной модели без оператора мутации любой достижимый элемент принадлежат выпуклой оболочке начальной популяции. Более того, поколение, которое будет получено на шаге с номером k+1, является

подмножеством выпуклой оболочки поколения, полученного на предыдущем *k*-м шаге эволюционного алгоритма.

Теорема 6.1 показывает ограниченность эволюционных моделей без оператора мутации. В частности, все решения, полученные путем применения эволюционного алгоритма без оператора мутации, всегда принадлежат выпуклой оболочке начальной популяции.

6.5 Мутация

Теперь понятно назначение мутации в эволюционных моделях. Задача операции мутации получить решение, которое выходит за границы выпуклой оболочки текущей популяции и тем самым обеспечить возможность увеличения пространства поиска и, в частности, возможность достижения именно глобального оптимума или близких к нему решений. Без операции мутации эволюционный алгоритм становится локальным. Соответствующим должен быть и выбор мутации. Хорошие результаты следует ожидать от гибридных алгоритмов, в которых оптимальное решение, полученное на текущем шаге эволюционного алгоритма, улучшается путем применения локального алгоритма оптимизации. Причем выбор базы окрестностей для локального поиска должен быть таким, чтобы обеспечить выход поиска за границы выпуклой оболочки текущей популяции.

6.6 Селекция

Оператор селекции позволяет выбрать отдельные решения для последующего скрещивания. При этом желательно, чтобы свойства, отвечающие за близость к оптимуму, перешли в следующее поколение. Поэтому разумно выбирать для скрещивания решения с «лучшим» значением целевой функции $F(x)$. Таким свойством обладает, например, пропорциональный оператор селекции, когда вероятность выбора пары для скрещивания пропорциональна значению целевой функции на этом решении (для задач на максимум при положительных значениях целевой функции рис.5а).

Иногда, для ослабления эффекта гиперсходимости можно сместить центр распределения (рис.5б).

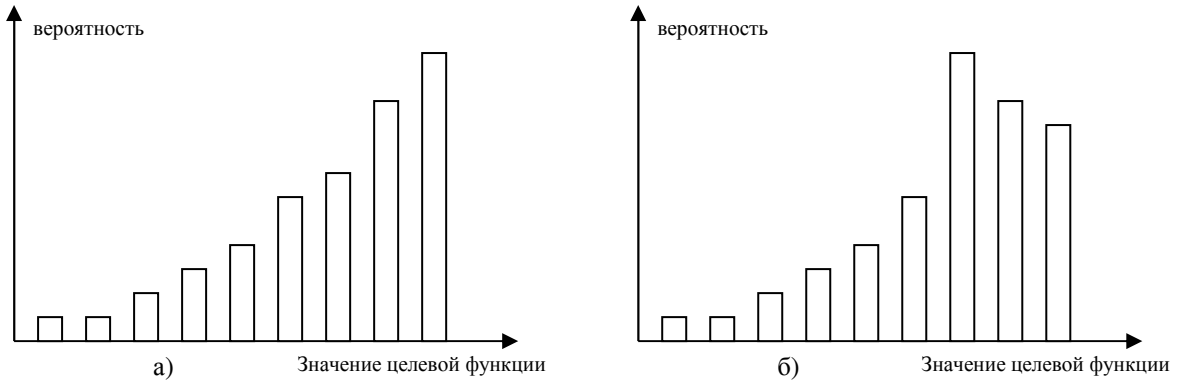


Рис. 5. Возможные виды распределений для оператора отбора

При пропорциональном выборе вероятность выбора решения x_i для выполнения операции скрещивания равна $p_i = \frac{F(x_i)}{\sum_{k=1}^{N} F(x_k)}$ любых двух точек. Другой вариант селекции состоит в порядковом выборе. В этом случае все элементы популяции упорядочиваются по убыванию значения целевой функции $F(x_1) \geq F(x_2) \geq \ldots \geq F(x_N) > 0$ (для задачи на максимум функции) и вероятность выбора определяется позицией элемента в этом ряду: $p_i = \frac{1}{N}\left(a - (2a-2)\frac{i-1}{N-1}\right)$, где число a выбирается из диапазона $1 \leq a \leq 2$. Возможны и другие методы селекции [46,72].

6.7 Отбор

Оператор отбора выбирает решения из промежуточной популяции, которые перейдут в следующее поколение. Наиболее простой способ выбора это отсев решений с «плохим» значением целевой функции. Однако для преодоления эффекта гиперсходимости можно применять и другие методы отбора. Например, можно ввести параметр «время жизни» для отдельных решений – число поколений в которых может присутствовать решение. Решения, которые достигли

предельного уровня времени жизни удаляются из популяции. Для отсева можно использовать параметр «предельное число скрещиваний». В этом случае из популяции удаляются элементы, для которых достигло предела число их участий в операции кроссовера.

6.8 Условие остановки

Наиболее типичными условиями остановки являются: а) превышение предельного числа поколений в эволюции; б) превышение предельного времени работы алгоритма; в) стабилизации популяции по значению целевой функции (все решения имеют одинаковое значение целевой функции); г) достижение необходимого уровня значений целевой функции на «лучших» решениях популяции.

6.9 Многокритериальная оптимизация

При построении эволюционной модели для многокритериальной оптимизационной задачи немного изменяются механизмы селекции и отбора. Это связано с тем, что для применения оператора селекции и оператора отбора элементы популяции необходимо упорядочить. В случае одного критерия элементы популяции упорядочиваются по значению критерия. В случае нескольких критериев такой метод упорядочения уже не годится.

Пусть в базовом пространстве $X = \{x\}$ задана векторная целевая функция $F : X \to R^k$, причем задача оптимизация ставится следующим образом

$$F(x) = (F_1(x), F_2(x), ..., F_k(x)) \to \max .$$

Для простоты будем полагать множество X конечным. Напомним, что Парето оптимальным элементом для этой задачи называется такой элемент $x^* \in X$, для которого не существует такого элемента $x \in X$, что $F_1(x^*) \le F_1(x), F_2(x^*) \le F_2(x), ..., F_k(x^*) \le F_k(x)$ и $F(x^*) \ne F(x)$. Множество всех Парето оптимальных элементов называется множеством Парето рассматриваемой задачи. Отношение $\prec$ полного линейного порядка на X называется эффективным,

если $\forall x, y \in X$ из условия $F_1(x) \le F_1(y), F_2(x) \le F_2(y), ..., F_k(x) \le F_k(y)$ следует, что $x \prec y$. Наряду с векторным критерием, рассмотрим какой-либо эффективный линейный порядок в базовом пространстве.

Для выполнения оператора селекции(выбора пар для скрещивания) элементы текущей популяции будем упорядочивать по убыванию в соответствии с отношением порядка $\prec$.

Введем теперь вспомогательную функцию $F_0^k(x)$ на объединении множества элементов текущей популяции Y_k и множества потомков C'_k. Значение функции $F_0^k(x)$ на элементе x будем полагать равным 1, если *x* является максимальным элементом для функции $F(x)$ на множестве $Y_k \cup C'_k$ и нулю в противном случае. Для выполнения оператора отбора поступим следующим образом: упорядочим элементы множества $Y_k \cup C'_k$ лексикографически, по убыванию значения $F_0^k(x)$ и по убыванию относительно выбранного порядка $\prec$. Пусть размер популяции равен *N*. В очередное поколение Y_{k+1} переходят первые *N* элементов промежуточной популяции. На последнем шаге первые элементы такого упорядочения промежуточной популяции со значением функции $F_0^k(x) = 1$ можно рассматривать как приближение множества Парето исходной задачи (точнее как коды элементов множества Парето).

В следующих главах рассмотренные выше принципы будут использованы для построения универсальной эволюционной модель для широкого класса задач дискретной оптимизации. Это класс задач является расширением класса задач дискретной оптимизации, для которых возможно применение «жадного» алгоритма.

7 ЭВОЛЮЦИОННЫЕ МОДЕЛИ С ГЕОМЕТРИЧЕСКИМ ОПЕРАТОРОМ КРОССОВЕРА

В этой главе будет рассмотрено несколько эволюционных моделей, которые основаны на геометрических операторах кроссовера в различных базовых пространствах с различными метриками. Подобные модели могут применяться для широкого круга задач, как дискретных, так и непрерывных. Напомним, что как было показано выше, геометричность кроссовера гарантирует наличие наследственных свойств и, соответственно, разбиение базового пространства на классы множеств с различными наборами свойств. А далее в рамках модели, в соответствии с теоремой о накоплении свойств, будет происходить накопление «полезных» свойств в текущей популяции. Это позволяет надеяться на то, что приближенное решение, найденное эволюционным алгоритмом, будет достаточно близко к оптимальному (по значению целевой функции).

7.1 Задача оптимизации в n- мерном эвклидовом пространстве

Рассмотрим общую задачу оптимизации в многомерном эвклидовом пространстве R^n. Будем полагать, что речь идет о безусловном экстремуме функции $F: R^n \to R^1$ в кубической области достаточно большого объема. То есть,

$$\begin{array}{l} F(x_1, x_2, ..., x_n) \to \max \\ x_i \in [0, M], \quad i = 1, 2, ..., n \end{array}. \qquad (7.1)$$

Для построения эволюционной модели в качестве базового пространства выберем множество $X = [0, M]^n$. Оператор начальной популяции случайным образом выбирает конечное множество точек во множестве *X*.

Оператор кроссовера каждой паре точек $x = (x_1, x_2, ..., x_n)$, $y = (y_1, y_2, ..., y_n)$ ставит в соответствие точку $z = (z_1, z_2, ..., z_n)$, где $z_i = x_i + (y_i - x_i)t, \quad i = 1, 2, ..., n$ и t – случайное число из промежутка [0,1].

Мутация меняет случайным образом координату z_i точки z на случайно выбранное вещественное число из промежутка $[0, M]$.

Оператор отбора – пропорциональный или любой другой из описанных в предыдущих разделах. Оператор селекции, условие остановки и другие параметры эволюционной модели могут быть любыми и подбираются уже для конкретной задачи.

7.2 Задача целочисленного линейного программирования

Задачу целочисленного линейного программирования рассмотрим в следующей постановке:

$$\begin{aligned}
&c_1x_1 + c_2x_2 + ... + c_nx_n \to \max \\
&a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \le b_1 \\
&a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \le b_2 \\
&..... \\
&a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \le b_m \\
&0 \le x_i \le m_i \\
&x_i \in Z, i = 1,2,...,n
\end{aligned} \qquad (7.2)$$

Здесь коэффициенты $c_i, a_{ji}, b_j, m_i \quad i = 1,2,...,n; \quad j = 1,2,...,m$ - действительные числа, причем $a_{ji} \ge 0$.

Для поиска оптимального решения задачи воспользуемся методом штрафов и видоизменим постановку:

$$\begin{aligned}
&c_1x_1 + c_2x_2 + ... + c_nx_n - M \cdot (1 - \mathrm{sgn}(b_1 - (a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n))) + \\
&-M \cdot (1 - \mathrm{sgn}(b_2 - (a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n))) + ... \\
&-M \cdot (1 - \mathrm{sgn}(b_m - (a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n - b_m))) \to \max \\
&0 \le z_i \le m_i \\
&x_i \in Z, i = 1,2,...,n
\end{aligned} \qquad (7.3)$$

Здесь $\mathrm{sgn}(x) = \begin{cases} 1, & \text{при } x \ge 0 \\ 0, & \text{при } x < 0 \end{cases}$, и M – большое положительное число. Целочисленная решетка Z^n может рассматриваться как метрическое пространство с манхэттенской метрикой. В качестве базового пространства X будем

использовать целочисленный параллелепипед в Z^n, который определяется неравенствами $0 \le z_i \le m_i, \quad i = 1,2,...,n$.

Оператор кроссовера каждой паре точек $x = (x_1, x_2, ..., x_n)$, $y = (y_1, y_2, ..., y_n) \in X$ ставит в соответствие точку $z = (z_1, z_2, ..., z_n)$, где точки $z_i = [\min(x_i, y_i), \min(x_i, y_i)] \cap Z, \quad i = 1,2,...,n$ выбираются случайным образом.

Мутация меняет случайным образом координату z_i точки z на случайно выбранное целое число из промежутка $[0, m_i], \quad i = 1,2,...,n$.

Другие составляющие эволюционной модели могут быть различными в соответствии с описанием эволюционной модели (см.выше).

7.3 Задача размещения производства без ограничений на мощности

Пусть задан граф полный граф K_n с числом вершин n, каждому ребру (i, j) которого приписан вес $\rho(i, j)$. Вершины графа будем интерпретировать, как точки возможного размещения пунктов коллективного пользования и точки потребления, вес ребра $\rho(i, j)$ - затраты на обслуживание точки i в пункте j. Каждой вершине i графа приписана величина $v(i)$ - стоимость открытия пункта обслуживания в точке i. Задача состоит в том, чтобы поместить заданное число $m<n$ пунктов в вершинах графа и приписать каждой вершине тот пункт, в котором она обслуживается. Причем в каждой вершине можно разместить не более одного пункта обслуживания и каждой вершине должен быть приписан ровно один пункт, в котором обслуживается эта вершина.

Положим

$$x_{ij} = \begin{cases} 1, & \text{если вершина } i \text{ обслуживается в пункте } j; \\ 0 & \text{в противном случае} \end{cases}$$

$$y_i = \begin{cases} 1, & \text{если пункт обслуживания размещен в вершине } i; \\ 0 & \text{в противном случае} \end{cases}$$

В принятых обозначениях задача может быть сформулирована, как задача оптимизации:

$$\sum_{i=1}^{n} v(i)\, y_i + \sum_{j=1}^{n} y_j \sum_{i=1}^{n} \rho(i,j) x_{ij} \to \min \qquad (7.4)$$

При ограничениях

$$\sum_{i=1}^{n} y_i = m; \qquad \sum_{j=1}^{n} x_{ij} = 1; \quad \sum_{j=1}^{n} y_j x_{ij} = 1, \ j = 1,2,...,n$$
$$x_{ij} \in \{0,1\}, \, y_i \in \{0,1\} \qquad (7.5)$$

При заданных значениях переменных y_i, $i = 1,2,...,n$, то есть при заданных пунктах обслуживания, оптимальное решение задачи можно найти по следующему простому правилу. Каждая вершина i соединяется ребром с тем пунктом обслуживания j, для которого $\rho(i,j) = \min\limits_{y_k=1} \rho(i,k)$. Это правило будем называть алгоритмом $\alpha(y)$. Таким образом, задача сводится к отысканию подходящего вектора $y = (y_1, y_2, ..., y_n)$ на базовом множестве $X = \left\{ y = (y_1, y_2, ..., y_n) \in B^n \mid \sum\limits_{i=1}^{n} y_i = 1 \right\}$.

Для построения эволюционной модели используем сужение стандартной метрики Хэмминга $\rho_{Нем}$ на базовое множество X. Отрезком между точками $y_1, y_2 \in X$ является пересечение $[y_1, y_2] \cap X$, где $[y_1, y_2]$ - отрезок в пространстве B^n, который задается соответствующей схемой.

Оператор кроссовера $Kr(y_1, y_2)$ выбирает случайным образом точку на множестве $[y_1, y_2] \cap X$. Для вычисления значения целевой функции по заданному значению вектора y находится значение величин x_{ij} по алгоритму $\alpha(y)$, а затем уже вычисляется значение функции в соответствии с (7.4). Мутация случайным образом меняет местами две координаты в решении y. Другие составляющие эволюционной модели выбираются произвольно.

Заметим, что к рассмотренной задаче очень близка по постановке следующая задача простой классификации: требуется разбить множество X точек метрического пространства на m попарно непересекающихся классов. В этом

случае $\rho(i, j)$-расстояние между соответствующими точками в пространстве X и $\forall i \;\; v(i) = 0$.

7.4 Задача «Судоку»

В качестве одного из примеров задач, в которых оправдано применение эволюционных моделей, рассмотрим популярную логическую игру «Судоку». Один из вариантов эволюционной модели с геометрическим оператором кроссовера для этой задачи подробно исследован в [76].

В игре «Судоку» требуется заполнить квадрат размера 9x9 позиций цифрами 1,2,…9 таким образом, чтобы в каждой строке, в каждом столбце и в каждом блоке 3x3 не было одинаковых цифр. При этом некоторые цифры, которые будем называть закрепленными, уже заняли свои позиции и эти позиции менять нельзя (рис.7.1).

							5	4
			2	7	1			
9	6	8						
				5		7		9
		3				8		
1		4		6				
						4	2	1
			9	4	3			
5	4							

Рис.7.1 Игра «Судоку»

Построим эволюционную модель задачи. Каждое допустимое решение будем представлять как элемент $x = (x_1, x_2, ..., x_9)$ прямого произведения множеств перестановок $S_{k_1} \times S_{k_2} \times ... \times S_{k_9}$, где k_i – число незанятых клеток в i-й строке квадрата. Перестановка x_i содержит лишь те числа из множества $\{1,2,…,9\}$, которые не являются закрепленными в i-й строке.

Таким образом, допустимое решение $x = (x_1, x_2, ..., x_9)$ описывает какое-то заполнение позиций квадрата цифрами, причем в каждой строке цифры попарно различны.

В качестве критерия $F(x)$ выберем сумму количеств повторений одинаковых цифр в колонках квадрата и в блоках 3x3. Задача отыскать

допустимое решение, для которого значение критерия минимально. Если для найденного решения *x* задачи значение критерия $F(x)=0$, то получено решение игры судоку.

На каждом из сомножителей S_{k_i} множества допустимых решений $X = S_{k_1} \times S_{k_2} \times ... \times S_{k_9}$ можно определить метрику (одну из метрик пространства перестановок). Таким образом, множество *X* также является метрическим пространством. Следовательно, в эволюционной модели можно использовать геометрический оператор кроссовера. Результаты исследований задачи для различных геометрических операторов кроссовера приведены в [79].

8 ЖАДНЫЕ АЛГОРИТМЫ. МАТРОИДЫ, ГРИДОИДЫ, НАСЛЕДСТВЕННЫЕ СИСТЕМЫ

Жадные (Greedy) алгоритмы возникли как точные алгоритмы решения некоторых дискретных оптимизационных задач. В частности, типичными примерами «жадных» алгоритмов являются известные алгоритмы Прима [60] и Краскала [70] для решения задачи о минимальном остовном дереве (МОД). Однако впоследствии жадными начали называть эвристические алгоритмы, которые отыскивают приближенное решение сложное задачи с низкой трудоемкостью за один цикл расчета. Напомним основные идеи этих алгоритмов.

8.1 Алгоритмы Прима и Краскала

Пусть задан граф *n*-вершинный граф $G = (V, E)$ с множеством вершин V и множеством ребер E, у которого каждому ребру $e \in E$ приписан вес $w(e)$. Вес любого частичного графа $G_1 \subseteq G$ определяется как сумма весов его ребер. Задача - отыскать остовное дерево (дерево, содержащее все вершины графа) в G, вес которого минимален.

Алгоритм Прима работает следующим образом:

Ребра упорядочиваются по возрастанию весов. На первом шаге выбирается ребро минимального веса и переносится в список выбранных ребер. На каждом последующем шаге ищется ребро минимального веса среди ребер, которые имеют общую вершину с уже выбранными ребрами и не образуют с ними цикла. Это ребро переносится в список выбранных ребер. Алгоритм заканчивает работу, когда будет построено остовное дерево (за n-1 шагов, где n-число вершин графа G) или когда невозможно будет пополнить список ребер (в том случае, когда граф несвязный)

Алгоритм Краскала отличается от алгоритма Прима лишь тем, что на очередном шаге среди оставшихся ребер ищется ребро минимального веса, не образующее цикла с уже выбранными. При этом условие наличия общей вершины с уже выбранными ребрами не является обязательным.

И тот и другой алгоритмы решают задачу о минимальном остовном дереве на связном графе за n-1 шагов (на каждом шаге добавляется новое ребро (рис. 6) .

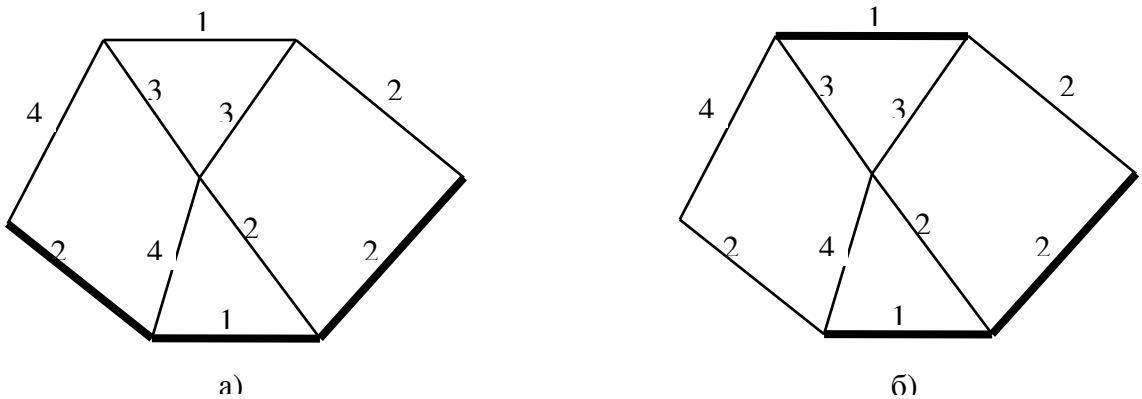


Рис.6 Первые 3 шага алгоритмов Прима (а) и Краскала (б)

Особенностью этих алгоритмов является то, что они решают задачу за один просмотр списка ребер (без возвращений).

В настоящее время «жадными» алгоритмами (Greedy algorithm) называют любой локальный алгоритм, который на каждом шаге «достаточно просто» решает локальную задачу оптимизации [29]. В частности, к этому классу алгоритмов принадлежат: алгоритм «иди в ближайший» для задачи коммивояжера, алгоритм «минимального элемента» для транспортной задачи, алгоритм «максимальной удельной стоимости» для задачи о ранце и ряд других.

К сожалению, Greedy - алгоритмы далеко не для всех задач позволяют получить оптимальное решение. Исследование класса задач, для которых точное оптимальное решение может быть получено с помощью «жадного» алгоритма, привело к понятию матроида.

8.2 Теория матроидов

Понятие матроида впервые было введено Уитни [81]. Достаточно полную современную библиографию по теории матроидов можно найти в [22,83].

Определение 8.1. Матроидом M называется пара (X,B), где X – непустое конечное множество, а B – непустая совокупность его подмножеств (называемых базами), удовлетворяющая следующим условиям:

1) никакая база не содержит в качестве собственного подмножества другую базу;

2) если B_1 и B_2 - базы и e - любой элемент из B_1, то существует элемент f из B_2, обладающий тем свойством, что множество $(B_1 \setminus \{e\}) \cup \{f\}$ также является базой. Применяя достаточное число раз свойство (2), можно показать, что любые две базы матроида M содержат одинаковое число элементов. Это число называется рангом матроида M.

Примеры матроидов.

1. Графовые (циклические) матроиды

С любым графом $G=(V,E)$ можно естественным образом связать некоторый матроид, взяв в качестве множества X множество ребер графа G, а в качестве баз - ребра остовных деревьев графа G (остовных лесов для несвязных графов). Этот матроид называется циклическим матроидом графа G и обозначается через $M(G)$.

2. Векторные матроиды

Аналогично, если X — конечное множество векторов в векторном пространстве V, то можно задать матроид на X, взяв в качестве баз всевозможные линейно независимые подмножества из X, порождающие то же подпространство, что и X. Матроид, полученный таким образом, называется векторным матроидом.

3. Матричные матроиды

Матричные матроиды отличаются от векторных матроидов, рассмотренных ранее только видом. Пусть $\{v_1, v_2, ..., v_m\}$ - элементы некоторого линейного пространства размерности *n*. Пусть $\{b_1, b_2, ..., b_n\}$ - базис пространства. Векторы $v_1, v_2, ..., v_m$ имеют однозначное разложение относительно этого базиса:

$$\begin{aligned} v_1 &= a_{11}b_1 + a_{12}b_2 + ... + a_{1n}b_n \\ v_2 &= a_{21}b_1 + a_{22}b_2 + ... + a_{2n}b_n \\ &..... \\ v_m &= a_{m1}b_1 + a_{m2}b_2 + ... + a_{mn}b_n \end{aligned} .$$

Рассмотрим матрицу коэффициентов разложения $A = \begin{pmatrix} a_{11}, a_{21}, ..., a_{m1} \\ a_{12}, a_{22}, ..., a_{m2} \\ ..... \\ a_{1n}, a_{2n}, ..., a_{mn} \end{pmatrix}$. Каждая такая матрица определяет матричный матроид *M(A)= (X,B)*, где *X* – множество ее столбцов, а элементами базы являются максимальные линейно независимые наборы столбцов матрицы.

Пусть задан матроид *M=(X,B)*. Подмножество *A* множества *X* называется независимым, если оно содержится в некоторой базе матроида *M*. Следовательно, базы *M* являются в точности максимальными независимыми множествами, то есть такими независимыми множествами, которые не содержатся ни в каких более крупных независимых множествах.

Таким образом, любой матроид однозначно определяется своими независимыми множествами. В случае векторного матроида подмножество из *E* независимо тогда и только тогда, когда его элементы, рассматриваемые как векторы в векторном пространстве, линейно независимы. Если *G* - граф, то независимыми множествами в *M(G)* являются те множества ребер из *G,* которые не содержат циклов, другими словами, множества ребер лесов, содержащихся в *G*.

Независимые множества матроида обладают следующими характеристическими свойствами:

1) любое подмножество независимого множества независимо;

2) если A и B — независимые множества, содержащие соответственно k и $k+1$ элементов, то существует элемент e, принадлежащий B и не принадлежащий A, такой, что множество $A \cup \{e\}$ независимо.

Из перечисленных свойств независимых множеств вытекает, что любое независимое множество может быть расширено до базы матроида (многократно используя свойство 2. Вышеуказанные свойства 1)–2) эквивалентны определению матроида. А именно: если семейство подмножеств множества X обладает свойствами 1)–2), то это семейство является системой независимых множеств некоторого матроида.

Подмножество множества X называют зависимым, когда оно не является независимым; минимальное зависимое множество называется циклом. Заметим, что если $M(G)$ — циклический матроид некоторого графа G, то циклами в $M(G)$ являются как раз циклы графа G. Понятно, что подмножество из X независимо тогда и только тогда, когда оно не содержит циклов.

Рассмотрим матроид $M=(X,B)$.

Определение 8.2. Рангом подмножества $A \subseteq X$ называется величина $\rho(A)$, равная мощности наибольшего независимого подмножества, содержащегося в A. Для ранга справедливы следующие свойства: [51]

1) $0 \le \rho(A) \le |A|$ для любого подмножества A из X. Заметим, что равенство $\rho(A)=|A|$ имеет место в том и только в том случае, когда множество A независимо;

2) если $A \subseteq B \subseteq X$, то $\rho(A) \le \rho(B)$; (8.1)

3) $\rho(A \cup B) + \rho(A \cap B) \le \rho(A) + \rho(B)$ для любых подмножеств $A, B \subseteq X$.

Эти три свойства ранговой функции также являются определяющими для матроида. А именно: если на множестве подмножеств множества X определена функция со свойствами 1)-3), то эта функция задает матроид. Независимым множеством этого матроида является любое подмножество A из X для которого $\rho(A)=|A|$.

Перейдем теперь к задачам оптимизации на матроидах. Матроид *M=(X,B)* называется взвешенным, если задана функция, определенная на множестве *X* - $\mu : X \to R^1$.

Весом произвольного непустого множества $A \subseteq X$, будем называть величину $\mu(A) = \sum_{x \in A} \mu(x)$. Вес пустого множества по определению полагаем равным нулю. Задача состоит в отыскании максимального независимого множества минимального веса в матроиде.

Покажем, что рассматриваемая задача оптимизации на матроиде может быть решена «жадным» алгоритмом α, описание которого приводится ниже.

Все элементы матроида упорядочим в порядке возрастания весов: $x_1, x_2, ..., x_n$. Положим $A_1=\{x_1\}$. На шаге *k*+1 будем выбирать элемент *x* из оставшейся последовательности с минимальным номером такой, что множество $A_{k+1} = A_k \cup \{x\}$ является независимым. Процесс заканчивается, когда все элементы последовательности исчерпаны.

Теорема 8.1 Построенное в результате работы алгоритма α множество является максимальным независимым множеством минимального веса.

Доказательство. Прежде всего, заметим, что «жадный» алгоритм приводит всегда к максимальному независимому множеству, то есть к базе матроида. Перенумеруем элементы матроида в поряде, каким они отбирались жадным алгоритмом. Пусть построенная база матроида $B = \{x_1, x_2, ..., x_k\}$. Рассмотрим базу $M = \{y_1, y_2, ..., y_k\}$ минимального веса. Будем предполагать, что ее элементы упорядочены в порядке возрастания весов. Найдем номер *i*, такой что

$$\mu(x_1) \le \mu(y_1), ..., \mu(x_i) \le \mu(y_i), \ \ \mu(x_{i+1}) > \mu(y_{i+1}).$$

В соответствии со свойствами матроида найдется такой индекс $j \in \{1, 2, ..., k+1\}$, что $\{x_1, x_2, ..., x_i, y_j\}$ - независимое множество. С другой стороны,

$\mu(y_j) \le \mu(y_{k+1}) < \mu(x_{k+1})$. Таким образом, жадный алгоритм не должен был выбрать элемент x_{i+1}. Полученное противоречие завершает доказательство. ■

Оказывается, что «жадный» алгоритм является своеобразной определяющей характеристикой матроида. Имеет место следующая теорема, доказательство которой можно найти в [51]:

Теорема 8.2. Пусть задано семейство непустых подмножеств *M* конечного множества *X*. Причем для всякого $A \in M$ любое непустое подмножество $B \subseteq A$ также принадлежит семейству *M*. Семейство множеств *M* является системой независимых множеств некоторого матроида тогда и только тогда, когда для любой функции весов $\mu : X \to R^1$ «жадный» алгоритм α, который описан выше, строит максимальное по включению множество минимального веса.

Обратим внимание, что «жадный» алгоритм для матроида *M=(X,B)* решает не только оптимизационную задачу, сколько задачу отыскания максимального независимого множества. Причем сам алгоритм в этом случае выглядит следующим образом:

Шаг 0. Элементы множества *X* линейно упорядочиваются по некоторому правилу. Начиная с первого элемента, производится перебор элементов в порядке, заданном упорядочением.

Шаг k. Отмечается первый из неотмеченных элементов, для которого выполняется условие присоединения: множество, полученное путем добавления рассматриваемого элемента к уже построенному множеству отмеченных элементов, является независимым.

Обобщения этого принципа построения алгоритма на структуры произвольной природы (не только на матроиды) приводит нас к понятию фрагментарного алгоритма, которое будут предметом рассмотрения следующего раздела.

8.3 Локальные алгоритмы и базы окрестностей

Приближенный вариант «жадного алгоритма» приводит к понятию локального алгоритма оптимизации.

Для начала дадим определение базы окрестностей решения. Рассмотрим задачу оптимизации вещественной функции $F: X \to R^1$ на множестве X произвольной природы:

$$F(x) \to \min_{x \in X}(\max).$$

Для любой точки $x \in X$ база окрестностей этой точки определяется как семейство подмножеств множества X, обладающее следующими свойствами:

1) точка x принадлежит каждому из множеств семейства;

2) любое конечное пересечение множеств базы окрестностей принадлежит базе окрестностей;

3) всякое объединение окрестностей из базы окрестностей принадлежит базе окрестностей.

Этих свойств достаточно для определения топологии окрестностей.

Локальный алгоритм решения задачи – это алгоритм, который позволяет найти оптимальное решение в заданной окрестности точки x_0.

Решение задачи оптимизации происходит пошагово по правилу

$F(x_{k+1}) = \min_{x \in U_k} F(x)$, где U_k – окрестность из базы окрестностей точки x_k.

Алгоритм заканчивает работу, когда $x_{k+1} = x_k$.

На базе локального алгоритма можно построить эвристический алгоритм решения задачи по следующей схеме:

1. выбирается начальная точка x_0;
2. на каждом шаге ищется решение x_k, которое оптимально в заданной окрестности точки x_{k-1};
3. алгоритм заканчивает работу, если в окрестности точки x^* точка x^* является оптимальной.

8.4 Гридоиды и допустимые множества

Естественным обобщением понятия матроида является гридоид [55,65-67]. Одно из определений гридоида следующее:

Определение 8.3. Гридоидом L называется пара (X,B), где X – непустое конечное множество, а B – непустая совокупность его подмножеств (называемых допустимыми множествами), удовлетворяющая следующим условиям:

1) $\varnothing \in B$;

2) для всякого непустого допустимого множества $Y \in B$ найдется такой элемент $y \in Y$, что разность $Y \setminus \{y\}$ допустимое множество, то есть $Y \setminus \{y\} \in B$;

3) для любых двух допустимых множеств $Y, Z \in B$, таких что $|Y|<|Z|$, найдется элемент $z \in Z \setminus Y$, такой что $Y \cup \{z\}$ – допустимое множество.

Если, кроме того, каждому элементу $x \in X$ сопоставлен его вес $\rho(x) \in R^1$, то гридоид будем называть взвешенным. При этом вес любого подмножества в X будем определять аддитивно, как сумму весов элементов этого подмножества.

Жадный алгоритм для взвешенных гридоидов, вообще говоря, не является точным. С его помощью можно найти минимальное (максимальное) по включению допустимое множество. Можно утверждать следующее. Для любого максимального множества существует система весов, при которой это множество:

1) является максимальным по весу;

2) может быть получено путем применения жадного алгоритма.

Для математических структур, более общих, чем матроиды и гридоиды, этот алгоритм уже будет давать лишь приближенные решения задачи оптимизации.

8.5 Наследственные системы

Еще одна математическая структура, для которой в принципе возможно применение жадных алгоритмов, – это наследственная система [18,19].

Пусть X – конечное множество и $A \subseteq 2^X$ – непустое семейство его подмножеств, удовлетворяющее следующей аксиоме наследственности: $a' \subset a \Rightarrow a' \in A$ для любого $a \in E$.

Определение 8.4. Пара $S = (X, A)$ – называется системой независимости или наследственной системой на X. Множества семейства A называются независимыми, все остальные подмножества X – зависимыми.

Если, кроме того, каждому элементу $x \in X$ сопоставлен его вес $\rho(x) \in R^1$, то наследственную систему будем называть взвешенной. При этом вес любого подмножества в X будем определять аддитивно, как сумму весов элементов этого подмножества.

В наследственной системе максимальное по включению независимое множество может быть построено жадным алгоритмом, аналогичным приведенным выше. Однако построить максимальное по включению независимое множество минимального веса в наследственной системе с помощью жадного алгоритма в общем случае уже не удается.

В следующем разделе будет обобщено понятие матроида, гридоида и наследственной системы. Будет рассмотрена наиболее общая структура (множество и семейство его подмножеств), для которой возможно применение жадного алгоритма для поиска максимальных по включению элементов. Безусловно, в общем случае жадный алгоритм уже не является точным алгоритмом поиска минимальных (максимальных) по весу допустимых подмножеств. Однако на таких структурах всегда может быть реализован жадный алгоритм как эвристика для поиска таких подмножеств. Это позволяет, во-первых, предлагать очень простые приближенные алгоритмы поиска оптимальных решений, во-вторых, служит основой для применения эволюционных моделей, которые будут рассмотрены в следующих разделах.

9. МАТЕМАТИЧЕСКИЕ МОДЕЛИ НА ОСНОВЕ ФРАГМЕНТАРНЫХ СТРУКТУР

Одним из основных принципов системного анализа является постулат наличия структуры у объекта исследования [107,133]. Когда говорится о сложных системах, всегда подразумевается наличие определенной структуры системы и степень сложности системы определяется степенью сложности этой структуры. Безусловно, элементы структуры сложной системы также имеют свою структуру, элементы которой в свою очередь тоже имеют структуры и т.д. [38,50] Кроме того, структура системы многогранна. Например, при моделировании работы сложной информационной системы рассматриваются ее организационная структура, техническая структура, функциональная структура и т.д.

Чтобы прервать этот процесс погружения необходимо фиксировать определенный уровень структурирования системы. Это и есть тот уровень, на котором происходит моделирование, анализ и исследование системы.

9.1 *Модели структур*

Моделью структуры системы будем называть пару (X,Ω), где X – множество определенной природы, Ω– отношение между элементами этого множества. Именно конкретные структуры являются объектом моделирования и исследования. Например, информационная структура учреждения представляет собой множество информационных объектов (тексты, таблицы, базы данных, и т.д.). Отношения между этими объектами – информационные потоки, с помощью которых происходит обмен информацией в системе. Элементами организационной структуры учреждения являются подразделения (цеха, отделы, лаборатории и т.д.), а отношениями между этими элементами является отношения подчиненности.

Исследуя сложные задачи, необходимо, прежде всего, выделить определенную структуру задачи. Дальнейшее исследование проводится уже на

основе выделенной структуры. В частности, такой подход к исследованию сложных задач приводит к принципу декомпозиции.

В подавляющем большинстве случаев математическое моделирование прикладных задач строится на основе структурного похода. Выделяется определенная структура, формализуются объекты этой структуры, отношения между объектами, а затем исследуется уже математическая задача, которая является формальным представлением соответствующей структуры.

В частности, по этому принципу построены большинство из оптимизационных моделей. Эти модели являются формальным представлением лишь одной из структур рассматриваемой прикладной задачи. Соответственно, нужно с осторожностью применять и результаты оптимизации, полученные на этих моделях. Во-первых, всякая модель носит приближенный характер. То есть оптимальное решение, полученное на основе модели, является лишь приближением истинного оптимального решения. Во-вторых, исходные данные прикладной задачи, как правило, определены с некоторой степенью неопределенности. При исследовании модели эта неопределенность снимается. Например, при исследовании вероятностных моделей предполагается заданным распределение исходных данных. Реальное же распределение может оказаться совсем иным. В-третьих, при решении оптимизационной задачи должна быть четко определена целевая функция (или хотя бы класс функций). В реальных задачах целевая функция может бать настолько сложной, что любая ее формализация будет весьма далека от истины.

Класс прикладных задач, для которых очень трудно применить стандартные оптимизационные модели, – это большинство задач прикладной экономической деятельности, задачи управления экономическими объектами, задачи оптимизации в реальных условиях с высокой степенью неопределенности (например, составление расписаний, управление запасами). Математические методы лишь помогают в решении подобных задач. Однако в большинстве случаев окончательный выбор оптимального решения задачи остается за человеком. Такие задачи отличают сравнительно низкая динамичность (эти

задачи решаются в основном таким «медленным» вычислительным устройством как человек) и высокая неопределенность исходных данных. Кроме того, при решении этих задач выставляются многочисленные условия, которые могут даже оказаться противоречивыми, системы приоритетов, которые не точно отражают истинные приоритеты, и много других подобных препятствий.

Очень часто описание условий задачи занимает столько времени и требует таких усилий, что более эффективным оказывается применение не оптимизационных методов, а просто экспертный выбор решения или случайный поиск. Такие задачи называют трудноформализуемыми [41]. Спектр подобных задач управления настолько велик и подходы к ним настолько индивидуальны, что говорить о построении какой-то общей модели пока просто невозможно. Считается, что в будущем решение этих задач лежит в области применения искусственного интеллекта [32,33,40].

Рассмотрим один достаточно общий подход к решению подобных задач. Подход, который, безусловно, не является универсальным, однако с его помощью очень часто удается снять проблемы поиска оптимального решения. При неопределенных условиях задачи естественно включить человека-эксперта в процедуру автоматического поиска оптимального решения. Новые перспективы развития вычислительной техники, связанные с высоким распараллеливанием вычислительных процессов и практической реализацией недетерминированных вычислительных систем (квантовые вычисления) в комбинации с предлагаемым подходом могут оказаться перспективными при создании многих информационных систем для решения практических задач экономического управления. Предлагаемый подход основан на обобщении понятия матроида, что приводит к общему алгоритмическому понятию фрагментарной структуры.

9.2 Аксиоматический подход к понятию фрагментарной структуры

В этом разделе будут сформулированы аксиомы, которые позволяют описать структуру множества фрагментов и определить общую универсальную схему фрагментарного алгоритма.

Определение 9.1. Фрагментарной структурой будем называть пару $\Omega = (X, \lozenge)$, где X – множество, $\lozenge$ – бинарное отношение на множестве X, обладающее следующими свойствами:

1) $\varnothing \lozenge \varnothing$
2) $\forall A, B \subseteq X \quad A \lozenge B \Rightarrow B \lozenge A$
3) $\forall A, B \subseteq X, A \lozenge \varnothing, B \lozenge \varnothing \Rightarrow (A \lozenge B \Leftrightarrow (A \cup B) \lozenge \varnothing)$

Отношение $\lozenge$ в дальнейшем будем называть отношением совместимости. Кроме того подмножества множества X, которые совместимы с пустым множеством будем называть допустимыми фрагментами.

Неформально, свойства совместимости могут быть сформулированы следующим образом:

1) пустое множество является допустимым фрагментом;
2) отношение совместимости симметрично;
3) для допустимых множеств их совместимость эквивалентна допустимости их объединения.

Следствие. $\forall A \subseteq X, A \lozenge \varnothing \Rightarrow A \lozenge A$. Допустимость фрагмента влечет его совместимость с собой.

Это свойство непосредственно вытекает из свойства 3.

Допустимый фрагмент называется минимальным фрагментом, если он не содержит никакого другого непустого допустимого фрагмента в качестве собственного подмножества.

Пусть дополнительно выполняется условие: для любого непустого допустимого фрагмента $A \subseteq X, A \neq \varnothing$ найдется такой минимальный фрагмент $B \subseteq A$, что разность $A \setminus B$ является допустимым фрагментом. Такую

фрагментарную структуру будем называть линейной. Очевидно, в линейной конечной фрагментарной структуре для любого допустимого фрагмента найдется последовательность минимальных фрагментов $B_1, B_2, ..., B_k$, такая что $B_1 \cup B_2 \cup ... \cup B_k = A$ и $\forall i = 1, 2, ..., k$ объединение $B_1 \cup B_2 \cup ... \cup B_i$ является допустимым фрагментом.

Фрагментарная структура может быть также задана с помощью унарного отношения допустимости, то есть путем задания семейства Σ допустимых множеств. При этом пустое множество входит в $\sum$, два подмножества множества *X* считаются совместными, если их объединение допустимо.

Допустимый фрагмент называется элементарным, если его нельзя представить в виде объединения двух допустимых фрагментов. В линейной фрагментарной структуре понятие элементарного и минимального фрагмента совпадают.

Допустимый фрагмент называется максимальным, если его объединение с любым непустым подмножеством *X* не является допустимым фрагментом. Заметим, что максимальный допустимый фрагмент несовместим ни с каким другими допустимыми фрагментами кроме пустого и тех фрагментов, которые являются его подмножествами.

Допустимый фрагмент будем называть квазимаксимальным, если его объединение с любым непустым допустимым фрагментом не является допустимым фрагментом.

Если задача может быть описана как задача на фрагментарной структуре, то это описание будем называть фрагментарной моделью. Приведем ряд примеров фрагментарных моделей.

9.3 Фрагментарные структуры на графах

1) Множества путей в графе. Фрагментами являются пути в графе. Отношение совместимости означает, что конец одного из путей совпадает с

началом другого. Эта фрагментарная структура линейна. Элементарные фрагменты – ребра графа.

2) Реберные покрытия графа образуют линейную фрагментарную структуру. Два покрытия совместимы, если никакие два ребра из этих покрытий не имеют общих вершин.

3) Покрытия графа звездами [34] образуют линейную фрагментарную структуру. Два покрытия совместимы, если никакие два ребра из этих покрытий не имеют общих вершин.

4) Более общим примером являются покрытия графа. Следуя [12], *G*-покрытием графа *H* будем называть набор его подграфов, каждый из которых изоморфен некоторому графу из заданного набора графов $G = \{G_1, G_2, ..., G_n\}$. Элементарным фрагментом будем считать любой подграф графа *H*, изоморфный одному из графов набора. Два фрагмента совместимы, если они реберно не пересекаются (или каждое ребро графа принадлежит не более чем *k* из составляющих их элементарных фрагментов).

9.4 Фрагментарная модель размещения многомерных объектов

Дискретная задача размещения сложных объектов имеет многочисленные приложения и рассматривается во многих публикациях [3,38,49].

Пусть имеется набор замкнутых подмножеств $\{T_1, T_2, ..., T_k\}, T_i \subseteq R^n$, каждое из которых обладает непустой внутренностью и некоторое подмножество $T_0 \subseteq R^n$. Размещением множества T_i в T_0 будем называть изометричное вложение $\varphi_i : T_i \to T_0$, то есть однозначное отображение, при котором сохраняются расстояния между точками множеств. Другими словами, если $\rho(x, y)$ - заданная функция расстояния в R^n, то $\forall x, y \in T_i \quad \rho(\varphi_i(x), \varphi_i(y)) = \rho(x, y)$.

Множество всех изометричных вложений объектов $\{T_1, T_2, ..., T_k\}$ образует фрагментарную структуру. Здесь два изометричных вложения являются совместимыми в том и только в том случае, когда внутренности их образов в T_0 не пересекаются.

9.5 Фрагментарная модель расписания работ

Задано множество работ $\{A_1, A_2, ..., A_k\}$, связанных отношением частичного упорядочения $\preceq$. Для каждой работы определена длительность d_i, $i = 1,2,...,k$. Расписанием будем называть любой набор интервалов $P = \{(t_1^i, t_2^i)\}_{i \in I \subseteq \{1,2,...,k\}}$, для каждого из которых имеет место неравенство $t_2^i - t_1^i \geq d_i, i = 1,2,...,k$. Будем говорить, что работа A_i входит в расписание P, если $i \in I$. Два расписания называются совместимыми, если, во-первых, не пересекаются наборы работ, входящих в эти расписания, во вторых, для любых двух работ A_i, A_j, каждая из которых входит хотя бы в одно из этих расписаний, из условия $A_i \preceq A_j$ следует, что $t_2^i \leq t_1^j$.

9.6. «Жадный» алгоритм на фрагментарной структуре

Пусть задана фрагментарная структура $\Omega = (X, \Diamond)$. Множество всех элементарных фрагментов будем называть базой фрагментарной структуры. Пусть база фрагментарной структуры конечна. Зададим произвольный линейный порядок на базе фрагментарной структуры и перенумеруем множества базы в соответствии с этим порядком.

Квазимаксимальный допустимый фрагмент может быть построен следующим «жадным» алгоритмом. На шаге с номером 0 выбирается фрагмент $\varnothing$. На каждом следующем шаге выбирается первый допустимый фрагмент из базы, который не совпадает с выбранными на предыдущих шагах элементарными фрагментами и совместим с уже построенным на предыдущем шаге фрагментом. Этот элементарный фрагмент объединяется с фрагментом, построенным на предыдущем шаге алгоритма. Алгоритм заканчивает работу, когда очередной шаг невозможен. То есть, либо исчерпано множество элементарных фрагментов, либо ни один из оставшихся фрагментов несовместим с уже построенным.

В дальнейшем фрагментарным алгоритмом будем называть любой алгоритм построения квазимаксимального фрагмента на основании базы фрагментарной структуры.

Будем говорить, что фрагментарная структура $\overline{\Omega} = (X, \overline{\Diamond})$ является расширением фрагментарной структуры $\Omega = (X, \Diamond)$, если $\forall A, B \subseteq X \quad A \Diamond B \Rightarrow A \overline{\Diamond} B$.

Теорема 9.3 Фрагментарная структура $\overline{\Omega} = (X, \overline{\Diamond})$ является расширением фрагментарной структуры $\Omega = (X, \Diamond)$ в том и только в том случае, когда каждый допустимый фрагмент в Ω является допустимым и в $\overline{\Omega}$.

Доказательство. Действительно, из определения расширения следует необходимость свойства. Пусть теперь всякий допустимый фрагмент в Ω является допустимым и в $\overline{\Omega}$. Пусть множества $A,B \subseteq X$ –совместимы во фрагментарной структуре Ω. Из этого следует, что их объединение $A \cup B$ допустимо в Ω и, следовательно, допустимо в $\overline{\Omega}$. Это и означает, что множества A,B являются совместимыми в $\overline{\Omega}$.■

Заметим, что любая фрагментарная структура может быть расширена до линейной. Пусть $\Omega = (X, \Diamond)$ - произвольная фрагментарная структура. В качестве базы фрагментарной структуры $\overline{\Omega}$ возьмем все одноэлементные подмножества допустимых множеств в Ω. Два допустимых множества $A,B \subseteq X$ будем считать совместимыми в $\overline{\Omega}$ в том и только в том случае, когда они являются подмножествами одного и того же допустимого множества в Ω.

Пример:

Фрагментарная структура – «деревья в связном графе» может быть расширена до фрагментарной структуры «множества ребер без циклов» в том же графе. Для структуры «деревья в связном графе» фрагментарным алгоритмом был алгоритм Прима, а алгоритм Краскала является фрагментарным алгоритмом для линейного расширения этой структуры.

Аналогичную процедуру можно выполнить и по отношению к известному алгоритму «иди в ближайший» для поиска кратчайшего гамильтонова цикла в связном графе. Фрагментарная структура задается всевозможными простыми

путями в графе и гамильтоновыми циклами. Условие совместимости – два пути (понимаемые как множества ребер) совместимы, если их объединение также является путем или гамильтоновым циклом. Алгоритм «иди в ближайший» - это фрагментарный алгоритм для описанной выше фрагментарной структуры. Опишем линейное расширение этой структуры. Базой являются отдельные ребра графа. Допустимыми множествами являются любые наборы ребер, которые не образуют циклов, и все гамильтоновы циклы. Фрагментарный алгоритм поиска кратчайшего гамильтонова цикла выглядит теперь следующим образом: ребра упорядочиваются по весу. На каждом шаге выбирается наиболее «легкое» ребро из еще не просмотренных ребер, которое не образует цикла с выбранными ребрами (кроме гамильтонова на последнем шаге). Это ребро добавляется в множество выбранных ребер.

Для реализации любого фрагментарного алгоритма на фрагментарной структуре достаточно задать условие присоединения отдельного фрагмента к множеству уже выбранных фрагментов. Это условие в терминах отношения совместимости выглядит следующим образом: рассматриваемый фрагмент совместим с объединением уже выбранных фрагментов.

Наоборот, условие присоединения однозначно определяет на множестве фрагментов линейную фрагментарную структуру. Элементарными фрагментами в этой структуре являются одноэлементные множества, допустимые фрагменты - все множества, которые могут быть получены из пустого путем последовательного добавления элементарных фрагментов с выполнением условия присоединения.

9.7 Оценка сложности фрагментарного алгоритма

Пусть общее число фрагментов, которые используются при отыскании решения задачи N. Обозначим верхнюю оценку трудоемкость проверки условия присоединения на каждом шаге – m**,** а n – верхняя оценка числа фрагментов, составляющих решение задачи. Трудоемкость сортировки фрагментов составляет $N \ln N$. Следовательно, трудоемкость фрагментарного алгоритма может быть

оценена числом $N\cdot \ln N\cdot m\cdot n$. Если трудоемкость m проверки условия присоединения на каждом шаге полиномиально зависит от N, то фрагментарный алгоритм также является полиномиальным.

9.8 Матроиды, гридоиды, наследственные системы и фрагментарные структуры

Очевидно, что понятие фрагментарной структуры является более общим, чем понятие матроида, гридоида и наследственной системы. В частности, любой гридоид и, следовательно, любой матроид является линейной фрагментарной структурой. Следующей задачей будет установить, в каком случае фрагментарная структура является матроидом или гридоидом.

Определение 9.2. Пусть задана линейная фрагментарная структура $\Omega = (X, \Diamond)$ на конечном множестве X. Рангом подмножества $A \subseteq X$ будем называть число $rg(A)$, равное мощности максимального допустимого фрагмента, содержащегося во множестве A.

Очевидными являются следующие свойства ранга:

1) для любого подмножества $A \subseteq X$ справедливо неравенство $0 \le rg(A) \le |A|$;

2) для любых подмножеств $A \subseteq B \subseteq X$ имеет место неравенство $rg(A) \le rg(B)$.

Теорема 9.4 Для того чтобы линейная фрагментарная структура являлась матроидом необходимо и достаточно, чтобы для любых подмножеств $A, B \subseteq X$ выполнялось неравенство:

$$rg(A \cup B) + rg(A \cap B) \le rg(A) + rg(B)$$

Доказательство теоремы непосредственно вытекает из свойств независимых множеств матроида (7.1), которые приведены в разделе 7.2.

Аналогичные теоремы имеют место и для гридоидов и наследственных систем.

Теорема 9.5 Для того чтобы линейная фрагментарная структура (X, E) была гридоидом необходимо и достаточно, чтобы для любых $A, B \subseteq E$ таких, что $|A| < |B|$ существовал элемент $x \in B \setminus A$ такой, что $A \cup \{x\} \in E$.

Теорема 9.6 Для того чтобы линейная фрагментарная структура (X, E) была наследственной системой необходимо и достаточно, чтобы для любых подмножеств $A \in E, A \neq \varnothing$ и $\forall x \in A$ выполнялось условие $A \setminus \{x\} \in E$.

Доказательство этих теорем непосредственно следуют из определения соответствующих структур.

Фрагментарный алгоритм позволяет за относительно малое число шагов найти максимальный фрагмент фрагментарной структуры. При этом веса фрагментов, другие особенности, которые имеют место в прикладных задачах, должны быть учтены на этапе описания фрагментарной структуры.

9.9 Достижимость

Фрагментарный алгоритм гарантирует отыскание максимального фрагмента, но не гарантирует оптимальности решения по критериям, которые используются в задаче.

Рассмотрим задачу отыскания оптимального решения на фрагментарной структуре *X* в соответствии с критерием $F(x) \to \max(\min)$.

Определение 9.3. Будем говорить, что фрагментарная структура оптимизационной задачи обладает свойством достижимости, если существует такое упорядочение элементарных фрагментов, для которого фрагментарный алгоритм приводит к оптимальному решению задачи.

Очевидна следующая

Теорема 9.7 Для того чтобы фрагментарная структура задачи обладала свойством достижимости необходимо и достаточно, чтобы пересечение множества оптимальных решений задачи и множества максимальных фрагментов было непустым.

Рассмотрим, как пример задачу симметричного размещения [25]. Упорядочим каким-либо образом множество *X* графов-решеток, которые нужно вложить в решетку основу. Занумеруем натуральными числами вершины решетки – основы. Будем рассматривать в качестве фрагмента вершины решетки - основы. Условие присоединения очередного фрагмента формулируется следующим образом: присоединяется (ищется изоморфное вложение) очередной (еще не выбранный) граф из множества *X* так, чтобы номер вершины, находящейся в левом верхнем углу решетки был минимальным при выбранной нумерации. При этом вновь присоединенный элемент не должен пересекаться с уже выбранными элементами размещения.

Пример такого размещения приведен на рис.9.1 Очевидно, что любое размещение фрагментов может быть реализовано таким образом за счет соответствующего упорядочения вершин решетки-основы.

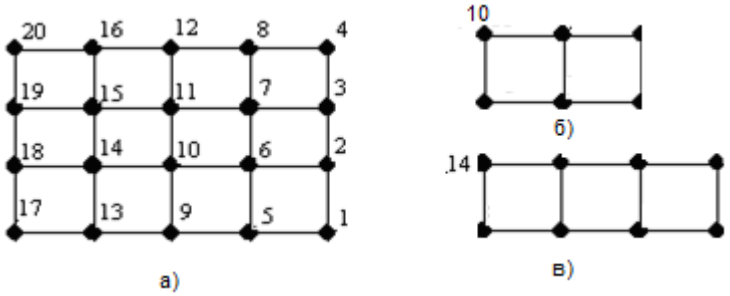


Рис. 9.1 Размещение фрагментов. а) решетка основа
б), в) размещение решеток в основе

Таким образом, предложенный способ формирования фрагментарной структуры обладает свойством достижимости.

В качестве следующего примера рассмотрим классическую транспортную задачу в следующей постановке:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \to \min$$

При ограничениях

$$\sum_{j=1}^{m} x_{ij} = a_i, \quad i = 1,2,...,n$$

$$\sum_{i=1}^{n} x_{ij} = b_j, \quad i = 1,2,...,m$$

$$x_{ij} \geq 0$$

Решение будем искать в виде суммы матриц-фрагментов $u(i, j, x)$, где

$$u_{kl}(i, j, x) = \begin{cases} x, & \text{при } k = i, l = j \\ 0 & \text{при } k \neq i \text{ или } l \neq j \end{cases}.$$

Каждая такая матрица описывает клетку транспортной таблицы.

В качестве условия присоединения выберем условия аналогичные тем, которые используются при построении начального допустимого плана задачи методом северо-западного угла: очередная клетка –фрагмент помещается в первый свободный северо-западный (верхний левый) угол таблицы.

Теорема 9.8. Предложенная выше фрагментарная структура для транспортной задачи не обладает свойством достижимости.

Доказательство теоремы следует из того, что не любые базисные наборы клеток транспортной таблицы могут быть получены методом северо-западного угла. Пример набора базисных клеток, которые не могут быть получены таким способом, приведен на рис.9.2 ■

xx xx xx

xx xx

xx xx

xx xx

Рис. 9.2 Недостижимое решение

С другой стороны, можно легко определить фрагментарную структуру транспортной задачи, для которой выполняется свойство достижимости. Условие совместимости будет выглядеть так: задается нумерация клеток таблицы, выбирается первая клетка в заданной нумерации, которая не образует цикла (независима) с уже выбранными клетками.

9.10 Примеры фрагментарных алгоритмов для некоторых модельных задач

Как пример применения фрагментарного алгоритма опишем механизм решения одной из задач покрытия произвольного графа типовыми подграфами - задачи минимального реберного покрытия графа звездами с ограниченным числом лучей.

Задача состоит в следующем: в заданном графе $G=(V,E)$, каждому ребру которого $e \in E$ приписан вес $\rho(e) \in R^1$, найти реберное покрытие звездами с числом лучей не более чем k, с минимальным числом компонент связности и при этом вес которого минимален. Эта двукритериальная задача исследовалась в работе [14]. Доказано, что эта задача, с одной стороны обладает свойством полноты, то есть, полное множества альтернатив задачи совпадает в худшем случае с множеством Парето, с другой стороны, поиск отдельного эффективного решения задачи – задача *NP*-трудная.

Построим фрагментарный алгоритм решения задачи. В качестве фрагмента будут рассматриваться звезды с числом лучей не более чем k. Ребра графа упорядочиваются по возрастанию весов.

Выбираем ребро в графе, вес которого минимален. Перебираем ребра графа в порядке возрастания весов, добавляя их к фрагменту-звезде в качестве лучей, если это возможно и число лучей звезды еще не превышает k. Когда первая звезда построена, удаляем все ребра, которые инцидентны вершинам этой звезды. Для оставшейся части графа повторяем процедуру до тех пор, пока во множество фрагментов не попадут все вершины графа.

Другим примером применения фрагментарного алгоритма является классическая задача погашения взаимных задолженностей [41]. Пусть задан

ориентированный граф с множеством вершин $\{v_i\}_{i=1}^{n}$. Каждому ребру (v_i, v_j) графа приписан вес ρ_{ij}. Задача найти поток (τ_{ij}) через ребра графа, который обладал бы максимальным суммарным весом $\sum_{i,j=1}^{n} \tau_{ij}$, причем $\forall i, j \quad 0 \le \tau_{ij} \le \rho_{ij}$.

Задача о погашении взаимных долгов может быть сформулирована как задача линейного программирования:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \tau_{ij} \to \max$$
$$0 \le \tau_{ij} \le a_{ij} \quad i, j = 1,2,...,n$$
$$\sum_{j=1}^{n} \tau_{ij} = \sum_{j=1}^{n} \tau_{ji} \quad i = 1,2,...,n$$

Оптимальное решение этой задачи может быть получено стандартными методами линейного программирования. Однако алгоритм линейного программирования не допускает фрагментации, то есть решение, полученное этим алгоритмом, не разлагается в объединение отдельных частей.

Приведем фрагментарный алгоритм решения задачи. В качестве фрагментов выбираются циклы погашения – ориентированные циклы в графе *G*, дуги которых взвешены числами (τ_{ij}), где $\forall i, j \quad 0 \le \tau_{ij} \le \rho_{ij}$. Алгоритм устроен следующим образом. Вначале ищется ориентированный цикл в графе *G*. В найденном цикле всем ребрам приписывается вес, величина которого равна *M* - минимальному из начальных весов ребер цикла. Затем этот фрагмент добавляется в решение и начальные веса всех ребер, входящих в цикл, уменьшаются на величину *M*. Удаляются все ребра, вес которых стал равен нулю. Процедура повторяется до тех пор, пока в графе существуют циклы. Решением задачи является найденная система циклов погашения. Исследования показывают, что, несмотря на то, что фрагментарный алгоритм в этом случае дает лишь приближенное решение задачи, скорость работы алгоритма и возможность использования в диалоге «Человек-

машина» в реальных системах погашения взаимных долгов делают его удобным инструментом решения практических задач.

Еще одна задача, в которой удается удачно применить фрагментарный подход, - задача размещения прямоугольных блоков рекламы на страницах рекламных изданий, которая описана в подразделе 2.2. Прямоугольные блоки из заданного набора блоков требуется разместить на прямоугольных листах без самопересечений по критерию симметрии или антисимметрии. Фрагментарный подход позволяет успешно решать эту задачу в диалоге «Человек машина». Здесь в качестве фрагментов используются прямоугольные блоки. Отношение порядка может быть выбрано различными способами – по возрастанию площади, периметра, по содержанию рекламного блока и т.д. Условием присоединение является отсутствие пересечений с уже установленными блоками и минимизация функционала симметрии (антисимметрии) [25].

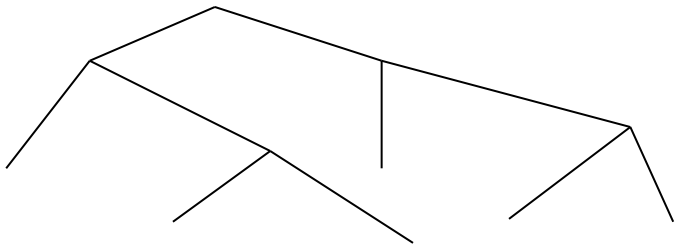


Рис.9.3 Бинарное дерево

Рассмотрим теперь задачу выделения бинарного дерева минимального веса в заданном графе $G=(V,E)$, каждое ребро которого e имеет вес $\rho(e)\in R^1$. Напомним, что бинарным деревом называется связный граф, у которого ровно одна вершина имеет степень 2, а все остальные имеют степени либо 3, либо 1 (рис.9). Такая задача часто возникает при решении задач бинарного поиска, при построении схем сортировки элементов, в задачах оптимального проектирования.

В качестве фрагментов рассмотрим множество всех двузвенных цепей в графе G. Все двузвенные цепи упорядочиваются по возрастанию весов цепей (сумма весов ребер, их составляющих). На первом шаге алгоритма выбирается цепь минимального веса. Условие присоединения имеет следующий вид: ищется двузвенная цепь минимального веса, средняя вершина которой совпадает с одной из висячих вершин уже построенного на предыдущем шаге алгоритма дерева, а концы не принадлежат множеству вершин этого дерева. Условие остановки

алгоритма – на очередном шаге не удалось найти ни одной двузвенной цепи, удовлетворяющей условию присоединения. Обратим внимание, что на каждом шаге условия присоединения могут быть дополнены другими условиями. Например, дополнительно могут учитываться такие требования как минимизация диаметра или радиуса дерева, максимальная сбалансированность дерева, другие формальные или неформальные условия.

Построим фрагментарную модель для задачи размещения, которая рассматривалась в разделе 2.1.4. Эта задача может быть сформулирована в терминах теории графов, как задача покрытия графа звездами. Пусть задан двудольный граф *G*, ребра которого взвешены числами v_{ij}, $i = 1,2,..,n$; $j = 1,2,...,m$. Множество вершин *I* соответствует центрам обслуживания, множество вершин *J* соответствует потребителям. Задача построить покрытие графа *G* звездами с центрами в множестве вершин *I*. Причем покрытие должно быть максимально равномерным по центрам обслуживания. Другими словами, введем критерий F_i – суммарный вес ребер, исходящих из вершины *i*. Значения критериев должны отличаться как можно меньше. Фрагментами в модели будем считать взвешенные ребра графа *G*. Упорядочим ребра в определенной последовательности. Будем последовательно выбирать ребра в этой последовательности таким образом, чтобы на каждом шаге каждый потребитель соединялся ребром не более чем с одним центром обслуживания. Алгоритм заканчивает работу, когда либо все потребители окажутся закрепленными за центрами обслуживания, либо когда будут просмотрены все ребра графа.

Если существует оптимальное решение задачи о размещении в предлагаемой постановке, то для некоторого порядка ребер фрагментарный алгоритм найдет это размещение. Действительно, достаточно упорядочить ребра таким образом, чтобы первые из них соответствовали ребрам в заданном решении задачи. Более того, любое допустимое решение задачи о покрытии двудольного графа звездами может быть построено путем применения фрагментарного алгоритма.

9.11 Фрагментарные модели в системах поддержки принятия решений

Метод создания фрагментарных алгоритмов, который описан в настоящей работе, может быть перенесен практически на все разделы прикладных дискретных задач. Простота формализации делает возможным создание на основе фрагментарного подхода систем поддержки принятия решений в таких трудно формализуемых, а порой и труднорешаемых задачах, как задачи составления оптимальных расписаний, задачи оптимального размещения объектов, задачи упаковки объектов сложной формы и множество других.

Фрагментарный алгоритм оставляет ЛПР два основных механизма вмешательства в процесс поиска оптимального решения – это способ упорядочивания фрагментов и условие присоединения. Благодаря этой свободе можно в процессе принятия решений ввести нечеткие, «лингвистические» требования, такие как «самый лучший», «наиболее подходящий», «красивый с точки зрения ЛПР» и т.д.

Таким образом, во фрагментарных алгоритмах нечеткость и размытость некоторых формулировок может быть совмещена с жесткими правилами упорядочивания и присоединения фрагментов. Это позволяет надеяться, что механизм фрагментарных алгоритмов получит широкое применение в создании систем поддержки принятия решений в разных областях применения интеллектуальных компьютерных систем.

Единственный класс задач дискретной оптимизации, для которых фрагментарные алгоритмы приводят к точным решениям, - это задачи на матроидах и гридоидах. Во всех остальных ситуациях фрагментарный алгоритм является эвристическим алгоритмом поиска приближенного решения.

Однако простота реализации алгоритма и ряд особенностей, которые делают его практически универсальным для широкого класса задач, приводят к вопросу об общих свойствах задач, для которых в принципе возможно применение фрагментарного алгоритма.

Фрагментарные структуры – это объекты, для которых в принципе возможно определение понятия «жадного алгоритма». Жадный алгоритм на фрагментарной структуре – это фрагментарный алгоритм при определенном способе упорядочения элементарных фрагментов фрагментарной структуры. Для многих экстремальных задач, которые могут быть заданы как задачи на фрагментарных структурах, можно предложить простые эвристические алгоритмы поиска приближенных решений. При этом фрагментарный алгоритм обладает свойством достижимости для заданного класса задач, если для любой индивидуальной задачи класса можно найти такое упорядочение элементарных фрагментов, при котором применение фрагментарного алгоритма приводит к точному оптимальному решению задачи. В частности, фрагментарные алгоритм для матроидов и гридоидов всегда обладают свойством достижимости.

Если фрагментарный алгоритм обладает свойством достижимости, то точное оптимальное решение экстремальной задачи может быть найдено путем перебора всех перестановок элементарных фрагментов и применения фрагментарного алгоритма для каждой такой перестановки. Однако такой алгоритм, безусловно, является сложным. Если ограничиться не всеми перестановками, а какой-то их частью, выбираемой по определенным правилам, то можно получать различные варианты эвристических алгоритмов поиска. Один из таких эвристических подходов к решению дискретных оптимизационных задач рассматривается в следующем разделе.

10. ЭВОЛЮЦИОННАЯ МОДЕЛЬ НА ФРАГМЕНТАРНЫХ СТРУКТУРАХ

Во многих дискретных задачах оптимизации множество допустимых решений может рассматриваться как фрагментарная структура. В этот класс попадают многочисленные задачи на графах с аддитивным критерием, в частности, задачи покрытия, задачи теории расписаний, задачи маршрутизации, задачи многоэтапного управления запасами и практически все задачи, для которых применим метод динамического программирования. Обратим внимание на то, что оптимальное решение в подобных задачах всегда является максимальным фрагментом фрагментарной структуры. Таким образом, перебрав все максимальные фрагменты, можно с гарантией найти оптимальное решение задачи.

10.1 Оптимизация на фрагментарных структурах

Напомним, что любой максимальный фрагмент может быть построен из элементарных фрагментов путем применения фрагментарного алгоритма при некоторой заданной начальной перестановке элементарных фрагментов. Следовательно, допустимые решения оптимизационных задач на фрагментарных структурах могут кодироваться перестановками. Механизм кодирования следующий:

1) задается перестановка элементарных фрагментов;

2) применяется фрагментарный алгоритм при заданной перестановке фрагментов и отыскивается максимальный фрагмент.

Этот фрагмент и есть то допустимое решение, кодом которого является первоначальная перестановка. Конечно, кодом ее можно назвать весьма условно. Полученное соответствие между допустимыми решениями оптимизационной задачи и перестановками не является ни инъективным, ни сюръективным. Однако, каждой перестановке отвечает единственное допустимое решение оптимизационной задачи. Этого достаточно для построения эволюционной модели задачи. Базовое множество этой модели – множество S_N всех

перестановок заданного порядка *N*. Перестановка *s* из *N* элементов представляется словом $i_1 i_2 ... i_N$, в котором каждая буква $i_k \in \{1,2,...,N\}$, причем буквы не повторяются.

Таким образом, если рассматриваемая дискретная задача имеет фрагментарную структуру и для нее удалось построить фрагментарную модель, то для этой задачи можно построить и эволюционную модель на перестановках. В качестве канонической эволюционной модели для задач с фрагментарной структурой будем использовать модель G_4, которая рассматривалась в разделе 3.

10.2 Эволюционно-фрагментарная модель

Напомним определение оператора кроссовера $Kr: S_N \times S_N \to S_N$ в модели G_4. Пусть заданы две перестановки $s_1 = i_1 i_2 ... i_N$ и $s_2 = j_1 j_2 ... j_N$. Будем просматривать эти перестановки слева направо и выполнять следующие операции: из первых букв двух слов-перестановок выбирается одна буква (по заданному правилу выбора) и записывается как буква нового слова-перестановки. Эта буква вычеркивается из слов-родителей. Буквы слов-родителей, следующие за вычеркнутой, сдвигаются влево на один символ. Далее процедура выполняется, пока все буквы родительских слов не будут исчерпаны. Правило кроссовера сохраняет отношение порядка между буквами слов-родителей. Пусть заданы две перестановки $s_1 = i_1 i_2 ... i_N$ и $s_2 = j_1 j_2 ... j_N$ и пусть в каждой из этих перестановках число *i* предшествует числу *j*. Тогда результат канонического кроссовера этих перестановок $Kr(s_1, s_2)$ также будет обладать этим свойством.

Таким образом, отношение порядка между элементами перестановки является наследственным свойством в рассматриваемой эволюционной модели. В соответствии с теоремой наследственности, это приводит к тому, что в процессе работы эволюционного алгоритма элементы перестановок будут выстраиваться в том порядке, который «более отвечает» оптимальному решению.

Приведем варианты применения правила для двух перестановок.

1. Минимальный кроссовер. Из двух первых букв перестановок родителей выбирается минимальная по значению. Пример выполнения процедуры кроссовера $\begin{cases}23164589\\85426931\end{cases} \Rightarrow 23164589$.

2. Максимальный кроссовер. Из двух первых букв перестановок родителей выбирается максимальная по значению. Пример: $\begin{cases}23164589\\85426931\end{cases} \Rightarrow 85426931$

3. Случайный кроссовер. Из двух первых букв перестановок родителей одна выбирается случайно с вероятностью 1/2. Пример: $\begin{cases}23164589\\85426931\end{cases} \Rightarrow 82316459$

Каноническое правило мутации в рассматриваемой модели заключается в замене местами двух наугад выбранных элементов в перестановке.

Опишем теперь и другие составляющие эволюционной модели, которую будем называть эволюционно-фрагментарной или ЭВФ моделью. Не нарушая общности, будем считать, что оптимизационная задача – это задача на максимум и значение критерия для всех допустимых решений положительно.

Значение критерия на перестановке вычисляется следующим образом: к заданной перестановке элементарных фрагментов применяется фрагментарный алгоритм и отыскивается максимальный фрагмент. Значение целевой функции задачи на этом максимальном фрагменте будем рассматривать как значение критерия на заданной перестановке. Таким образом, фрагментарный алгоритм используется для вычисления значения критерия на перестановках.

Правило построения начальной популяции. Объем (число элементов) популяции задается натуральным числом $V \le 2^N$. Случайным образом генерируется V перестановок N элементов. Распределение равномерное. Перестановки в выборке могут повторяться. Учитывая ограниченность эволюционной модели (без оператора мутации все полученные в рамках модели решения принадлежат выпуклой оболочке начальной популяции), рекомендуется

выполнять несколько запусков эволюционного алгоритма с разными начальными популяциями.

Правило селекции. Выбирается M пар из текущей популяции объема V. Каждый элемент пары выбирается случайным образом. Распределение может быть равномерным (вероятность выбора одинакова для всех элементов популяции) или пропорциональным (вероятность выбора элемента пропорциональна значению целевой функции на этом элементе)

Рис. 10.1 Блок-схема эволюционно фрагментарного алгоритма

Правило отбора в новую текущую популяцию стандартное. Все элементы предыдущей популяции и потомки, полученные в результате кроссовера и

мутации, упорядочиваются по убыванию значения критерия. Выбираются первые по порядку V элементов в этом упорядочении.

Правило остановки может быть различным. Например, по числу поколений или по времени работы алгоритма.

Блок - схема эволюционно-фрагментарного (ЭВФ) алгоритма для полученной модели приведена на рис.10.1. Этот алгоритм является универсальным. Он работает для любых задач на фрагментарных структурах. Индивидуальным является лишь механизм вычисления значения критерия с помощью фрагментарного алгоритма.

11. ЭВОЛЮЦИОННО-ФРАГМЕНТАРНЫЕ МОДЕЛИ ЗАДАЧ ДИСКРЕТНОЙ ОПТИМИЗАЦИИ

В этом разделе будет рассмотрен ряд дискретных оптимизационных задач, для которых можно сравнительно легко построить эволюционно-фрагментарную модель. Основные правила построения ЭВФ-модели следующие:

1. Представление решения задачи в виде объединения элементарных фрагментов. Описание множества элементарных фрагментов. Задание условия присоединения и формализация фрагментарного алгоритма. Проверка свойств достижимости.

2. Процедуры кодирования-декодирования, которые позволяют по решению построить его код(или один из кодов), а по коду восстановить решение (или одно из решений). Каждый из кодов является перестановкой элементарных фрагментов.

3. Целевая функция на множестве кодов и алгоритм, который позволяет по коду-перестановке вычислить значение целевой функции на этом решении.

4. Универсальная(стандартная) эволюционная модель.

Первые три этапа построения относятся к фрагментарной модели задачи и отражают ее индивидуальность. Последний этап – эволюционная модель – является общим для всех задач. Этот этап был подробно описан в предыдущем разделе.

Примеры моделей, которые будут рассмотрены ниже, условно разделены на типы, относящиеся к различным разделам дискретной оптимизации.

11.1 Задачи на графах

Напомним, что графом называется пара (X,G), где X – конечное множество – вершины графа, а G - множество пар (неупорядоченных или упорядоченных для ориентированных графов) элементов из X. Будем рассматривать лишь графы без кратных ребер. Вершины графа будем нумеровать последовательно числами

1,2,… За более детальной информацией по теории графов отсылаем читателя к книгам [12,17,21,51].

Задача о гамильтоновом цикле

Требуется найти простой цикл, содержащий все вершины графа. Эта задача может рассматриваться как задача оптимизации. В качестве элементарных фрагментов будем рассматривать ребра графа, перенумерованные числами 1,2,…*m*. Условие присоединения очередного ребра – ребро либо образует простой(без самопересечений) путь максимальной длины с уже выбранными ребрами или гамильтонов цикл. Задача найти фрагмент с максимальным числом ребер. Фрагментарный алгоритм просматривает выбранную перестановку ребер и находит в ней первое ребро, которое удовлетворяет условию присоединения. Это ребро присоединяется к уже построенному фрагменту. Затем просмотр перестановки ребер начинается сначала. Алгоритм завершает работу, когда ни одно из ребер не может быть присоединено к фрагменту, то есть когда получен максимальный фрагмент. Критерием в этой задаче является число ребер в построенном максимальном фрагменте. Очевидно, всякий гамильтонов цикл в графе является достижимым, то есть является максимальным фрагментом при определенной перестановке ребер. Таким образом, к задаче применима эволюционно-фрагментарная модель.

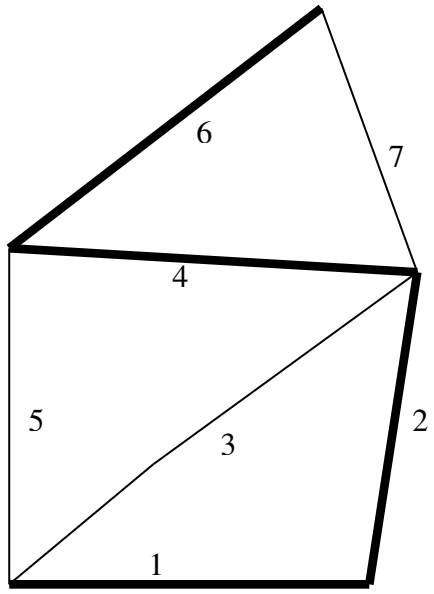


Рис. 11.1 Результат работы фрагментарного алгоритма для задачи гамильтонов цикл

Задача о максимальной клике в графе

Требуется найти клику(полный подграф) в заданном графе, содержащую максимальное число вершин графа. Для построения фрагментарной модели в качестве элементарных фрагментов будем рассматривать вершины графа, перенумерованные числами 1,2,…*n*. Условие присоединения очередной вершины к существующему фрагменту – вершина смежна всем вершинам этого фрагмента. Критерием в задаче является число вершин в построенном максимальном фрагменте. Очевидно, любая максимальная клика является достижимой в предлагаемой фрагментарной модели. Таким образом, к этой задаче также применима стандартная эволюционная модель.

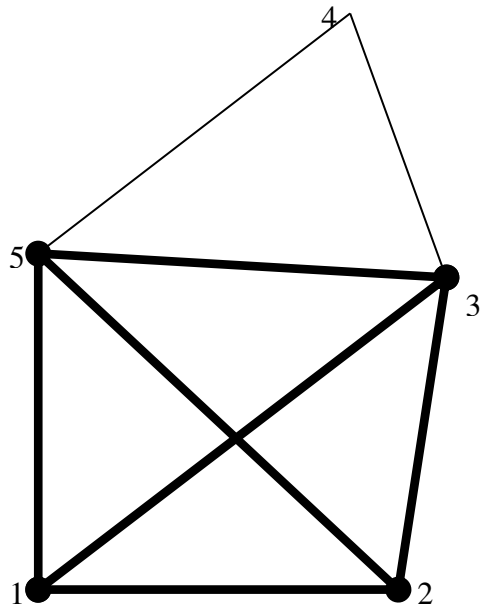


Рис. 11.2 Результат работы фрагментарного алгоритма для задачи о максимальной клике

Задача о максимально плоском суграфе

Задача состоит в следующем: выделить в заданном связном графе плоский суграф с максимально возможным числом ребер. Элементарными фрагментами в модели являются ребра графа. Ребра нумеруются числами 1,2,…,*n*. Решение задачи кодируется списком(упорядоченной последовательностью) элементарных фрагментов.

Фрагментарный алгоритм, который по последовательности ребер строит максимальный плоский суграф-фрагмент, состоит из двух этапов.

1. Построение остовного дерева. На каждом шаге первого этапа просматривается список-решение и из него выбирается ребро, одна и только одна вершина которого уже присутствует в выбранных на предыдущих шагах ребрах. На начальном шаге выбирается первое ребро. Ребра укладываются на плоскость последовательно, в соответствии с порядком выбора, таким образом, чтобы

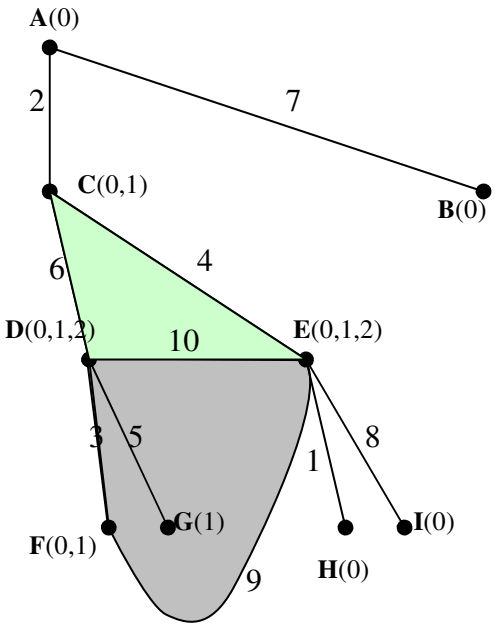


Рис.11.3 Результат работы фрагментарного алгоритма для задачи о максимально плоском суграфе

свободный конец уложенного ребра находился ниже и правее от конца, который инцидентен уже уложенному ребру. Затем список просматривается сначала. В конце первого этапа на плоскость будет уложено ровно n-1 ребер, где n – число вершин графа.

2. Укладка оставшихся ребер. Область плоскости, ограниченная графом(вся плоскость) помечается числом 0(белый цвет). Все вершины дерева помечаются числом 0(раскрашиваются в белый цвет). Оставшиеся ребра просматриваются один раз последовательно и на каждом шаге ребро добавляется к укладке без пересечений, если оба его конца имеют хотя бы одну одинаковую метку(цвет). После добавления и укладки ребра появляется новая связная область на плоскости, ограниченная образованным циклом. Вновь образованная область плоскости получает очередной номер(раскрашивается в новый цвет), все вершины, которые оказались внутри этой области получают метку(цвет) области, а все граничные вершины области добавляют к своим меткам метку новой области. Если среди меток вершин очередного ребра нет одинаковых, то ребро пропускается и не входит в суграф. Алгоритм заканчивает работу, когда все ребра будут просмотрены. Критерием задачи является число неуложенных ребер, которое должно быть минимальным.

На рис.11.3 приведен пример укладки графа с 10 ребрами. Максимальный фрагмент задан перестановкой (2,6,7,3,5,4,1,8,9,10). Каждой вершине приписаны цвета областей, с которыми она граничит.

Вопрос о достижимости в рамках приведенной фрагментарной модели остается открытым. Тем не менее, на основе приведенной фрагментарной модели легко построить эволюционную модель для поиска максимально плоского суграфа в графе и его укладки на плоскость.

11.2. Экстремальные задачи на взвешенных графах

Взвешенный граф задается как набор взвешенных ребер $\{(x, y, w(x, y))\}$. Здесь x,y – номера вершин, $w(x,y)$ – вес ребра (x,y), который может быть целым или, в общем случае, вещественным числом.

Задача коммивояжера

В задаче коммивояжера(ЗК) [54] требуется найти цикл в связном графе с минимально возможным весом, содержащий все вершины графа.

Рассмотрим два подхода к построению фрагментарной модели ЗК.

1. В качестве элементарных фрагментов в этой задаче можно рассматривать вершины(номера вершин) графа. В этом случае условие присоединения к списку вершин – рассматриваемая вершина отсутствует в списке; Фрагментарный алгоритм просто выбирает все вершины последовательно в соответствии с заданной перестановкой. Алгоритм заканчивает работу, когда число вершин во фрагмента совпало с числом вершин графа. Очевидно, что любое решение ЗК в такой постановке является достижимым. Значение целевой функции на перестановке вершин $s = (i_1, i_2, ..., i_n)$ вычисляется по формуле:

$$F(s) = w(i_1, i_2) + w(i_2, i_3) + ... + w(i_{n-1}, i_n) + w(i_n, i_1)$$

2. Ребра графа упорядочиваются $e_1, e_2, ..., e_m$. В качестве элементарных фрагментов рассматриваются номера ребер. Условие присоединения в этом случае – ребра набора не образуют циклов (за исключением цикла, содержащего все вершины). Фрагментарный алгоритм выбирает ребра в соответствие с заданной перестановкой ребер так, чтобы выполнялось условие присоединения. Алгоритм заканчивает работа, когда исчерпан список ребер или число выбранных ребер совпало с числом вершин графа. Любое решение ЗК в этой постановке

достижимо. Пусть в список s попали ребра $e_{i_1}, e_{i_2}, ..., e_{i_n}$. Тогда значение целевой функции определяется формулой

$$F(s) = w(e_{i_1}) + w(e_{i_2}) + ... + w(e_{i_n})$$

Некоторые замечания к предложенным моделям. Если граф не является полным, то он дополняется до полного и всем дополнительным ребрам приписывается вес M – большое положительное число. В этом случае значение целевой функции на перестановке фрагментов s будет иметь вид $F(s) = \alpha(s)M + \beta(s)$, где $\alpha(s), \beta(s)$ - вещественные числа. В рамках популяции перестановки упорядочиваются по правилу: s_1 предшествует s_2, если $\alpha(s_1) < \alpha(s_2)$ или $\alpha(s_1) = \alpha(s_2)$ и $\beta(s_1) \le \beta(s_2)$.

С помощью предложенных моделей можно решать задачу отыскания гамильтонова цикла в произвольном(не взвешенном) графе. Достаточно каждому ребру приписать некоторый(произвольный) вес.

Следует иметь ввиду, что при большом количестве одинаковых весов ребер графа, может существовать много оптимальных решений. В этом случае эволюционный алгоритм начинает работать неустойчиво. Поэтому при практической реализации модели рекомендуется слегка «возмущать» веса ребер малыми вещественными приращениями, чтобы исключить многоэкстремальность задачи и, тем самым, улучшить сходимость эволюционного алгоритма.

Задача покрытия графа ребрами

В задаче покрытия взвешенного графа ребрами следует выделить набор ребер графа минимального общего веса, который содержит все вершины графа. Пусть задан граф без изолированных вершин. Элементарными фрагментами во фрагментарной модели задачи являются ребра графа. Фрагментарный алгоритм работает следующим образом. Просматриваются ребра графа в порядке заданном перестановкой. На каждом шаге очередное ребро добавляется к списку ребер,

если хотя бы одна из его вершин не принадлежит множеству вершин ребер списка. Алгоритм заканчивает работу, если все вершины графа принадлежат множеству вершин ребер списка или список ребер исчерпан. Пусть выбрана перестановка *s* ребер графа и пусть на конечном шаге фрагментарного алгоритма список ребер имеет вид $\{e_1, e_2, ..., e_k\}$. Значение целевой функцией в данной модели на перестановке определяется формулой

$$F(s) = w(e_1) + w(e_2) + ... + w(e_k)$$

В предложенной модели графа достижимыми являются любые покрытия графа ребрами, в которых нет путей длины более чем 2. Таким образом оправдано применение эволюционно-фрагментарной модели для поиска покрытия минимального веса.

11.3 Задачи раскроя и упаковки

Задачи раскроя и упаковки – еще один класс задач, для которого удобно применять эволюционно фрагментарную модель.

Задача плоского прямоугольного раскроя.

Задан прямоугольник - основа размером $m \times n$, m,n - натуральные числа и набор прямоугольников $\{(a_i, b_i)\}_{i=1}^{N}$ с целочисленными сторонами. Задача разместить некоторые из прямоугольников набора на прямоугольнике основе без пересечения их внутренностей, заняв при этом максимально возможную площадь. Естественно, фрагментами в данной задаче [23] будем считать различные размещения прямоугольников семейства, то есть четверки вида (t_i, l_i, x_i, y_i), где (t_i, l_i) - координаты верхней левой вершины прямоугольника относительно основы. Условие присоединения - выбирается Top-Left размещение, то есть очередной прямоугольник набора размещается в верхнем левом углу еще не

занятой площади, так чтобы не пересекаться внутренностью с уже размещенными прямоугольниками. Очевидно, всякая перестановка прямоугольников набора задает определенный максимальный фрагмент - набор размещений, который называется картой раскроя. Конечно, не всякая допустимая карта раскроя может быть построена с помощью предложенного фрагментарного алгоритма. Таким образом, в рамках эволюционной модели можно отыскивать лишь приближенные решения задачи раскроя. В качестве критерия качества решения могут выбираться различные функции. В классическом случае – это плотность раскроя или общая площадь отходов при раскрое. Однако в рамках предложенной модели может решаться и задача о размещении с максимальной (минимальной) степенью симметрии.

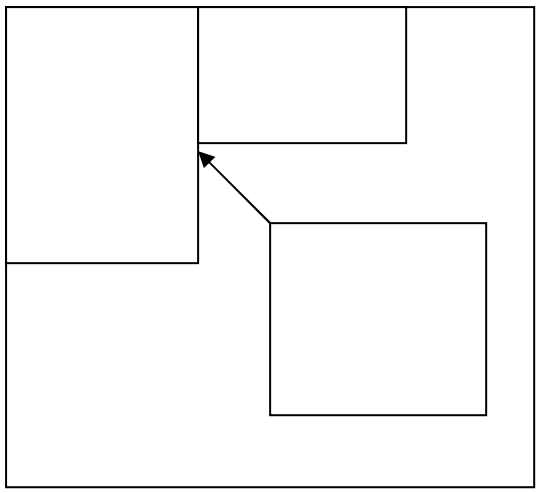


Рис. 11.4 Top-Left размещение

Игра «Пентамино»

Рассмотрим теперь интересную задачу, которая связана с известной игрой «Пентамино» [6]. Задача состоит в следующем: покрыть область плоскости, площадью 60 единиц, двенадцатью фигурами пентамино, составленными из 5 единичных квадратов.

В упрощенной переборной схеме фрагментарного алгоритма в качестве элементарных фрагментов участвуют 53 элемента, каждому из которых соответствует размещение одной из фигур пентамино на плоской целочисленной решетке. Примеры таких размещений приведены на рис. 11.5

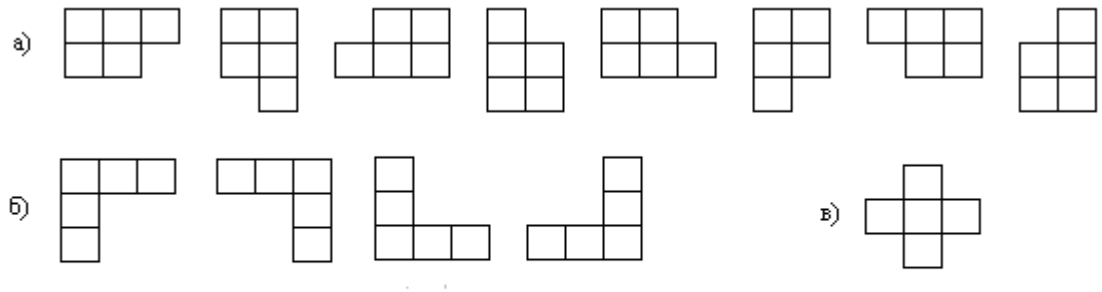


Рис.11.5 Различные размещения фигур пентамино
а) 8 элементов в группе симметрии; б) 4 элемента в группе симметрии;
в) 1 элемент в группе симметрии.

Решение задачи определяется перестановкой 53 элементарных фрагментов Условие присоединение в фрагментарном алгоритме – Top-Left размещение, причем присоединяемый фрагмент не должен совпадать с уже присоединенными с точностью до поворота на угол кратный 90^0. Примеры покрытия такого типа приведены на рис.11.6.

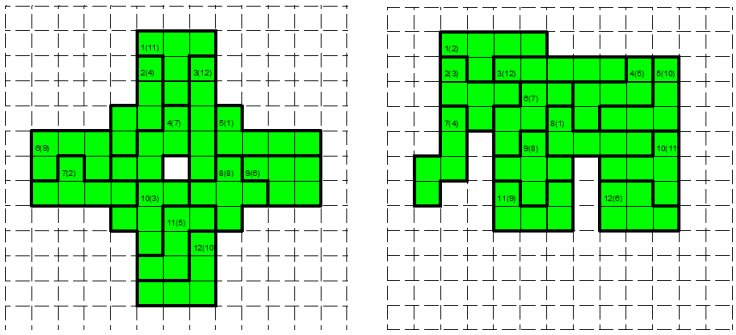


Рис 11.6 Возможные покрытия в задаче «Пентамино»

В качестве критерия в задачах этого типа можно брать плотность заполнения фигурами пентамино – рисунка-основы. Легко показать, что если существует покрытие 60-клеточного рисунка всеми 12-ю фигурами пентамино, то оно достижимо с помощью предложенного фрагментарного алгоритма.

Задача упаковки поликубов

Обобщением предыдущей задачи является задача упаковки многомерных объектов в контейнер заданного вида [28]. Пусть в качестве объектов упаковки берутся измеримые по Лебегу компактные односвязные множества в R^n.

Пусть X - измеримое по Лебегу компактное множество в R^n. В силу определения меры Лебега для всякого $\varepsilon>0$ существует масштаб измерения, в котором найдется поликуб Y такой, что:

1) $X \subseteq Y$;

2) объем поликуба Y отличается от объема множества X менее чем на ε.

Таким образом, упаковка сложных объектов может быть выполнена следующим алгоритмом: на начальном этапе заменяем объекты объемлющими их поликубами таким образом, чтобы суммарный объем поликубов отличался от объема пакуемых объектов не более чем на заданную величину ε. Далее, с помощью предложенной выше эволюционной модели размещаем поликубы в контейнере. На последнем этапе заменяем изометричные вложения поликубов изометричными вложениями исходных объектов (рис.11.7).

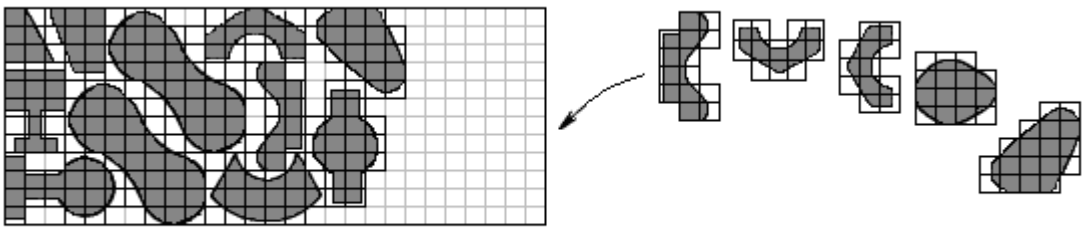


Рис. 11.7 Упаковка сложных объектов в контейнер

11.4 Задачи теории расписаний

Фрагментарный-эволюционный подход можно успешно применять для различных задач теории расписаний [69]

Задача Джонсона

Задача Джонсона [43] или задача многих станков формулируется следующим образом: набор из N деталей должен пройти обработку на M станках. Каждая деталь обрабатывается в строго определенной последовательности, одинаковой для всех деталей. Для каждой детали заданы продолжительности обработки этой детали на каждом из станков. В один и тот же момент времени на каждом станке может обрабатываться не более одной детали.

Каждая деталь в один и тот же момент времени может обрабатываться не более, чем на одном станке. Процедура обработки детали на каждом станке непрерывна. Задача - найти расписание (последовательность обработки деталей) минимальной общей продолжительности. В этой задаче в роли фрагмента выступает элемент последовательности номеров обрабатываемых деталей. Кодом решения является перестановка элементарных фрагментов. Вообще говоря, существуют допустимые и даже оптимальные решения, которые не могут быть закодированы подобным образом. Поэтому корректнее задачу дополнить условием: последовательность обработки деталей для каждого станка одна и та же.

11.5 Задачи булева программирования

Классическая задача о ранце

Задача о ранце – простейшая задача булева программирования [45] вида

$$c_1x_1 + c_2x_2 + ... + c_nx_n \to \max$$
$$a_1x_1 + a_2x_2 + ... + a_nx_n \le v$$
$$x_i \in \{0,1\}, i = 1,2,...,n$$

Здесь коэффициенты $c_i, a_i, v \quad i = 1,2,...,n$ - вещественные числа, причем $a_i \ge 0$ и $v > 0$.

Решение определяется последовательностью номеров переменных $i_1, i_2, ..., i_n$. Фрагментарный алгоритм на каждом шаге присваивает значение 1 очередной переменной в последовательности, если при этом не нарушаются ограничения задачи, и 0 в противном случае. Всякое оптимальное решение может быть построено таким путем и, следовательно, достижимо в рамках фрагментарной модели.

Следующий этап моделирования – стандартная эволюционная модель на перестановках.

Задача о рюкзаке

Совершенно аналогично решается и более общая задача о рюкзаке:

$$c_1x_1 + c_2x_2 + ... + c_nx_n \to \max$$
$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \le b_1$$
$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \le b_2$$
$$.....$$
$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \le b_m$$
$$x_i \in \{0,1\}, i = 1,2,...,n$$

Здесь коэффициенты $c_i, a_{ji}, b_j \quad i = 1,2,...,n; \quad j = 1,2,...,m$ - действительные числа, причем $a_{ji} \ge 0; \quad b_j \ge 0$.

Решение определяется последовательностью номеров переменных $i_1, i_2, ..., i_n$. Как и в предыдущем примере, фрагментарный алгоритм на каждом шаге присваивает значение 1 очередной переменной в последовательности, если при этом не нарушаются ограничения задачи. Любое оптимальное решение является достижимым.

Далее, как обычно, строится эволюционная модель на перестановках.

Приведенные примеры далеко не исчерпываю все богатство задач, для которых могут быть построена эволюционно-фрагментарная модель. Интересно, что и обычная задача линейного программирования также может быть рассмотрена как задача оптимизации на фрагментарной структуре. При этом фрагментами можно считать всевозможные наборы базисных переменных задачи. Общая задача оптимизации для функции многих переменных также может быть приближенно решена с использованием ЭВФ-алгоритма. Спектр задач, допускающий эволюционно-фрагментарную модель постоянно расширяется.

Таким образом, можно говорить об универсальности данного подхода по отношению к большому классу дискретных оптимизационных задач. А раз так, то имеет смысл создавать специализированное программное обеспечение именно для описания подобных моделей и отысканию приближенных решений задач в рамках эволюционно-фрагментарного моделирования.

12 СУБД для тестирования и оценки качества ЭВФ-алгоритмов

Одним из способов оценки качества эволюционных алгоритмов является тестирование на тестовых базах данных, отдельных задачах, случайно сгенерированных серий задач. С этой целью была разработана система тестирования ЭВФ-алгоритмов система создания баз тестовых примеров на основе стандартного пакета программ. В текущем разделе приводится подробное описание этой системы.

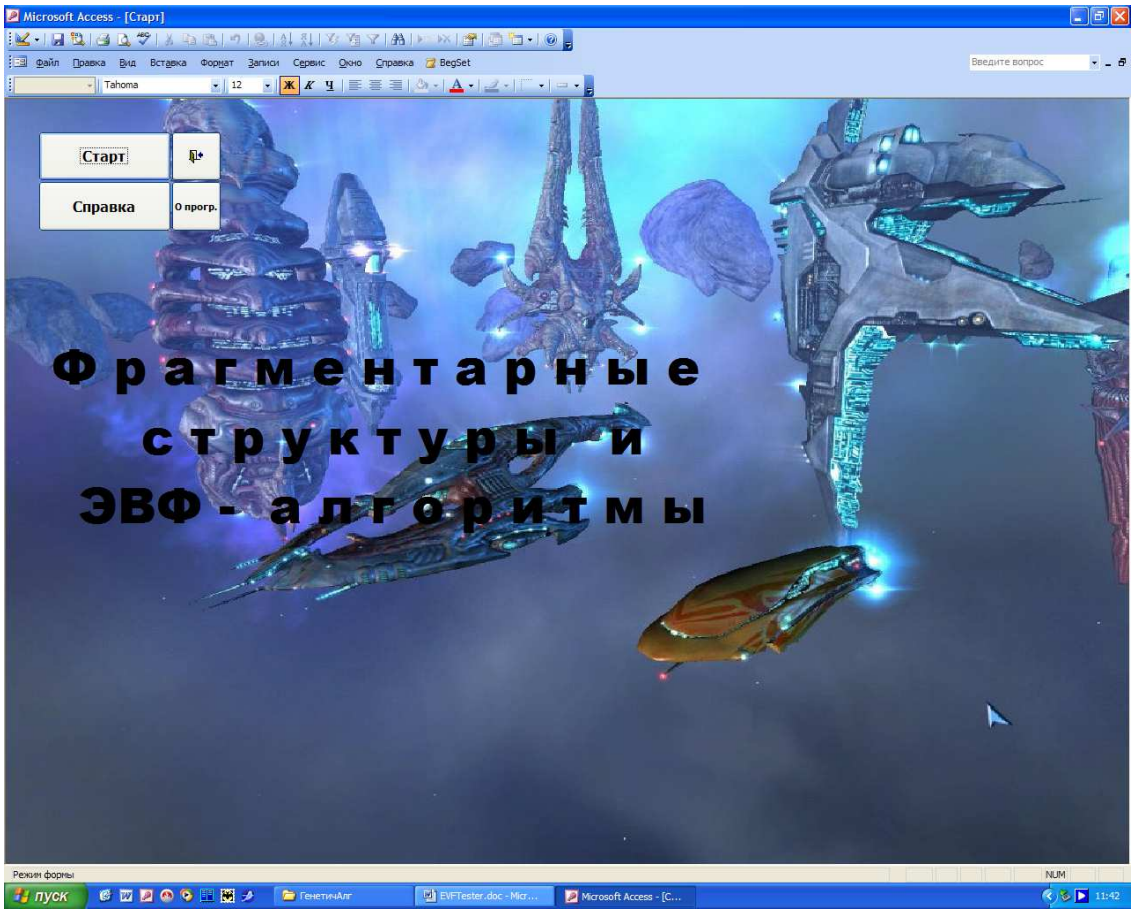


Рис.12.1 Программный комплекс для тестирования ЭВФ алгоритмов и поиска решений дискретних оптимизационных задач

12.1 Состав системы и требования к оборудованию

Компьютерная система «Фрагментарные структуры и ЭВФ-алгоритмы» предназначена для тестирования и оценки качества ЭВФ-алгоритмов. Система включает следующие файлы:

Файл основной программы-СУБД: EVFTester.mdb

Файл документации: EVFTester.DOC

Тестовые базы данных: файлы с расширением .mdb, в которых содержится серии задач. Количество таких файлов неограниченно.

Требования к оборудованию:

Процессор с частотой не ниже 2 мгц.

Объем оперативной памяти не менее 512 мб

Объем дисковой памяти не менее 50 Мб

Операционная система: WINDOWS XP/VISTA

СУБД: ACCESS XP 2002/2003/2007

12.2 Начало работы

Запуск системы осуществляется двойным щелчком левой кнопки мыши на файле EVFTester.mdb. В стартовом окне (рис.12.1) системы выбирается тип задач, которые будут рассматриваться.

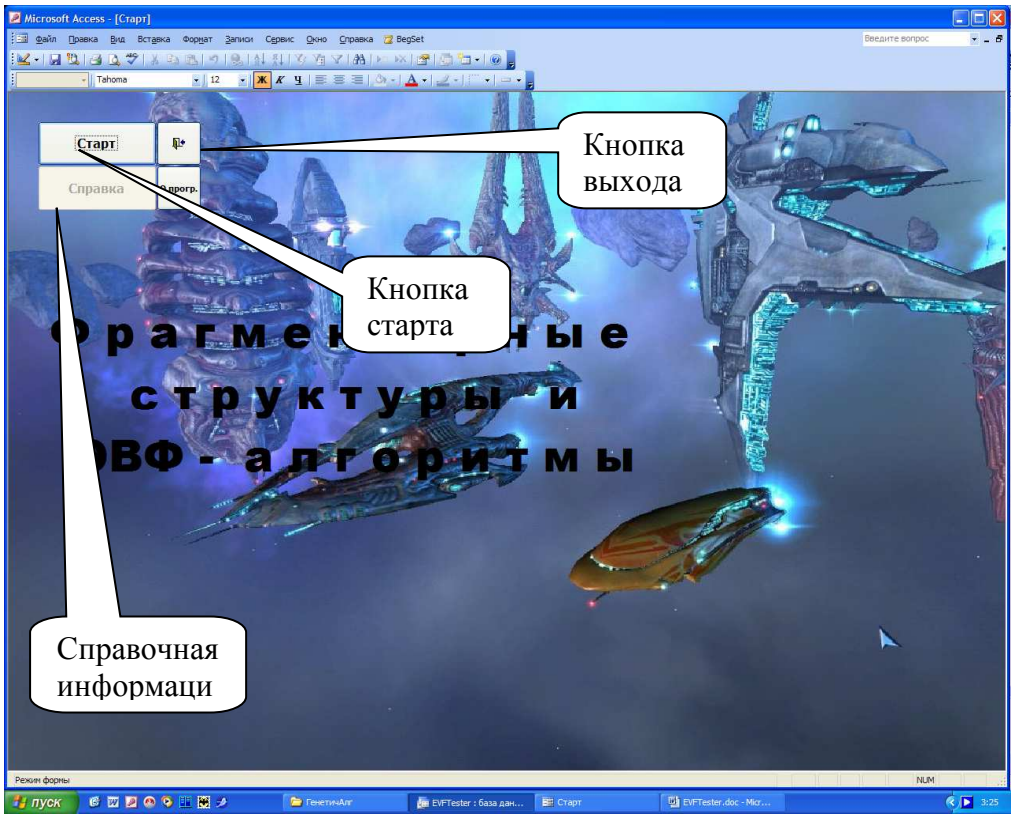


Рис.12.1 Стартовое окно системы

На сегодня в перечне типов присутствуют следующие типы задач:

1. Задачи ЦЛП

2. Задачи размещения блоков

3. Задачи на графах

4. Расписания

Однако этот список может пополняться.

Для начала работы системы необходимо нажать кнопку «Старт». Для окончания работы используется кнопка выхода или закрытие окна.

12.3 Главное окно программы

После нажатия кнопки старта открывается главное окно программы (рис.12.2). В этом окне представлена таблица с описаниями задач выбранного класса.

Каждый тип задач состоит из нескольких классов задач. Класс задач определяется набором параметров, который определяет индивидуальную задачу класса. Класс пожжет быть разбит на подклассы по некоторым признакам.

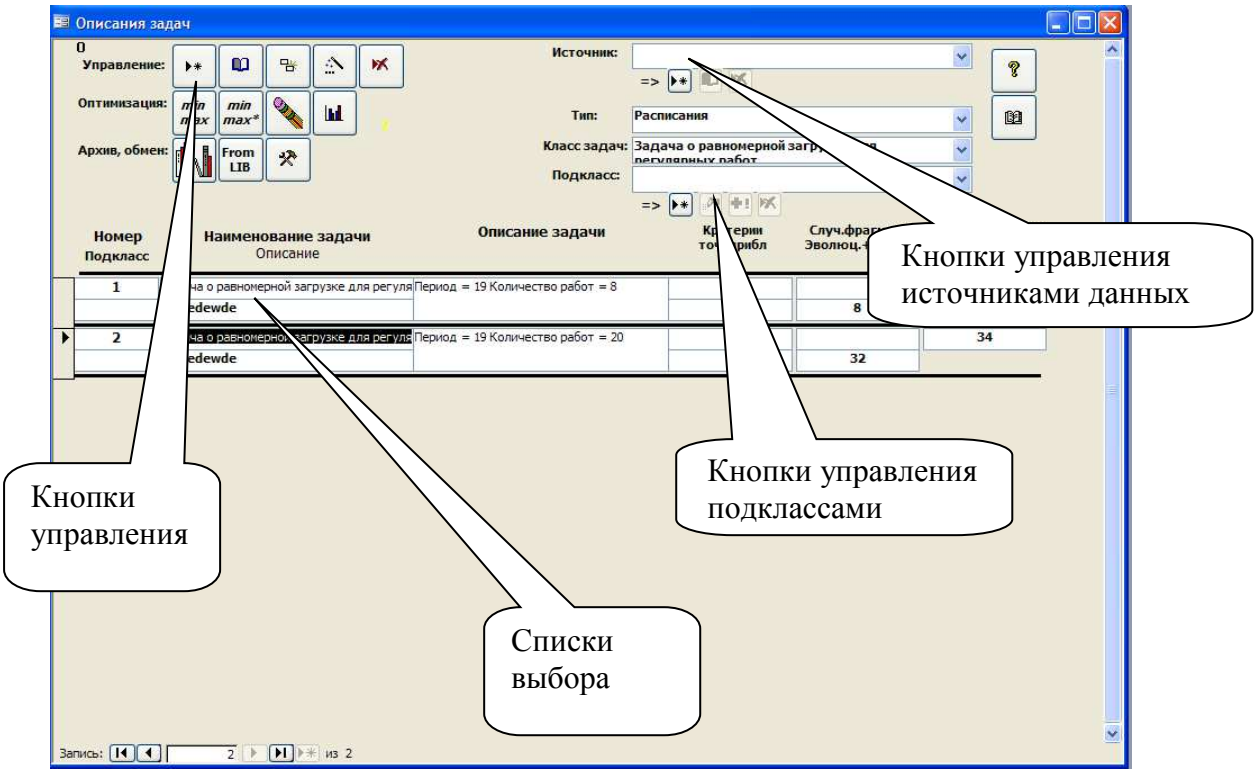


Рис. 12.2 Главное рабочее окно программы

Однако принадлежность задачи тому или иному подклассу может быть изменена. Каждая задача обязательно входит в некоторый класс, но может не входить в подклассы этого класса.

В заглавии формы расположены кнопки управления, списки выбора и титульная строка таблицы описаний задач.

Поле «Источник» указывает базу данных, в которой находятся рассматриваемые описания задач. Если это поле пусто, то в качестве базы используется сама база EVFTester.mdb. Рекомендуется проводить вычислительные эксперименты в отдельных базах-источниках. Впоследствии такие базы могут свободно распространяться и использоваться для исследований.

Кнопка  в заголовке формы позволяет получить справку по текущему окну задачи.

12.4. Работа с источниками. Кнопки управления источниками

Выбор источника из списка. Источник выбирается из списка источников по названию из списка источников (рис.12.3)

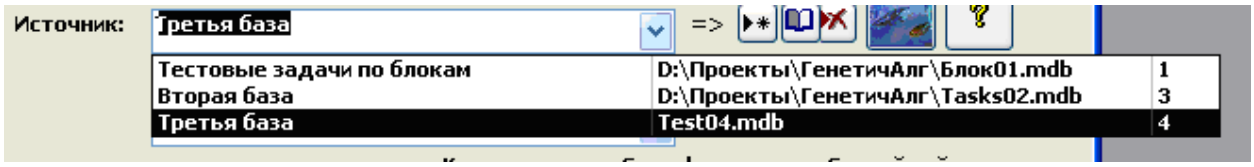


Рис.12.3 Описание источников.

В списке указаны наименование источника, полное имя файла базы данных и номер источника в списке. Если список пуст или нужный источник в списке отсутсвует, то необходимо создать новый элемент списка, нажав кнопку «добавить»  .

Двойной щелчок левой кнопки мыши на поле «Источник» очищает это поле и автоматически подключает данные текущей базы данных программы, то есть файл EVFTester.mdb.

Добавление описания источника в список. При нажатии кнопки добавления источника открывается окно описания источника (рис.4)

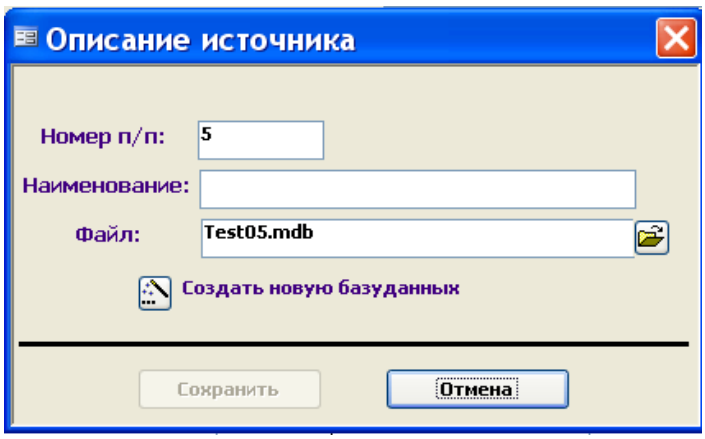


Рис.12.4 Окно описания источника

В этом окне последовательно вводится наименование источника – произвольный текст и указывается файл базы данных источника. Номер присваивается источнику автоматически. Поиск источника можно выполнить с помощью кнопки выбора источника . Если создается новый источник, то необходимо в поле «файл» ввести имя файла базы данных этого источника, а затем воспользоваться кнопкой мастера для создания новой базы данных-источника.

Чтобы поместить выбранный (или созданный) источник в список источников нужно нажать кнопку «сохранить». Для отказа от выбора достаточно нажать кнопку «отмена».

При создании новой базы - источника в перечень классов помещаются все классы существующие на текущий момент в базе данных программы.

Изменение описания источника в списке. Для того чтобы изменить описание источника в списке нужно нажать кнопку изменения данных об источнике, а затем в окне описания источника (рис.4) внести соответствующие изменения.

Удаление описания источника из списка. Для того чтобы удалить описание текущего источника из списка нужно нажать кнопку удаления описания источника и подтвердить удаление.

12.5. Выбор класса и подкласса задач

Выбор класса задач производится из списка классов данного типа задач в указанном источнике. Перечень описаний задач класса отображается в табличной части формы. Двойной щелчок левой кнопки мыши очищает поле «класс». При этом табличная часть формы исчезает.

По мере развития системы в список классов будут добавляться новые классы.. С каждым классом задач связан алгоритм описания задачи класса, алгоритмы отыскания решения и алгоритмы визуализации задач класса. Описание этих алгоритмов для каждого конкретного класса задач приводятся в отдельном разделе документации, посвященном этому классу.

Задачи одного класса могут быть сгруппированы в подклассы. Принцип группировки может быть произвольным. Например «задачи малой размерности», «задачи, для которых можно применять точный алгоритм» и т.д. Группировка задач произвольна и определяется пользователем. Задачи можно переносить из одного подкласса в другой. Можно объединять подклассы, удалять описания подклассов.

Выбор подкласса задач производится из списка подклассов данного класса в источнике. Перечень описаний задач подкласса отображается в табличной части формы. Двойной щелчок левой кнопки мыши очищает поле «подкласс». При этом в табличной части формы будут отображены все задачи класса.

Для того чтобы добавить название нового подкласса в список подклассов нужно нажать кнопку добавления подкласса  . Для изменения названия текущего подкласса можно применить кнопку  .

Для того, чтобы объединить текущий класс с другим существующим подклассом требуется нажать кнопку объединения подклассов  . При этом откроется диалоговое окно (рис.12.5) объединения подклассов.

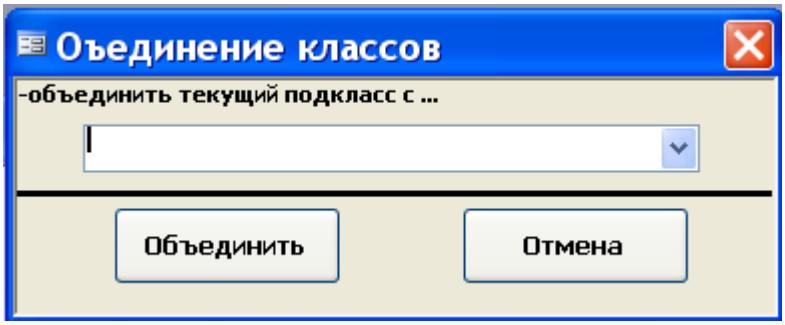


Рис.12.5 Окно объединения подклассов

Для выполнения объединения нужно указать подкласс, с которым происходит объединение, и нажать кнопку «объединить». Для отказа от операции используется кнопка «отмена». После того, как текущий подкласс будет объединен с выбранным подклассом, название подкласса будет удалено из списка подклассов, а в поле подкласс появится название подкласса, с которым произошло объединение. В табличной части формы будут выведены все задачи из объединения подклассов. Если в диалоге объединения не указать подкласс (оставить пустым поле выбора), то после объединения задачи не будут принадлежать никакому подклассу, а в таблице отобразятся все задачи текущего класса задач.

Удаление наименования подкласса из списка осуществляется кнопкой удаления  . После удаления подкласса все его задачи уже не принадлежат ни одному из подклассов.

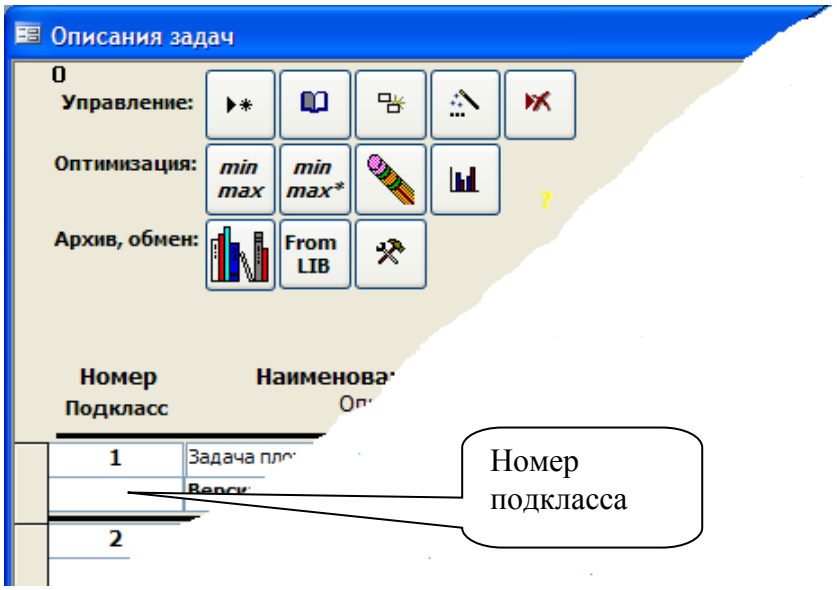


Рис.12.6. Номер подкласса

Номер подкласса отражается в табличной части формы (рис.12.6). Для изменения принадлежности подклассу индивидуальной задачи нужно установить курсор на поле номера подкласса в записи, соответствующей этой задаче и дважды щелкнуть левой кнопкой мыши. В результате откроется окно диалога рис.12.7, в котором можно изменить принадлежность задачи подклассу. Изменение подкласса в этом окне возможно лишь в том случае, когда подкласс не выбран из списка выбора подкласса. То есть в поле «подкласс» в заголовке формы значении е отсутствует. В противном случае выбор подкласса в диалоговом окне (рис.12.7) невозможен.

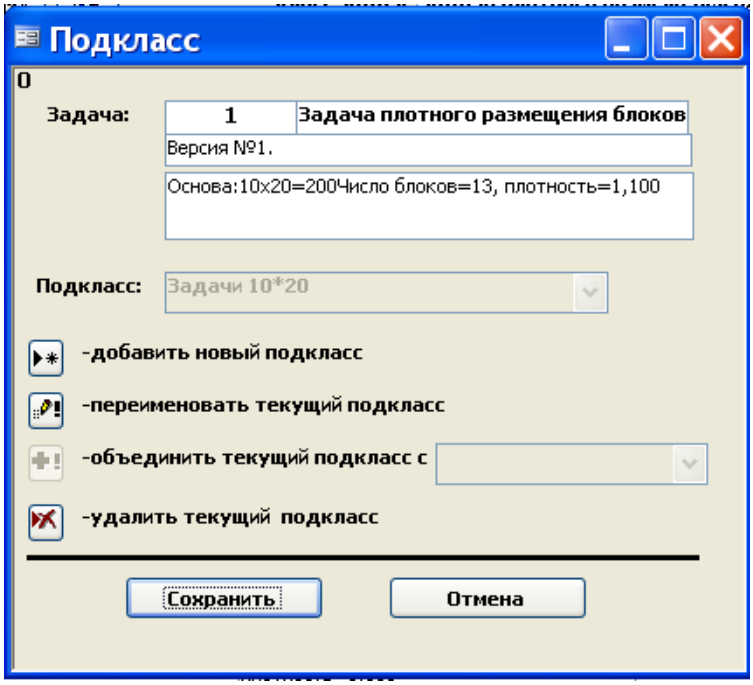


Рис.12.7 Окно работы с подклассами

В этом же диалоговом окне доступны все кнопки управления подклассами, то есть можно выполнить все операции (добавление, изменение, объединение, удаление), которые были описаны выше.

12.6 Табличная часть формы описания задач

В табличной части формы (рис.12.8) выводятся строки таблицы, содержащей перечень сгенерированных в системе задач.

Номер Подкласс	Наименование задачи Описание	Описание задачи	Критерии точ/прибл	Случ.фрагмент Эволюц.+фрагм.	Случайный поиск
1	Задача директора (задача одного станка)	Количество работ = 11	1556	2198	1636
1	Версия №1.			1615	
2	Задача директора (задача одного станка)	Количество работ = 91	153315	214719	193372
1	Версия №2.			188431	
3	Задача директора (задача одного станка)	Количество работ = 94	134087	205646	175898
1	Версия №3.			180697	
4	Задача директора (задача одного станка)	Количество работ = 54	57499	79919	68619
1	Версия №4.			67814	
5	Задача директора (задача одного станка)	Количество работ = 100	183017	267660	239875
1	Версия №5.			232042	
6	Задача дир ача одного станка)	Количество работ = 42	31057	41610	

Рис.12.8 Табличная часть окна описания задач

В первой колонке указываются порядковый номер задачи и номер подкласса. Вторая колонка содержит наименование задачи и ее краткое описание. Полное описание задачи приводится в третьей колонке. Следующие три колонки содержат последние результаты работы алгоритмов различных типов для выбранной задачи. Результат работы алгоритма – это вычисленное значение целевой функции. В программе используются следующие типы алгоритмов:

1) точный алгоритм;

2) известный приближенный алгоритм;

3) фрагментарный алгоритм при некотором упорядочении фрагментов;

4) ЭВФ-алгоритм;

5) алгоритм случайного поиска на множестве допустимых решений.

Особенности реализаций алгоритмов для различных классов задач описаны в отдельных файлах инструкции.

Чтобы более детально увидеть параметры последних расчетов можно воспользоваться кнопкой «min-max».

Перечень задач в таблице можно фильтровать и сортировать по обычным правилам работы с таблицами СУБД ACCESS.

12.7 Кнопки управления задачами

Перейдем теперь к описанию действий кнопок управления задачами, которые располагаются в правой части заголовка формы (рис.12.9)

Рис. 12.9 Кнопки управления задачами

Верхний ряд кнопок связан с формированием и редакцией описаний задач. В нижнем ряду находятся кнопки расчета, архива и анализа решений. Средний ряд – это управление поиском оптимальных решений

Кнопка 

Эта кнопка позволяет сформировать одну новую запись в таблице описаний задач. Кнопка вызывает окно описания новой задачи (рис.12.10),

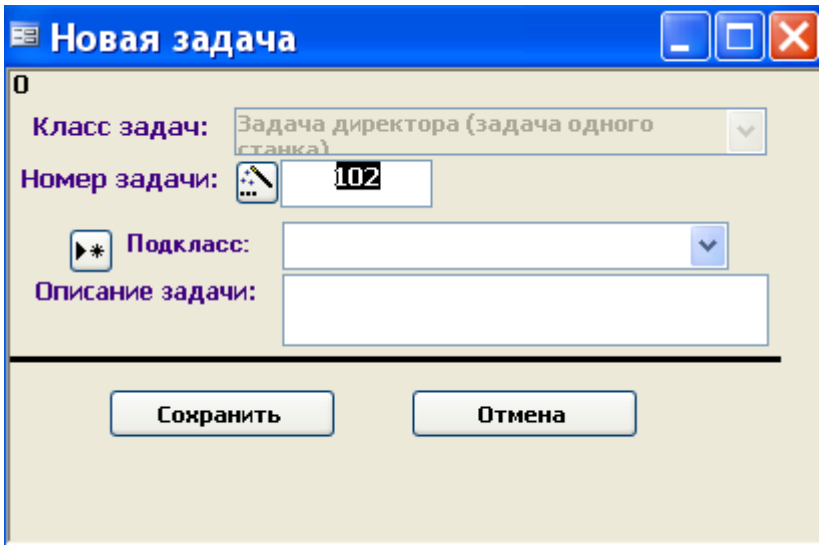


Рис.12.10 Окно ввода новой задачи

в котором вводятся подкласс задачи (если он не установлен apriory) и краткое описание задачи. Нажатие кнопка «сохранить» приводит к появлению новой записи в таблице описаний задач. Однако это лишь заглавие. Чтобы сформировать новую задачу с введенным заглавием нужно нажать кнопку описания параметров задачи.

Кнопка 

Эта кнопка открывает окно описания параметров задачи (генератор). Для каждого класса задач вид этого окна индивидуален. Например, для задачи коммивояжера окно описания параметров представлено на рис.12.11.

В окне задаются параметры генерации графа (связность, ориентация, число вершин и т.д.), затем при нажатии кнопки «генерировать» формируется описание графа с указанными параметрами генерации. Описание заносится в таблицу-описание, которая индивидуальна для каждого класса задач. Просмотреть и изменить таблицу-описание можно, нажав кнопку просмотра  . В этом же режиме можно ввести таблицу-описание вручную без генерации случайной задачи.

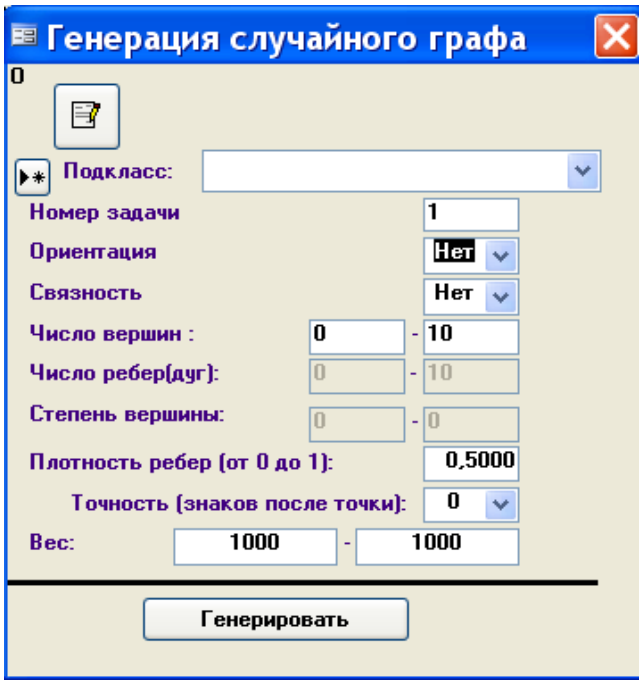


Рис.12.11 Окно генератора задач на графах

Если к задаче уже была применена процедура поиска решения, то изменить ее параметры нельзя. В этом случае все кнопки в окне описания параметров(рис.12.11) будут недоступны. Для того, чтобы все-таки отредактировать параметры задачи нужно отменить решение путем нажатия кнопки  .

Кнопка 

Эта кнопка позволяет создать полную копию текущей задачи (включая таблицу-описание). В окне дублирования (рис.12.12) вводятся краткое описание новой задачи. Можно также указать новый подкласс из уже существующих подклассов задач в данном классе.

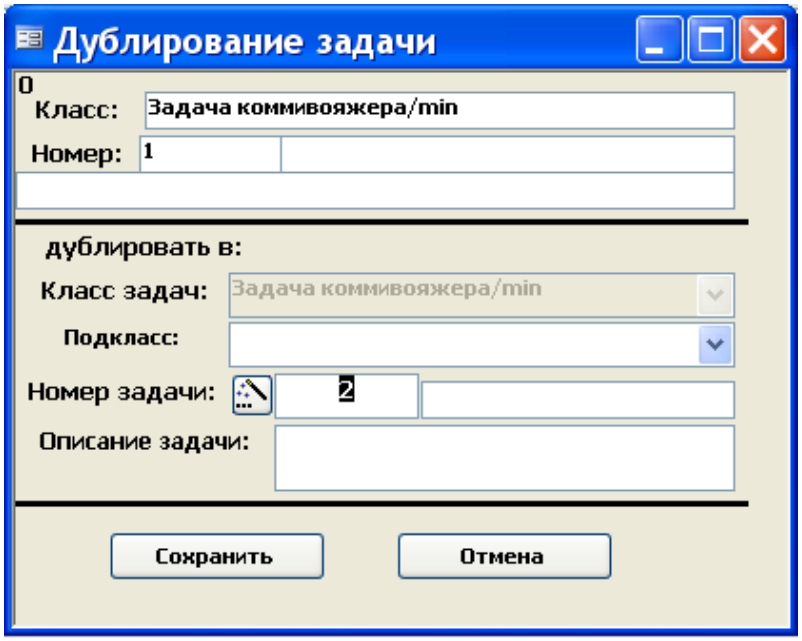


Рис.12.12 Окно дублирования задачи

Кнопка 

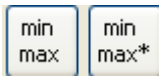
Кнопка открывает окно генерации серии случайных задач. Окно генерации позволяет ввести параметры описания задачи (см. выше). Кроме того, в этом окне наряду с параметрами генерации не обходимо указать количество задач серии. В результате нажатия кнопки «генерировать» будет построена серия случайных задач в соответствии с параметрами описания.

Разумно при генерации большой серии задач вводить новый подкласс, который будет содержать эту серию.

Кнопка 

Эта кнопка позволяет удалить одну или несколько задач. При нажатии кнопки предлагается удалить текущую задачу (ту на которую указывает указатель записи). Если кнопка удаления нажимается при удерживаемой клавише SHIFT, то

предлагается ввести диапазон номеров удаляемых задач. В результате будут удалены все описания всех задач класса из заданного диапазона номеров.

Кнопки 

Кнопки открывают окна поиска решений оптимизационной задачи (серии задач). Механизмы работы в этих окнах будет описаны ниже.

Кнопка 

Эта кнопка позволяет отменить результаты последних процедур поиска решения. В результате отмены параметры описания задачи станут доступными для редактирования.

Кнопка 

Нажатие кнопки открывает окно архива решений для выбранной задачи (та, на которую указывает указатель записи). Описание работы в этом окне приводится ниже.

Кнопка 

Кнопка позволяет вызывать тестовые задачи из известных тестовых библиотек(например OR[12]).

Кнопка 

Кнопка открывает окно анализа серии решений задач. В анализ попадают все задачи, которые представлены в табличной части формы (и только они). Работа этого окна описана ниже.

12.8 Поиск решения

Индивидуальный расчет

Для поиска оптимальных решений любой из задач перечня нужно перейти в окно поиска решений (рис.12.13).

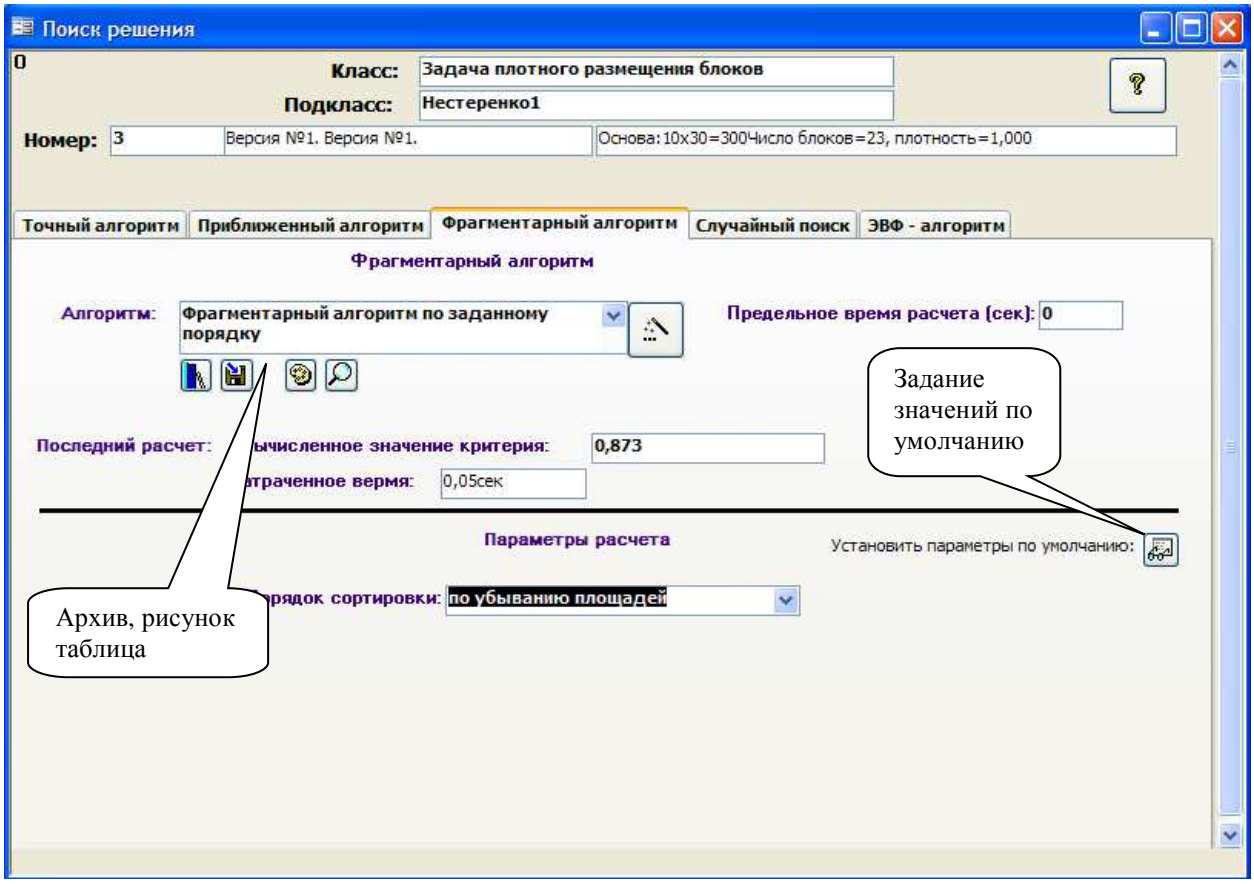


Рис.12.13 Окно поиска оптимального решения

Область данных формы «Поиск решения» разбита на пять зон, каждая из которых посвящено одному из типов алгоритмов для выбранного класса задач. Каждый тип алгоритмов имеет свой набор параметров расчета. Параметры расчета по умолчанию всем типам присваиваются нажатием кнопки  .

В любой из указанных зон выбирается из списка конкретный алгоритм решения задачи, устанавливаются параметры расчета, запускается процедура расчета путем нажатия кнопки расчета  для соответствующего алгоритма. В поле «вычисленное значение критерия» заносится результат расчета оптимального значения критерия.

Предельное время расчета устанавливается в секундах. Если предельное время расчета равно нулю, то расчет проводится без ограничения по времени. В противном случае алгоритм останавливается по истечению времени расчета.

Если время расчета исчерпано, то для процедуры точного алгоритма выполняется попытка в течение такого же промежутка времени найти приближенное оптимальное значение критерия. Если приближенное значение найдено, то оно выводится в поле значения критерия со знаком «*».

Для остальных типов алгоритмов по истечению времени расчета в поле значения критерия выносится значение, полученное на последнем шаге соответствующего алгоритма.

Значение критерия может иметь вид $\pm aM + b$, где M – большое положительное число. Наличия величины M в значении критерия показывает, что в описание решения входят несуществующие фрагменты.

Если процедура поиска оптимального решения закончилась неудачей, то в поле значения критерия заносится слово «нет».

Для алгоритма случайного поиска необходимо указать количество розыгрышей, то есть решений, который определяются случайным образом.

Для ЭВФ-алгоритма необходимо установить следующие параметры расчета:

а) размер(численность) популяции;

б) количество скрещиваемых пар в одном поколении;

в) количество поколений;

г) вероятность мутации;

д) коэффициент отбора.

Кнопка 

Эта кнопка позволяет сохранить найденное алгоритмом решение в архиве решений индивидуальной задачи.

Кнопка 

Кнопка выводит на экран/печать визуализацию решения. Для каждого класса задач визуализация имеет свою форму. Например, для задачи коммивояжера визуализация представлена на рис.12.14. Обращаем внимание, что использовать визуализацию разумно лишь для малых размерностей задач.

Кнопка 

Кнопка дает возможность представить найденное решение задачи в виде таблицы. Структура таблицы индивидуальна для каждого класса задач.

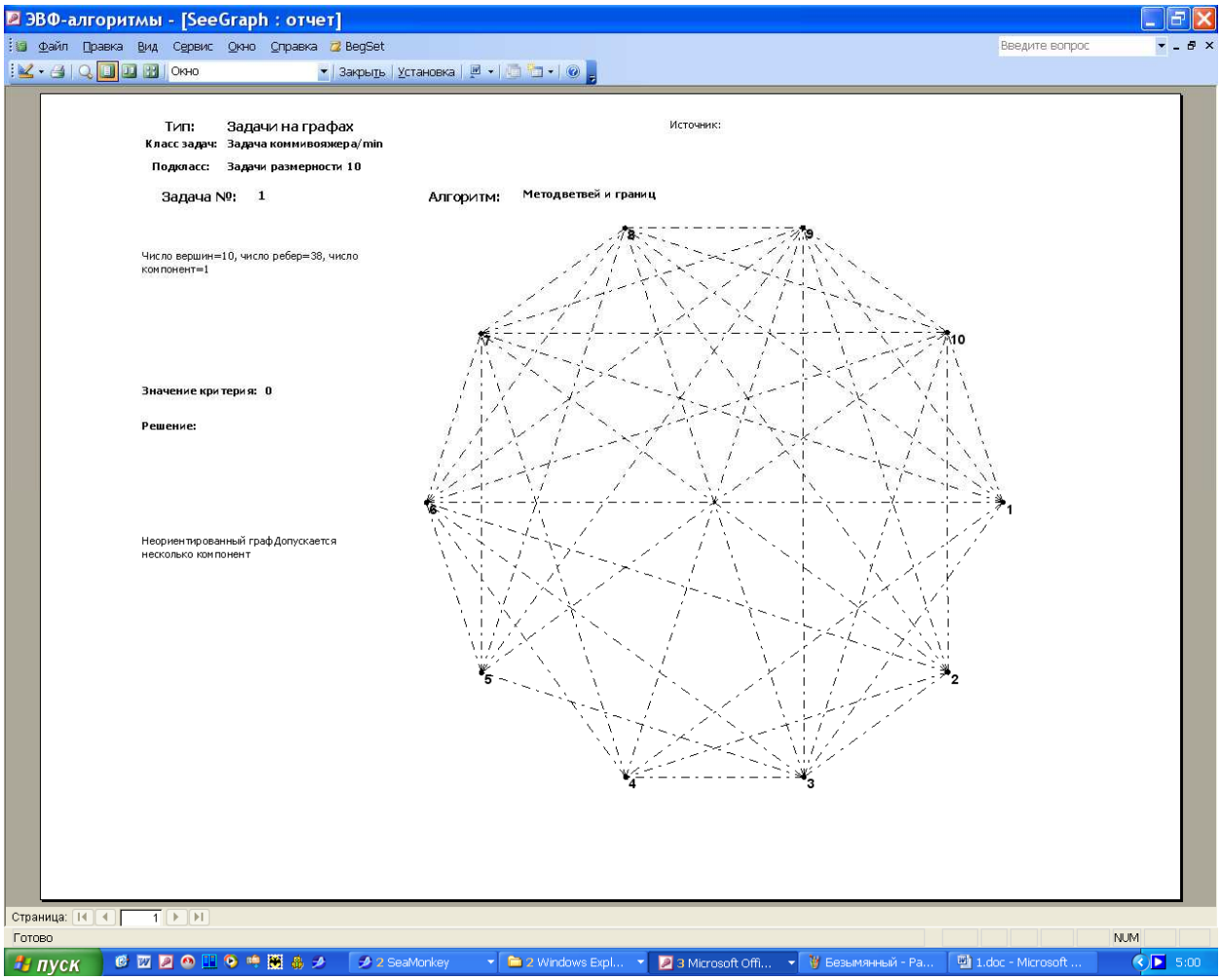


Рис.12.14 Визуализация решения

Групповой расчет

Для поиска оптимального решения сразу для группы задач нужно воспользоваться кнопкой группового поиска решений 

. В окне поиска решений (рис.12.15) необходимо выбрать алгоритмы расчета, установить параметры этих алгоритмов и нажать кнопку «выполнить расчеты». В результате все расчету будут последовательно проведены для всех задач, которые выведены в табличной части формы «Описания Задач».

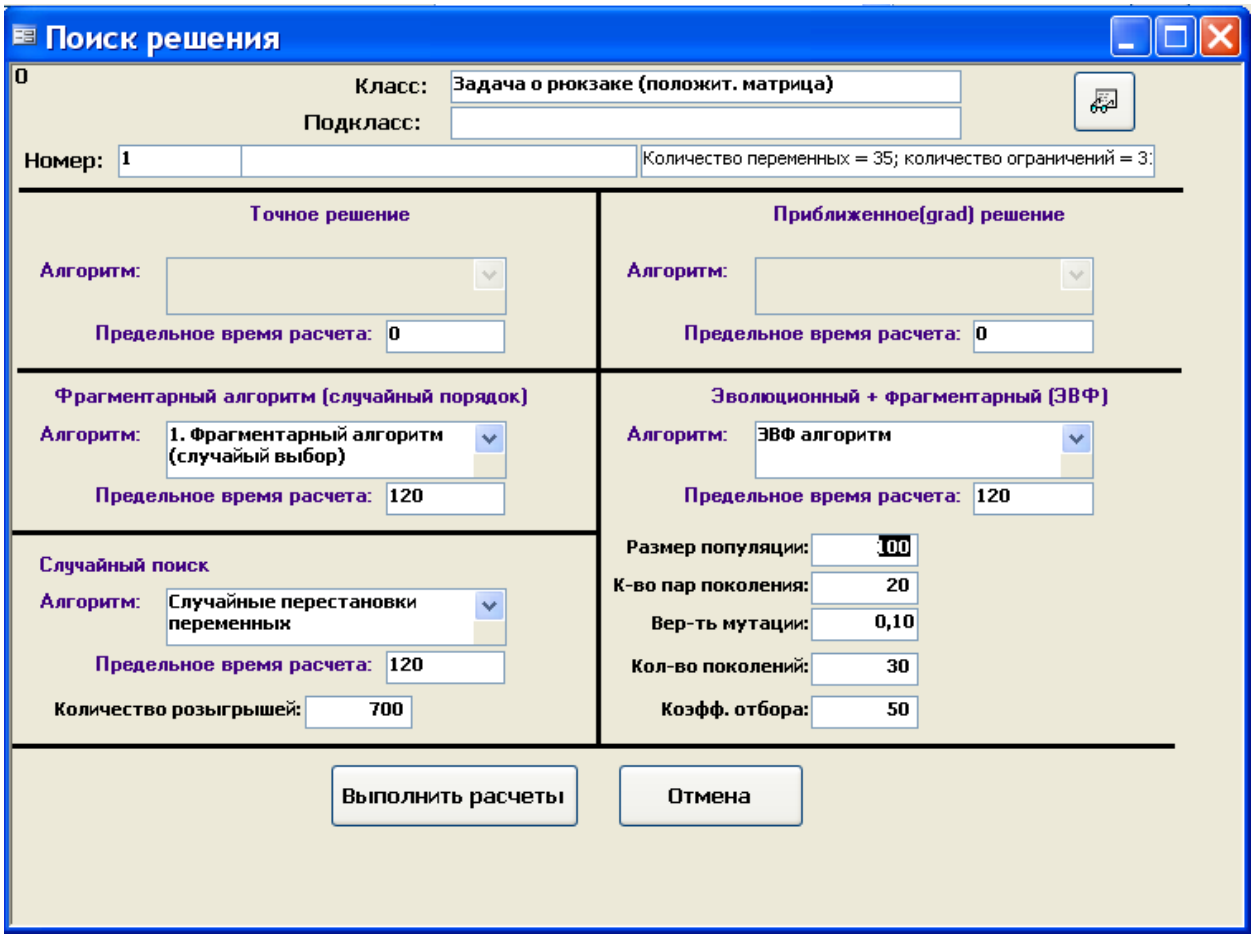


Рис.12.15 Окно группового расчета

12.9 Анализ результатов

Для того, чтобы получить сравнительные оценки качества алгоритмов на серии задач необходимо выполнить следующие действия:

а) выполнить расчеты для исследуемой серии задач по всем сравниваемым алгоритмам;

б) добиться того, чтобы в табличной части формы «Описания задач» присутствовали все задачи серии и только они. Этого можно достигнуть, выделив соответствующий подкласс и установив необходимые фильтры;

в) нажать кнопку анализа результатов  .

В результате откроется окно диаграмм, которые отражают сравнительное качество работы различных алгоритмов на заданной серии задач рис.16.

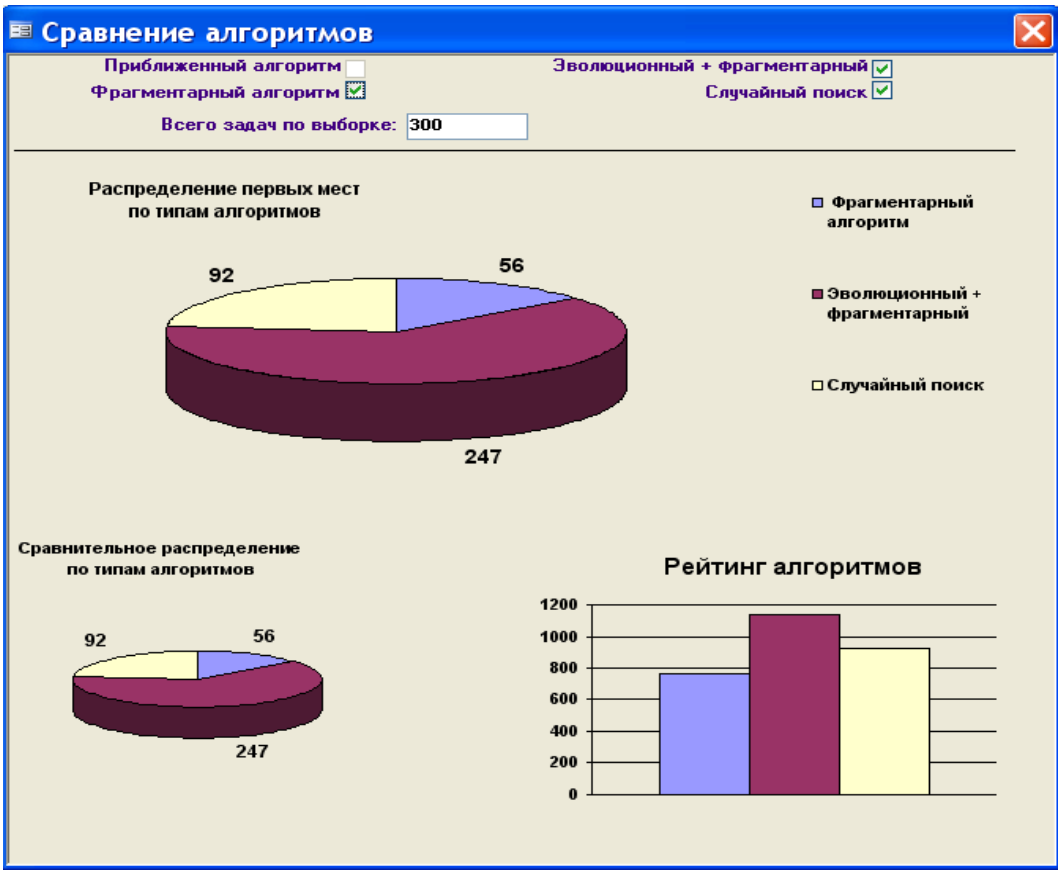


Рис.16 Окно анализа результатов

В заголовке окна выделяются типы алгоритмов для сравнения (типы алгоритмов, которые не описаны для данной серии задач, недоступны для отметки).

Распределение первых мест по типам алгоритмов: диаграмма показывает количество задач, для которых рассматриваемый тип алгоритмов приводил к наилучшим результатам среди всех применяемых алгоритмов в рассматриваемой серии задач. Эта диаграмма применима лишь при сравнении приближенных алгоритмов.

Сравнительное распределение по типам алгоритмов: число показывает сколько раз в серии задач результат, полученный по этому типу алгоритмов, был не хуже результатов, полученных по другим типам.

Рейтинг алгоритмов: вычисляется по правилу Борда, как сумма балов, набранных алгоритмом по всем задачам выборки. За первое место алгоритм получает 4 балла, за второе – 3 балла, за третье - два балла, за четвертое - 1 балл и за пятое место в сравнении – 0 баллов.

Предметный указатель

Литература

1. Банди Б. Методы Оптимизации. Вводный курс / Б.Банди. –М.: Радио и связь, 1988, - 128с.
2. Батищев Д.И. Генетические алгоритмы решения экстремальных задач / Д.И. Батищев. – Воронеж : Воронежский технический университет, 1995. - 65 с.
3. Береснев В.Л. Дискретные задачи размещения и полиномы от булевых переменных /В.Л.Береснев. – Новосибирск : Изд-во Ин-та математики, 2005, - 408 с.
4. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности. / Г.К.Вороновский, К.В.Махотило, С.Н.Петрашев, С.А.Сергеев – Х.: ОСНОВА, 1997. – 112 с.
5. Гладков Л.А. Генетические алгоритмы / Л.А.Гладков, В.В.Курейчик, В.М.Курейчик. – М. : Физматлит, 2006. – 319 с.
6. Голомб С.В. Полимино / С.В.Голомб. – М. : Мир, 1975. – 208 с.
7. Гуляницкий Л.Ф. Применение Н-метода для решения задач комбинаторной оптимизации на перестановках /Л.Ф.Гуляницкий, Д.А.Гобов // Системные исследования и информационные технологии. –2007. –№2. –С. 74–86.
8. Гуляницкий Л.Ф. Гибридная метаэвристика, основанная на оптимизации муравьиными колониями и Н-методе./ Л.Ф.Гуляницкий, С.И.Сиренко// Компьютерная математика. – 2009, –№ 1. –С. 142-151.
9. Гэри М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон; пер. с англ. –М. : Мир, 1982. – 416 с.
10. Деза Е.И. Энциклопедический словарь растояний. /Е.И.Деза, М.М.Деза. – М.: Наука, 2008. – 432 с.
11. Диксон Дж. Проектирование систем: изобретательство, анализ и принятие решений./Дж.Диксон. – М.: Мир, 1969, –440с.
12. Донец Г.А. Экстремальные покрытия графов /Г.А.Донец, А.Я.Петренюк. – Кіровоград : Комбінаторні конфігурації, 2009. –172 с.
13. Емеличев В.А. Лекции по теории графов / В.А.Емеличев, О.И.Мельников, В.И.Сарванов, Р.И.Тышкевич. – М.: Наука, 1990. –384с. (Изд.2, испр. М.: УРСС, 2009. 392 с.)
14. Емеличев В.А. О некоторых алгоритмических проблемах многокритериальной оптимизации на графах / В.А.Емеличев, В.А.Перепелица // Журн. вычисл. математики и мат. Физики. - 1989. - Т. 29, № 2. – С. 171–183.
15. Емельянов В. В. Теория и практика эволюционного моделирования./ В.В.Емельянов, В.В.Курейчик, В.М.Курейчик. – М. : Физматлит, 2003. – 432 с.
16. Еремеев А.В. Генетические алгоритмы для задачи о покрытии / А.В.Еремеев // Дискрет. анализ и исслед. операций. Сер. 2. – 2000. – Т. 7, № 1. –С. 47-60.

17. Зыков А. А. Основы теории графов. /А.А.Зыков. – М.: «Вузовская книга», 2004. – 664 с. — ISBN 5-9502-0057-8.(М.: Наука, 1987. 383с.)
18. Ильев В.П. Задачи на системах независимости, разрешимые жадным алгоритмом / В.П.Ильев // Дискрет. матем. – 2009. – Т. 21, № 4. – С.85–94.
19. Ильев В.П. Оценка точности алгоритма жадного спуска для задачи минимизации супермодулярной функции / В.П.Ильев // Дискрет. анализ и исслед. операций. Сер. 1. – 1998. – Т. 5, № 4. – С. 45-60.
20. Карпенко А.П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой : учебное пособие / А.П.Карпенко. – М : Издательство МГТУ им. Н. Э. Баумана, 2014. – 446 с.
21. Касьянов В.Н. Графы в программировании: обработка, визуализация и применение. Серия "Научное издание". / В.Н.Касьянов, В.А.Евстигнеев. – СПб.: БХВ-Петербург, 2003. – 1104 с
22. Ковалев М.М. Матроиды в дискретной оптимизации / М.М.Ковалев - 2-е изд., стер. – М.: Едиториал УРСС, 2003. – 224 с.
23. Козин И.В. Использование ЭВФ-алгоритмов для решения задачи прямоугольного раскроя / И.В.Козин, С.И.Полюга // Питання прикладної математики і математичного моделювання : зб. наук. праць / [ред. кол. … О. М. Кисельова (голов. ред.) та ін.]. – 2009. – С. 199-208.
24. Козин И.В. Фрагментарные структуры и эволюционные алгоритмы / И.В.Козин // Питання прикладної математики і математичного моделювання : зб. наук. праць / [ред. кол.: О. М. Кисельова (головний редактор) та ін.]. – 2008. – С. 138-146.
25. Козин И.В. Фрагментарный алгоритм для задачи симметричного размещения / И.В.Козин // Радиоэлектроника, информатика, управление. – 2005. – № 1. – С. 76-83.
26. Козин И.В. Отыскание множеств альтернатив многокритериальной задачи теории рас писаний при помощи эволюционного алгоритма / И.В.Козин, А.С.Бондаренко // Динамические системы. – 2010. – Вып. 28. – Сімферополь : КФТ. – С. 153-161.
27. Козин И.В. Об оценке мощности шарового покрытия пространства перестановок/ И.В.Козин, А.С.Бондаренко, С.И.Полюга // Вісник Запорізького національного університету. Фізико-математичні науки. – 2009. – № 1. – С. 134-138.
28. Козин И.В. Эволюционная модель упаковки многомерных объектов / И.В.Козин, С.И.Полюга // Вісник Запорізького національного університету. Математичне моделювання і прикладна механіка. – 2010. – № 1. – С. 61-67.
29. Кормен Т.Х. Алгоритмы : построение и анализ / Т.Х.Кормен, Ч.И.Лейзерсон, Р.Л.Ривест, К.Штайн - 2-е изд. – М. : Вильямс, 2005. – 1296с.
30. Краснощеков П.С. Принципы построения моделей. / П.С.Краснощеков, А.А.Петров - 2-е изд., пересмотр. и доп. – М.: ФАЗИС, 2000. – 412 с.

31. Курейчик В.М. Генетические алгоритмы. Состояние. Проблемы. Перспективы / В.М.Курейчик // Известия РАН. ТиСУ. – 1999. – №1. – С. 144-160.
32. Ларичев О.И. Системы поддержки принятия решений для слабоструктурированных проблем: требования и ограничения / О.И.Ларичев, А.Б.Петровский// Человеко-машинные процедуры принятия решений: сб. тр. – М.: ВНИИСИ, – 1988. – N 1. –С.4-13.
33. Ларичев О.И. Системы поддержки принятия решений. Современное состояние и перспективы развития / О.И.Ларичев, А.Б.Петровский // Итоги науки и техники. Сер. Техническая кибернетика. - М.: ВИНИТИ, – 1987. – Т.21. –С. 131-154.
34. Максишко Н.К. Моделі та методи розв’язання прикладних задач покриття на графах та гіперграфах / Н.К.Максишко, Т.В.Заховалко. – Запорожье: Полиграф, 2009. – 244 с.
35. Мухачева Э.А. Рациональный раскрой промышленных материалов. Применение АСУ / Э. А. Мухачева. – М. : Машиностроение, 1984 – 178 с.
36. Мухачева А.С. Генетический алгоритм поиска минимума в задачах двумерного гильотинного раскроя / А.С.Мухачева, А.В.Чиглинцев // Информационные технологии. Машиностроение. –М.: – 2001, №3.– С. 27 – 32.
37. Новожилова М.В. Моделирование и решение задачи размещения неориентированного объекта в области с переменными метрическими характеристиками / М.В.Новожилова // Проблемы бионики. – Харьков : ХГТУРЭ, –1998. – Вып. 49. – С. 95 – 98.
38. Перегудов Ф.И. Введение в системный анализ / Ф.И.Перегудов, Ф.П.Тарасенко. – М. : Высшая школа, 1989. – 367 с.
39. Перепелица В.А. Многокритериальные модели и методы для задач оптимизации на графах / В.А.Перепелица/ – Saarbrucken :LAP Lambert Academic Publishing GmbH& Co. KG, 2013. – 336 с.
40. Рассел С. Искусственный интеллект: современный подход = Artificial Intelligence: A Modern Approach (AIMA) / С.Рассел, П.Норвиг. - 2-е изд. – М.: Вильямс, 2006. – 1408 с. — ISBN 5-8459-0887-6.
41. Самарский А.А. Математическое моделирование / А.А.Самарский, А.П.Михайлов. – М. : Физматлит, – 2001. – 320с.
42. Сачков В.Н. Введение в комбинаторные методы дискретной математики / В.Н.Сачков. – М. : Наука, 1982. – 384 с.
43. Севастьянов С.В. Введение в теорию расписаний / С.В.Севастьянов. – Новосибирск : Новосибирский государственный университет, 2003. – 173 с.
44. Сергиенко И.В. Задачи дискретной оптимизации: проблемы, методы решения, исследования / И.В.Сергиенко, В.П.Шило - К.: Наукова думка. – 2003. – 264 с.
45. Сигал И.Х., А.П.Иванова. Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы. 2-е Изд./ И.Х.Сигал, А.Иванова. – М., Физматлит, 2007. –304 с.

46. Скобцов Ю.А. Основы эволюционных вычислений : учеб. пособ./ Ю.А.Скобцов – Донецк: [ДонНТУ], 2008. – 326 с.
47. Солтан В.П. Введение в аксиоматическую теорию выпуклости /В.П.Солтан. –Кишенев: Штиинца, 1984, – 224 с.
48. Стоян Ю.Г. Математические модели и оптимизационные методы геометрического проектирования / Ю.Г.Стоян, С. В Яковлев. – К. : Наук. думка, 1986. – 268 с.
49. Субботін С.О. Неітеративні, еволюційні та мультіагентні методи ситнезу нечіткологічних і нейромережних моделей. / С.О.Субботін, А.О.Олійник, О.О.Олійник.– Запоріжжя: [ЗНТУ], 2009. – 375 с.
50. Сурмин Ю.П. Теория систем и системный анализ: Учеб. пособие. /Ю.П.Сурмин – К.: МАУП, 2003. – 368 с.
51. Уилсон Р. Введение в теорию графов / Р.Уилсон. – М. : Мир, 1977. - 208 с.
52. Щербина О.А. Метаэвристические алгоритмы для задач комбинаторной оптимизации (обзор) / О.А.Щербина //Таврический вестник информатики и математики. – 2014. – № 1. –С. 56-72.
53. Beasley J.E. OR-library: distributing test problems by electronic mail / J.E.Beasley // J. Oper. Res. Soc. _ – 1990. Vol. 41, N 11. –P.1069-1072.
54. Bellmore M. The Traveling Salesman Problem: A Survey / M.Bellmore, G.Nemhauser // Operation Reasearch, – 1968 Vo.16 – P.538–558.
55. Bjorner A. Introduction to greedoids / A.Z.Bjorner, G.M.Ziegler // White N. Matroid Applications / Cambridge : Cambridge University Press, 1992. – P.284-357
56. Cochran J.K. A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines / Jeffery K. Cochran, Shwu-Min Horng, John W. Fowler //Computers & Operations Research. – 2003, Vol. 30, N. 7, –P. 1087-1102.
57. Deb K. Realcoded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems./ K.Deb, A.Kumar // Complex Systems, – 1995. – V 9(6), –P. 431--454.
58. Goldberg D.E. Genetic algorithms in search, optimization and machine learning./ D.E. Goldberg – Reading:Addison Wesley, Machine learning –1989.– V.3 –P. 95-99.
59. Goldberg D.E. Loci and the Traveling Salesman Problem / D.Goldberg, Jr.Alleles, R.Lingle. // Proceedings of the 1-st International Conference on Genetic Algorithms. – Hillsdale: L. Erlbaum, – 1985. – P. 154-159.
60. Graham R.L. On the history of the minimum spanning tree problem / R.L.Graham, P.Hell // Annals of the History of Computing. – 1985. – Vol. 7. No. 1. –P. 43-57.
61. Herrera F., Lozano M., Verdegay J.L. Tackling real-coded genetic algorithms: operators and tools for the behaviour analysis / F.Herrera, M.Lozano, J.L.Verdegay // Artificial Intelligence Review, – 1998. – Vol. 12, No. 4, – P. 265-319.

62. Herrera F., Lozano M., Sanchez A.M. Hybrid Crossover Operators for Real-Coded Genetic Algorithms: An Experimental Study / F.Herrera, M.Lozano, A.M.Sanchez // Soft Comput. – 2005. –V.9(4), –P. 280-298.
63. Hifi M. A hybrid algorithm for the two-dimensional layout problem: the cases of regular and irregular shapes / M. Hifi, R.M'Hallah // Internat. Transaction in Oper. Res. – 2003. – Vol. 10. – P. 195–216.
64. Holland J.H. Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence Текст . / J.H.Holland. — The MIT Press, Cambridge, 1992. –228p.
65. Korte B. Greedoids / B.Korte, L.Lovasz, R.Schrader. – Berlin : Springer-Verlag, – 1991. – 699p.
66. Korte B. Greedoids and Linear Objective Functions / B.Korte, L.Lovasz // SIAM Journal on Algebraic and Discrete Methods. – 1984. – Vol. 5, N. 2. – P. 229-238.
67. Korte B. Mathematical Structures Underlying Greedy Algorithms / B.Korte, L.Lovasz // Fundamentals of Computation Theory : Proceedings of the 1981 International FCT-Conference, Szeged, Hungaria, August 24-28 / F. Gecseg, editor. - Berlin [etc.] : Springer-Verlag, – 1981. - P. 205-209. - (Lecture Notes in Computer Science ; No. 117).
68. Koza J. R. Genetic Programming / J.R.Koza. - Cambridge : MIT Press, 1992 –819p.
69. Kozin I.V. Evolutionary fragmentary algorithm for permutation flow shop problem / I.V.Kozin, O.S.Bondarenko // Таврійський вісник інформатики та математики. – 2009. – № 2. – С. 47-51.
70. Kruskal J.B. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem / J.B.Kruskal // Proc. AMS. – 1956. – Vol 7. - No. 1. – P. 48–50.
71. Lawler E. L. Combinatorial Optimization: Networks and Matroids / E.L.Lawler. – New York : Holt, Rinehart, and Winston, 1976. – 374 p.
72. Mitchel M. An Introduction to Genetic Algorithms / M.Mitchel. - Cambridge : MIT Press, 1998. – 158 p.
73. Moraglio A Topological interpretation of crossover. / A.Moraglio, R. Poli //In Proceedings of the Genetic and Evolutionary Computation Conference, – 2004. – P 1377–1388.
74. Moraglio A Topological Crossover for the Permutation Representation / A.Moraglio, R. Poli // Intelligenza Artificiale, – 2011. volume 5, issue 1, –P. 49-70.
75. Moraglio A,Towards a geometric unification of evolutionary algorithms, a thesis submitted for the degree of doctor of philosophy department of computer science university of essex november – 2007
76. Moraglio A., Geometric PSO for the Sudoku Puzzle/A. Moraglio, // J. Tog, Genetic and Evolutionary Computation Conference, – 2007, –P. 118 - 125
77. Paul E. Black, "Gray code", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology.

31 August 2009. (accessed TODAY) Available from: http://www.nist.gov/dads/HTML/graycode.html

78. Ranjithan S.R. Constraint Method-Based Evolutionary Algorithm (CMEA) for Multiobjective Optimization / S.R.Ranjithan, S.K.Chetan, H.K.Dakshina //Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization. Springer Verlag, – 2001, –P. 299-313.
79. Rodzin S.I. Schemes of Evolution Strategies / S.I.Rodzin //Proceedings of the 2002 IEEE International Conference on Artificial Intelligence Systems (ICAIS). - Los Alamos : IEEE Comp. Society, – 2002. –P. 375-380.
80. Taillard E. Benchmarks for basic scheduling problems /E. Taillard // European Journal of Operational Research. – 1993. – Vol. 64, N. 2. – P. 278—285.
81. Whitney H. On the abstract properties of linear dependence / H.Whitney // American Journal of Mathematics. – 1935. – Vol. 57. No. 3. –P. 509-533.
82. Wright A. Genetic algorithms for real parameter optimization /A. Wright // Foundations of Genetic Algorithms, V. 1. – 1991. – P. 205-218.
83. Ziegler G.M. Oriented Matroids Today [Electronic resource] / G.M. Ziegler // The Electronic Journal of Combinatorics ; dynamic surveys. - Access mode : http://www.emis.ams.org/journals/EJC/Surveys/ds4.pdf.

Наукове видання

Козін Ігор Вікторович

ЕВОЛЮЦІЙНІ МОДЕЛІ В ДИСКРЕТНІЙ ОПТИМІЗАЦІЇ
(російською мовою)

ISBN