S. E. Mironov, A. Yu. Vasilev
Saint-Petersburg state electrotechnical university «LETI»

MATCHING OF CELLS – THE SUBSYSTEM OF THE AUTOMATION HIERARCHICAL PROCESS TOLERANT DESIGN OF CMOS VLSI MACROBLOCKS LAYOUT

The results of researches in the field of automation hierarchical process tolerant design of CMOS VLSI macroblocks are presented. Describes the developed software tools for the design topology of the macroblocks and the methodology hierarchical topology design using the graphical editor of the subsystem "Matching of Cells" are described.

Hierarchical layout design, layout compaction, process tolerant design, design automation

УДК 004.023;656.027

Е. В. Скакалина
Полтавский национальный технический университет им. Ю. Кондратюка

Использование генетических алгоритмов для решения задачи оптимизации транспортных перевозок

Разработан эвристический алгоритм на базе генетического алгоритма, позволяющий генерировать оптимальные расписания для организации транспортных перевозок. Алгоритм полностью удовлетворяет условиям решения логистических задач и имеет преимущество в скорости по сравнению с другими разновидностями классических генетических алгоритмов. Программная реализация выполнена в среде C#.

Эвристика, генетический алгоритм, транспортная задача, логистика, анализ, библиотека классов

Эффективность и качество функционирования транспортных комплексов (ТК) в значительной степени зависят от рациональной координации работы различных видов транспорта, оптимального перераспределения между ними объемов перевозок и формирования на этой основе необходимых управленческих решений. Решение этих и других научно-прикладных задач требует формирования соответствующих организационных и функциональных структур на региональном и государственном уровнях. Данные структуры должны взять на себя выполнение функций, связанных с организацией работы различных видов транспорта как по горизонтали (различные виды транспорта), так и по вертикали (производственный, региональный и государственный уровни). Эффективная организация в работе различных видов транспорта требует решения ряда актуальных научно-прикладных задач, основными среди которых являются:

– формирование государственного и региональных ТК как единых функциональных образований в структуре государственного управления транспортной системой;

– разработка математических моделей эффективного функционирования отдельных элементов ТК в современных рыночных условиях;

– разработка методов и алгоритмов оптимизации работы ТК;

– формирование информационных и функционально-организационных структур и систем организационного управления производственными процессами ТК;

– разработка методов прогнозирования производственно-хозяйственной и финансовой деятельности ТК;

– формирование рынка транспортно-экспедиционных услуг.

На данный момент единственный по своей технологической сути процесс перевозок грузов на отдельных уровнях и этапах своего жизненного цикла функционирования раздроблен, интересы отдельных субъектов не скоординированы и, в целом, ориентированы на экономическую кон-

фронтацию, т. е. отсутствует системное взаимодействие заказчиков и транспортников. Существующая система организации производства и управления не обеспечивает оптимального выбора экономико-организационных критериев и не ориентирует транспортников на высокую эффективность результатов их деятельности. Интеграционные процессы, которые происходят в ТК на данном этапе его функционирования, не могут обеспечить его инновационного развития и не отвечают существующим жестким требованиям по структурной и временной адекватности мировым логистическим процессам.

Организация консолидированного процесса перевозок должна обеспечить снижение затрат на процессы перевозок грузов в сравнении с вариантом самостоятельного выполнения полного комплекса логистических операций самим грузовладельцем. Необходимо обеспечить развитие транспортной инфраструктуры в пунктах непосредственного взаимодействия различных видов транспорта и создание в них многофункциональных ТК, а также информационных и логистических центров. Таким образом, проблема существенного улучшения организационного управления производством в ТК страны и регионов на современном этапе является актуальной и требует соответствующего решения на основе реализации современных технических, технологических и организационно-экономических мероприятий.

Одним из способов экономии ресурсов при транспортировке грузов является применение систем поддержки принятия решений (СППР) в области транспортной логистики. Разработка программных пакетов, комплексно решающих задачи этой предметной области, требует проведения серьезных научных исследований с целью получения эффективных алгоритмов, пригодных для применения в повседневной практике.

Интеллектуальная информационная система (ИИС), лежащая в основе любой СППР, должна иметь архитектуру приложения и дружественный интерфейс для удобства пользования.

ИИС должна иметь следующий функционал:

- Обеспечение информационной поддержки управления транспортными перевозками.
- Решение транспортной задачи различными методами.
- Универсальный класс для работы с генетическими алгоритмами.
- Нахождение оптимального решения для составления расписания транспортных перевозок.

Цель транспортной деятельности считается достигнутой при выполнении шести условий:

– необходимый товар;
– необходимого качества;
– в достаточном количестве;
– в определенное время;
– в условленном месте;
– при минимуме затрат.

К настоящему моменту известно достаточно много алгоритмов для решения задач маршрутизации транспорта (ЗМТ). Большей частью это – эвристические методы, используемые при наличии матрицы расстояний или информации о расположении вершин на плоскости. ЗМТ является NP-трудной задачей, поэтому наиболее интенсивно поиск ведется в направлении приближенных алгоритмов. Предлагались точные методы решения ЗМТ (например, метод ветвей и границ), но время вычислений при их применении растет слишком быстро. ЗМТ предполагает значительно большее количество вариантов решений для просмотра, чем ЗК, при одинаковом количестве вершин, в то время как применение метода ветвей и границ для решения ЗК уже затруднительно для наборов данных с 30 вершинами и больше.

Среди классических методов решения ЗМТ можно перечислить следующие [1]:

1. Конструктивные алгоритмы – алгоритм Кларка–Райта и его расширения, последовательный алгоритм вставки Моля–Джеймса, последовательный алгоритм вставки Кристофидеса–Мингоззи–Тосса.
2. Двухфазные классические алгоритмы.
3. Алгоритм заметания, алгоритм лепестков.
4. Алгоритм Фишера–Джекумера, алгоритм Брамела–Симчи–Леви.
5. Классические улучшающие алгоритмы.

Формальные математические модели принятия решений в настоящее время все более полно отражают сложность реальных практических проблем, что, с одной стороны, делает их более адекватными реальным системам, а с другой – приводит к необходимости решать все более сложные задачи оптимизации. Основные свойства реальных практических задач оптимизации – наличие многих критериев, существенных ограничений, разношкальных переменных и алгоритмическое задание функций – делают невозможным применение традиционных методов. Выходом из такой ситуации является использование адаптивных стохастических алгоритмов, успешно преодолевающих указанные трудности.

Одним из наиболее часто применяемых в такой обстановке подходов являются эволюционные алгоритмы, представляющие собой стохастические оптимизационные процедуры, имитирующие процессы естественной эволюции, в частности – генетические алгоритмы (ГА). Алгоритмическое задание функций и разношкальность переменных не представляют дополнительных трудностей для ГА, которые работают с бинарными представлениями решений и не требуют информации о свойствах целевых функций. Однако наличие многих критериев и ограничений затрудняет применение ГА в практических задачах, так как большинство подходов, предложенных в области эволюционной оптимизации, ориентированы только на одну проблему, т. е. либо на многокритериальность, либо на наличие ограничений. Подходы, сочетающие оба направления, встречаются редко, и их эффективность не всегда удовлетворительна [2].

Таким образом, совершенствование существующих и разработка новых эффективных адаптивных поисковых алгоритмов условной многокритериальной оптимизации является актуальной научной задачей.

Решение транспортной задачи с помощью генетических алгоритмов. ТЗ с ограничением по времени можно описать следующим образом [3]. Имеется некоторое количество автотранспорта, один склад (депо) и некоторое количество клиентов. Для каждого транспортного средства требуется составить маршрут, на протяжении которого транспортное средство посещает ряд клиентов (например, с целью доставки какого-либо груза). На маршрут каждого транспортного средства накладывается ряд ограничений. Основные ограничения представлены формулами:

$$\sum_{k\in V}\sum_{j\in N} X_{ij}^{k} = 1, \forall i \in C, \quad (1)$$

$$\sum_{i\in C} d_i \sum_{j\in N} X_{ij}^{k} \le q, \forall k \in V, \quad (2)$$

$$a_i \le S_i^k \le b_i, \forall i \in N, \forall k \in V. \quad (3)$$

Ограничение (1) полагает, что каждый клиент обслуживается только одним транспортным средством и только один раз. Переменные X_{ij}^{k} принимают значения {0, 1}, где 1 означает, что автомобиль движется от вершины i к вершине j, 0 – обратное. Верхним индексом k обозначается соответствующий автомобиль ($k \in V$, где V – количество идентичных автомобилей грузоподъемностью q). Ограничение (2) определяет, что транспортное средство не может обслужить больше клиентов, чем позволяет его грузоподъемность d_i; $i \in C$ – спрос соответствующего клиента, C – множество клиентов. Ограничение (3) – это ограничение по времени; прибытие автомобиля к клиенту должно быть в пределах временного окна, где S_i^k – время прибытия соответствующего автомобиля к определенному клиенту, а $[a_i, b_i]$ – промежуток времени, так называемое временное окно (time window), в течение которого должен быть обслужен клиент. Для данной задачи формулируются следующие цели (целевые функции): первичная цель – минимизировать общее количество транспортных средств, необходимых для обслуживания всех клиентов; вторичные – минимизировать общее время обслуживания всех клиентов и общее расстояние, пройденное всеми транспортными средствами.

Точные методы, основанные на методах направленного перебора, не позволяют эффективно решать данные задачи большой размерности. Эвристические и метаэвристические методы, к которым относятся генетические алгоритмы, дают в этом случае более приемлемые результаты.

Проектирование библиотеки классов генетических алгоритмов. Программная реализация библиотеки классов ГА реализована на платформе .NET Framework и оболочке разработки MS Visual Studio, язык программирования – *C#*.

Создана универсальная библиотека классов, которая реализует основные шаги ГА:

1. Создание популяции.
2. Отбор.
3. Скрещивание.
4. Мутация.

Создан базовый класс BaseSpecies. Все виды должны быть производными от базового класса BaseSpecies <TSpecies>, где TSpecies – конкретный тип производного вида:

```
abstract public class BaseSpecies<TSpecies>:
{
Comparable where
TSpecies: BaseSpecies<TSpecies>
{...}
static protected Cross (double x, double y)
{...}
static protected double Mutation (double val)
{...}
static protected double TestChromosomes()
{...}
}
```

Как видно из кода, BaseSpecies – это абстрактный класс, который содержит статические protected-методы для скрещивания и мутации. Разберем некоторые методы подробнее. Начнем с методов, которые необходимо реализовать в производных классах: static protected double TestChromosomes();.

Этот метод должен устанавливать внутреннюю защищенную булеву переменную m_Dead, которая показывает, подходят ли хромосомы из ограничений, которые на них накладывают (например, в данном случае, попадают ли значения хромосом в отрезок [–5.0; 5.0]). Если значения хромосом устраивают проектировщика, то m_Dead устанавливается в false, иначе – true. С этой переменной (получаемой через свойство Dead) популяция отвергает заведомо неудачные виды. Следующий метод, который необходимо реализовать в производном классе, – abstract public TSpecies Cross (TSpecies Species);.

Этот метод создает новый вид, скрещивая себя и другой вид, который передается как аргумент. Здесь нужно сделать некоторые пояснения. Обычно, если вид содержит несколько хромосом, скрещивают хромосомы «одного вида», т. е. в данном случае скрещиваем a0 себя (т. е. this) и a0 другого вида, аналогично скрещиваем a1 и a1, т. е. нет смысла скрещивать a0 себя и a1 другого класса. Смысл скрещивания заключается в том, что берем одну часть хромосомы себя и вторую часть другого вида и создаем из них новую хромосому, которая и будет удерживаться в новом виде. Часто хромосомы скрещивают побитово. Суть побитового скрещивания заключается в том, что сначала случайно выбираем точку разрыва (скрещивания) хромосомы, затем создаем новую хромосому, которая состоит из левой части первой хромосомы и правой второй. Пусть, например, есть две восьмибитовые хромосомы: 10110111 и 01110010. Случайно выбираем точку разрыва (отмечена символом '|'):

101 | 10111 и 011 | 10010

Отсюда можно получить две следующие хромосомы – 101 10010 и 011 10111. Также можно искать две и больше точек пересечения. Таким образом, необходим конструктор, который создает вид из хромосом. Для этого в класс BaseSpecies включены реализации известных статических методов для скрещивания хромосом некоторых типов (Double, Int64 и Int32). Разберем подробнее первые 2 метода. В данном случае есть две точки пересечения: первая – в середине слова, вторая – знак числа, т. е. сначала скрещиваются хромосомы побитово без учета знака, а потом также случайно выбирается знак от одной из хромосомы – public void Mutation().

Следующий шаг – мутация. Мутация действует на одну хромосому. В теории при мутации могут происходить любые изменения, но в данной реализации мутация меняет один бит в слове. Мутации происходят сравнительно редко. Например, часто на практике вероятность мутации устанавливают 5…10 %, но иногда стоит попробовать сделать ее побольше (примерно 30 %). Выбор хромосомы для мутации может бать случайным (как простейший вариант – генератор случайных чисел). Довольно часто в качестве мутации для мелких чисел лучше применять не написанную выше функцию, а просто генератор случайных чисел, который выдает случайное число в нужном интервале.

Реализация популяции (класс Population) – public class Population<TSpecies> where TSpecies:BaseSpecies<TSpecies>. Как видно, имеется generic-параметр, представляющий собой вид, производный от BaseSpecies <TSpecies>.

Начнем со свойств, которые нужно настроить перед запуском алгоритма:

• *MaxSize (Int32)* – максимальное число видов, которое может содержать популяция. Это тот размер, до которого «урезается» популяция после скрещивания. По умолчанию – 500.

• *CrossPossibility (Double)* – вероятность скрещивания. Она должна быть в интервале (0.0; 1.0]. Если это условие не выполняется, то реализуется исключение ArgumentOutOfRangeException. По умолчанию это значение установлено равным 0.95.

• *MutationPossibility (Double)* – вероятность мутации. Она должна быть в интервале [0.0; 1.0). Если это условие не выполняется, то реализуется исключение ArgumentOutOfRangeException.

В работе значение вероятности отбора принимаем равным 1.0. В противном случае возникла бы необходимость удаления худших видов из популяции согласно значениям некоторой вероятности. Но в любом случае виды, у которых хромосомы не попадают в нужный интервал или не удовлетворяют каким-либо другим установленным условиям (так называемые мертвые виды), надо удалять.

Рассмотрим публичные методы, к которые необходимо обращаться:

– *public void Add (TSpecies species*) – добавить новый вид в популяцию. Необходимое количество видов должно добавляться вручную. Перед началом работы алгоритма необходимо, чтобы в популяции было в наличии хотя бы 2 вида. Число видов в популяции может быть меньше, чем установленное значение MaxSize. Если после скрещивания размер популяции меньше MaxSize, то просто не удаляются худшие виды (даже мертвые);

– *void NextGeneration ()* – получить следующую популяцию. Работа этого метода может занимать достаточно много времени, поэтому лучше всего его вызывать из отдельного потока.

Общий план использования библиотеки.

Рассмотрим общий план шагов при использовании библиотеки:

• Создать класс вида, производный от BaseSpecies <TSpecies>, где в качестве аргумента обобщения в BaseSpecies передается имя самого класса вида.

• Внутри класса вида создать набор хромосом необходимых типов.

• Перегрузить абстрактный метод public TSpecies Cross (TSpecies Species), который должен создать новый вид. Как параметр метода выступает особь того же вида, с которым необходимо провести скрещивание.

• Перегрузить свойство (только get public double FinalFunc ()), которое возвращает значение целевой функции. Если целевая функция сложная, то ее значение удобно хранить в отдельной переменной, значение которой и возвращает это свойство, а сам расчет целевой функции осуществлять в конструкторе вида.

• Перегрузить метод public void Mutation ().

• Перегрузить метод public override void TestChromosomes (), который устанавливает значение переменной m_Dead в true, если данный вид нас сознательно не устраивает, например, если значение его хромосом попадает в заданные пределы, и в false в противном случае.

• Создать популяцию Population <TSpecies>, которая будет хранить особей нужного вида.

• Вызвать метод Reset () популяции для удаления всех видов, если они уже были в популяции.

• Через свойство MaxSize популяции установить максимальный размер популяции.

• Установить вероятность мутации через свойство MutationPossibility. Значение этого свойства должно лежать в пределах от 0 до 1. Значение по умолчанию установлено 0.1.

• Установить вероятность скрещивания через свойство CrossPossibility. Значение этого свойства должно лежать в пределах от 0 до 1, причем нулевая вероятность скрещивания не допускается. Значение по умолчанию установлено 0.95.

• Добавить в популяцию необходимое число особей соответствующего вида через метод void Add (TSpecies).

• Вызывать метод void NextGeneration () до тех пор, пока в популяции в этом будет необходимость, например, пока не будет достигнуто нужное значение допустимой погрешности.

• Через свойство TSpecies BestSpecies можно получить лучшую на данной итерации (в данном поколении) особь вида, в которой целевая функция меньше, чем у других.

Разработка программной реализации. Благодаря созданной библиотеке классов можно реализовать любые модификации генетических алгоритмов. Так, используем ее для реализации следующего алгоритма:

Шаг 1. Конструирование начальной популяции определенного размера (оператор инициализации).

Шаг 2. Выбор двух родительских хромосом из популяции (оператор селекции).

Шаг 3. Копирование выбранных хромосом и применение генетических операторов для создания новых хромосом (операторы кроссинговера, мутации).

Шаг 4. Отбор и последующее удаление хромосом из популяции для восстановления ее первоначального размера.

Шаг 5. До тех пор пока не пройдено заданное число шагов, возврат к шагу 2, иначе конец работы алгоритма.

Первый шаг – нахождение начальной популяции, для чего решим транспортную задачу методом «северо-западного угла» и методом «минимальных элементов». Исходные параметры : $A = \{320, 280, 250\}$. Matr $B = \{150, 140, 110, 230, 220\}$. В табл. 1 отображены исходные данные.

Таблица 1

	*B*1	*B*2	*B*3	*B*4	*B*5
*A*1	20	23	20	15	24
*A*2	29	15	16	19	29
*A*3	6	11	10	9	8

Формирование начальной популяции отображено в табл. 2.

Таблица 2

	*B*1	*B*2	*B*3	*B*4	*B*5
*A*1	N	N	N	230	N
*A*2	N	140	110	N	220
*A*3	150	N	N	N	N

После формирования выборки программный комплекс осуществляет кроссинговер и мутацию. Затем формируется оптимальное решение (табл. 3).

Таблица 3

	*B*1	*B*2	*B*3	*B*4	*B*5
*A*1	120	N	N	200	N
*A*2	N	140	110	30	N
*A*3	30	N	N	N	220

На основе предложенного алгоритмического обеспечения разработана программная среда, позволяющая решать практические задачи по принятию управленческих решений. Изучая проблему оптимизации логистических процессов для хозяйствующих объектов, можно сделать вывод, что генетические алгоритмы являются достаточно мощным математическим инструментом для решения широкого круга прикладных задач, в том числе и таких, которые трудно или вообще невозможно решить другими методами. Следующие функции реализованы в виде программного обеспечения:

– создание универсального класса для работы с генетическими алгоритмами;

– решение транспортной задачи различными методами;

– нахождение оптимального решения для составления расписания транспортных перевозок.

СПИСОК ЛИТЕРАТУРЫ

1. Пожидаев М. С., Костюк Ю. Л. Сбалансированная эвристика для решения задачи маршрутизации транспорта с учетом грузоподъемности // Вестн. ТГУ. УВТиИ. 2010. № 3. С. 65–72.

2. Гуменникова А. В., Ильина Т. Р. Гибридный алгоритм адаптивного поиска для решения задач условной многокритериальной оптимизации // Вестн. Сибирского гос. аэрокосмического ун-та им. акад. М. Ф. Решетнева. 2004. Вып. 5. С. 70–76.

3. Емельянова Т. С. Решение эталонных транспортных задач с кластерным расположением клиентов с использованием генетических алгоритмов // Нечеткие системы и мягкие вычисления (НСМВ-2008): сб. науч. тр. Второй Всерос. науч. конф. с междунар. участием, Ульяновск, 2008. Т. 1. С. 195–199.

E. V. Skakalina
Poltava national technical Yu. Kondratyuk university

THE USE OF GENETIC ALGORITHMS TO SOLVE THE OPTIMIZATION PROBLEM OF TRANSPORTATION

The heuristic algorithm based on the genetic algorithm to generate an optimal schedule for the organization of transportation is developed. The algorithm fully satisfies the conditions in solving logistical problems and has the advantage of speed in comparison with other varieties of classical genetic algorithms. The software implementation is done in the C# environment.

Heuristics, genetic algorithm, the transportation problem, logistics, analysis, class library