МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. М.В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 8

Под общей редакцией
чл.-корр. РАН Л.Н. Королева

МАКС ПРЕСС

МОСКВА – 2007

УДК 519.6+517.958
ББК 22.19
П75

Печатается по решению Редакционно-издательского совета факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова

П75 **Программные системы и инструменты:** Тематический сборник факультета ВМиК МГУ им. М.В. Ломоносова: № 8/ Под общ. ред. Л.Н. Королева. – М: Издательский отдел факультета ВМиК МГУ (лицензия ИД №05899 от 24.09.2001г.); МАКС Пресс, 2007. – 196 с.
ISBN 978-5-89407-322-4
ISBN 978-5-317-02186-3

В данный сборник включены научные работы и сообщения, связанные с общими вопросами программирования и информатики, касающиеся, в том числе дистанционного образования, исследований и разработок, связанных с анализом экспериментальных данных, имитационным моделированием систем реального времени, сетевой обработкой, а также с описанием некоторых инструментальных систем, которые могут оказаться полезными.

В этих публикациях нашли отражение исследования и разработки в области создания программных систем, выполненных учеными, аспирантами и студентами факультета. Большая часть результатов доложена на Ломоносовских Чтениях 2007 года.

Статьи сборника будут интересны студентам, аспирантам и специалистам в области разработки прикладных программных систем с использованием новых информационных технологий.

УДК 519.6+517.958
ББК 22.19

Издательский отдел
Факультета вычислительной математики и кибернетики
МГУ им. М.В. Ломоносова
Лицензия ИД N 05899 от 24.09.01 г.

119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус

Напечатано с готового оригинал-макета
в издательстве ООО "МАКС Пресс"
Лицензия ИД N 00510 от 01.12.99 г.
Подписано к печати 18.12.2007 г.
Формат 60x90 1/16. Усл.печ.л. 12,25. Тираж 100 экз. Заказ 657.

119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова, 2-й учебный корпус, 627 к.
Тел. 939-3890, 939-3891. Тел./Факс 939-3891.

ISBN 978-5-89407-322-4
ISBN 978-5-317-02186-3

СОДЕРЖАНИЕ

В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные в основном описаниям инструментальных программных систем, разработанных авторами публикаций, относящихся к методам дистанционного обучения, имитационному моделированию, а также методам обработки экспериментальных данных.

В нем также публикуются статьи и заметки общего характера, касающиеся разделов информатики, тесно связанных с программированием.

Редколлегия

Раздел I
Общие вопросы программирования и информатики

Г.Г.Рябов, В.А.Серов

Компьютерные комбинаторно-топологические построения и их преобразования

Введение

Последние три десятилетия в области компьютерного моделирования отмечены беспрецедентным расширением фронта исследований дискретных геометрико-топологических структур. Такие структуры являются основой формирования среды для применения численных методов самого различного характера (поэтому часто используется термин «вычислительная среда»). Большое влияние на эти работы оказывают выдающиеся достижения в математике XX века, непосредственно связанные с топологией, геометрией и алгеброй, широкое освещение которых дано в книге С.П.Новикова «Топология» [5].

Тем не менее представляется, что один из определяющих рычагов современного научно-технического прогресса — вычислительная математика и техника, еще недостаточно испытала на себе влияние столь фундаментальных достижений, несмотря на широкое использование сеточных методов и богатый багаж методов решения задач на графах.

В настоящее время в области использования компьютерных геометрико-топологических структур сошлись интересы таких областей, как теоретическая физика, компьютерное зрение, моделирование химического и биологического синтеза на молекулярном уровне, нанотехнологии и других областей. С другой стороны, насущные задачи развития самих компьютерных систем и управления ими (GRID-системы, системы клеточных автоматов и др.) требуют более глубокого понимания параллелизма и самоорганизации, во многом определяемых геометрико-топологическими структурами [18].

При всем разнообразии требований и особенностей в этой области следует отметить тенденции, имеющие общий характер:

1. Для больших научно-технических проектов (климатический прогноз, аэродинамика летательных аппаратов, томографические задачи, конструирование новых лекарств и т.д.) размеры сеток, кубических комплексов и других геометрически-топологических

структур в настоящее время достигают диапазона 10^8-10^9. Но уже сейчас актуален диапазон 10^{10}-10^{12}.

2. В самих структурах возрастает необходимость учета все большего числа особенностей геометрико-топологического характера.
3. В ряде случаев требуются оперативные (часто в процессе непосредственного компьютерного моделирования) преобразования структур с сохранением определенных инвариантов (в общем случае управлением изменениями некоторых характеристик).

Таким образом, создание и пополнение эффективных инструментальных средств на компьютерах, реализующих геометрико-топологические построения, с возможным последующим использованием методов и операций для аппаратной реализации, является одной из самых актуальных задач.

Основная цель данной статьи рассмотреть вариант методики создания таких инструментальных средств не с точки зрения максимального охвата возможных построений, а с точки зрения сочетания точной компьютерной интерпретации фундаментальных понятий и методов и использования свойств симметрии (в самом широком понимании) для организации представления, хранения и преобразований соответствующих данных в компьютере. Хотя предлагаемый инструментальный комплекс носит экспериментальный характер и принятые ограничения при построении моделей весьма значительны, однако узловые вопросы «вложения» в компьютер в первую очередь интерпретации генерации и преобразования геометрико-топологических структур на этом фоне будут выглядеть достаточно рельефно.

Близкие по идеологии работы

Работы в указанной области и соответствующие публикации исчисляются тысячами. Поэтому здесь мы ограничимся ссылками на работы, которые относятся к направлениям, получившим устойчивые названия «дискретная дифференциальная геометрия» и «двоичная геометрия и топология». К первому направлению относятся работы, проводимые во многих центрах, среди которых выделяются Калтех, МТИ, Лос-Аламос, ИММ РАН и др., традиционно ведущие центры в области математической физики. Ко второму направлению относятся работы, проводимые в десятках центров, организующим звеном которых является Университет в Окленде (Новая Зеландия). Их прежде всего характеризуют исследования геометрико-топологических форм изображений.

В целом излагаемый ниже подход ближе ко второму направлению и опирается на результаты следующих работ: аксиоматика *конечных топологий*, наиболее полно изложенная в работах В.Ковалевского [6], модели *глобальной полиэдризации*

(триангуляции), предложенные Ю.Кенмочи, А.Имийя [10], модели и алгоритмы сжатия (*скелетонирования*), разработанные Ж.Бертраном, М.Купри и др. [9], *метрические аппроксимации* на решеточных и клеточных структурах в трудах Г.Боргефорс и Р.Клетте [11,17], синтез и преобразования *триангулированных и клеточных структур* в трудах Б.Четверушкина и др. [15], *вариационные методы на тетраэдральных структурах* в работах М.Десбруна и др. [14], *звездные подразбиения и бизвездные преобразования* в книге В.Бухштабера и Т.Панова [13].

В целом фундаментальной основой в работе являлся труд Л.С.Понтрягина «Основы комбинаторной топологии» [1].

Неформальные положения

Создание удобного «материала» для конструирования на компьютере геометрико-топологических структур можно грубо сравнить с использованием различных материалов и инструментов для ваяния и зодчества. При этом физическая гибкость материала (глина, воск и т.п.) может и не быть определяющей при наличии специальных инструментов и навыков обработки более твердых материалов (гранит, мрамор). Так же и *кусочно-линейные структуры симплициальных комплексов*, несмотря на свою «негладкость», при наличии соответствующих инструментов преобразования и масштабируемости, могут оказаться весьма эффективными для компьютерного использования. Более того, такие построения можно сравнить с «кристаллическими», поскольку симплексы на целых точках (вершинах) и примитивных (без внутренних целых точек) ребрах можно рассматривать как строительные кристаллы, а комплексы, как блоки, «цементирующие» такие кристаллы-кирпичи. От компьютерного представления, хранения и обработки этих конечных элементов в связи с общим комбинаторным характером задач во многом зависит эффективность применения инструментальных средств.

Саму генерацию структуры можно рассматривать как частный случай последовательных преобразований, управляемых по определенным правилам и поэтому нужно четко описать критерии и элементы такого управления. В самом общем виде подразумевается управление преобразованиями с контролем над связностью и топологическими инвариантами (характеристика Эйлера-Пуанкаре и др.).

Часто важен учет определенных метрических соотношений в таких структурах для практических задач. Отсюда необходимость метрического инструмента в структурах-моделях невыпуклых фигур.

Также существенным является удобство операций с триангулированными структурами, связанными с огромным разнообразием прикладных задач.

Схему взаимодействия основных областей, привлекаемых к созданию инструментальных средств можно условно представить, как это показано ниже.

Рисунок 1.

Общую идеологию излагаемой инструментальной системы можно грубо представить следующим образом.

1. Комбинаторно-топологические построения реализуются на базе *симплициальных комплексов* с вершинами в *целых точках* и ребрами, соответствующими *примитивным векторам*. Производится предварительная полиэдризация (триангуляция) пространства.

Задание множества вершин на таком триангулированном пространстве определяет симплициальный геометрический комплекс (часто можно не задавать матрицы инцидентности, но они могут быть легко получены). При различной «окраске» вершин можно рассматривать несколько комплексов и их взаимное положение. Линейные преобразования на базе *унимодулярных матриц* комплексов

легко реализуются, не выводя за рамки множеств целых точек и примитивных векторов. Основными элементарными преобразованиями топологического характера комплексов являются *расширение* и *сжатие*, как присоединение или изъятие из комплекса вершины (а в нашей конструкции и соответствующих симплексов) с учетом связности и иных инвариантов. Основным (конечным) элементом анализа здесь является *звезда-комплекс* вершины-кандидата на присоединение или изъятие из комплекса. Предложен и реализован метод табличного хранения всех вариантов ответов на топологический анализ звезды-комплекса вершины. Управляемая последовательность по каким-либо критериям расширений (или сжатий) может быть во многих случаях (при отсутствии пересечений звезд-комплексов вершин) параллельно исполняемой.

2. Геометрия чисел. На базе целых точек **Zn** и примитивных ребер для каждой точки **Rn** строится решеточный веер, аналог построения веера Фарея для **R2** из примитивных ребер, соответствующих *несократимым дробям* в *последовательности Фарея*. [2,3]. На ребрах, порождающих веер, вводится метрика- для каждой пары целых точек расстояние между ними равно длине кратчайшего пути по ребрам веера (в случае преград этот путь обходит их). Разработан алгоритм построения веера (*веерная триангуляция*) по заданной относительной погрешности между длиной «реберной ломаной» и евклидовой длиной между целыми точками.

3. Алгебра. Во многих местах используются свойства линейных преобразований, в том числе и определяемых целочисленными унимодулярными матрицами (свойство сохранения объемов) [4].

Вариант генерации некоторого симплициального комплекса можно условно представить как последовательность следующих действий:

Задание (одного из набора) варианта триангуляции пространства (с определенным видом комплексов-звезд вершин). Задание комплексов-запретов (преград), которые не могут быть включены в генерируемый комплекс.

Задание «затравки» в виде некоторого начального комплекса. Задание функции управления расширением (сжатием), зависящей от метрических, объемных соотношений и режимов запрещения при этом склеек и разрывов или допущения их при определенной регламентации (например, по числу склеек). Остановка процесса таких расширений и сжатий может быть задана различными критериями, такими например как достижение определенного объема или попадания некоторых метрических соотношений в заданную область. Также возможна остановка процесса по достижению «расширяющегося комплекса» границ общей модели.

Цело-точечные триангуляции на плоскости

Этот раздел можно рассматривать как конструктивный и наглядный показ основных идей, развитие которых актуально для более высоких размерностей.

Пусть $S \in \mathbf{R2}$, $S=\{0<x<m;0<y<n\}$; $m, n \in \mathbf{Z}$. Вершинами триангуляции служат целые точки $\{\mathbf{x}\} \in \mathbf{Z2} \cap S$. Обозначим через Up — множество всех возможных ребер (отрезков), соединяющих целые точки и соответствующих примитивным векторам с максимальным модулем координат $\leq p$.

Вектор с целочисленными координатами является примитивным, когда модули его координат есть взаимно простые числа. Отсюда ребра (отрезки) из Up не содержат внутренних целых точек.

В дальнейшем в этом разделе рассматриваются триангуляции на множествах **Z2** и Up для ограниченного S и обозначаются как T(S,Up). Излагаются вопросы:

1. Генерация и преобразования триангуляций. Классификация триангуляций.
2. Триангуляции и симплициальные комплексы.
3. Преобразования симплициальных комплексов и управление преобразованиями.

Эти вопросы освещаются здесь под углом компьютерных процедур. Поэтому процесс триангуляции S может рассматриваться как последовательная процедура соединения пар целых точек (вершин триангуляции) из S ребрами (отрезками) из Up с соблюдением условий:

1. Не допускаются пересечения ребер.
2. Процедура заканчивается, когда нельзя дополнительно провести ни одного ребра из Up без пересечения уже проведенных.

Такая процедура всегда заканчивается, поскольку всегда есть возможность «добрать» концовку триангуляции на ребрах с длинами отрезков 1 и $\sqrt{2}$. На рисунке 2а показана триангуляция для S(m=4,n=5) на ребрах U_3.

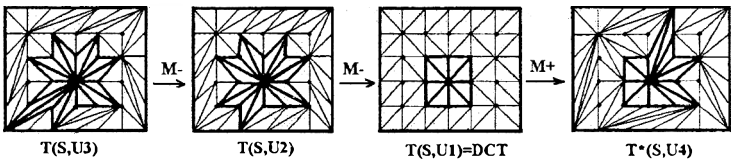

Рисунок 2. Триангуляция на плоскости (а) и M– и M+ перестройки (б,в,г.).

1. Рассмотрим два вида *перестроек* триангуляций- единичная *монотонно-убывающая* (М–) и единичная *монотонно-возрастающая*.(М+) [16].

М–: в конкретной T(S,Up) отыскивается ребро с максимальным p (оно является большей диагональю в параллелограмме, образованным ребрами со строго меньшим p) и заменяется на меньшее ребро (с меньшим p) в этом параллелограмме. Такой параллелограмм всегда существует в силу единственности представления несократимых правильных дробей как медиантной суммы соседних членов в последовательности Фарея [2,3].

Применяя последовательно такие М-преобразования, мы придем к $T(S,U_1)$,когда все ребра будут длиной 1 или $\sqrt{2}$ (см. рисунок 2в):

$$T(S,Up) \xrightarrow{M-} T(S,Up-1) \xrightarrow{M-} \dots \rightarrow T(S,U_1) \qquad (1)$$

Такие триангуляции T(S,U1) будем называть *диагонально-каноническими* (DCT), и каждой из них можно поставить в соответствие матрицу mxn из нулей и единиц. Нулем обозначено одно направление диагонали в каждом квадрате, а единицей — ортогональное ему направление диагонали. Такой матрицей для T (S,Up) на рисунке 2 будет:

1 0 0 0 0 0
0 1 1 0 1 0
0 1 1 0 0 1
0 0 0 1 1 1
0 0 0 0 0 1

Число различных таких матриц для S(mxn) равно 2^{mn}.

Поскольку любую T(S,Up) последовательностью М– преобразований можно однозначно привести к некоторой T(S,U1), то матрица также определяется однозначно. Все T(S,Up) с одной матрицей будем считать *одним классом*.

Единичная монотонно-возрастающая перестройка М+ в параллелограмме заменяет меньшую диагональ на большую. Применяя последовательно М+ преобразования допустим к T(S,U1) из (1), из-за неоднозначности выбора параллелограммов будет образовываться последовательность триангуляций, отличная от (1):

$$T(S,U1) \xrightarrow{M+} T^*(S,U2) \xrightarrow{M+} \dots \rightarrow T^*(S,Up^*) \qquad (2)$$

Процесс останавливается, когда в $T^*(S,Up^*)$ нет ни одного параллелограмма с ребром, меньшей диагональю этого параллелограмма. Такую триангуляцию считаем одной из тупиковых. Еще раз отметим, что эти преобразования не выводят из одного класса.

Таким образом , если требуется сгенерировать некоторую T(S,Up) (естественно с учетом допустимого заданного p) и заданного

класса с помощью заданной матрицы, то последовательность действий будет аналогична (2). Заметим, что классы обладают различными комбинаторными возможностями продвижения по цепочке M+ преобразований. Так если все строки и столбцы матрицы состоят из кодов 1010... или 0101..., то это уже тупиковая форма, не допускающая T(S,Up) с p>1.

2. Каждую T(S,Up)можно рассматривать как геометрический симплициальный комплекс и каждой внутренней точке $\mathbf{x} \in S$ соответствует *подкомплекс-звезда* star(**x**), который является множеством всех треугольников (с их вершинами и сторонами), имеющих эту общую вершину. На рисунке 2 показаны более жирными линиями звезды точки (3,2) при различных T(S,Up).

Среди множества всех диагонально канонических триангуляций (DCT) возможны только 6 (с учетом поворотов и зеркальных отображений) неконгруэнтных типов звезд, которые состоят соответственно из 4, 5, 6, 7 и 8 треугольников (см. рисунок 3). Заметим, что комбинаторно неэквивалентных 5 типов, поскольку 3-ий и 4-ый типы бизвездно эквивалентны. Ниже на рисунке 3 показаны двойственные им многоугольники.

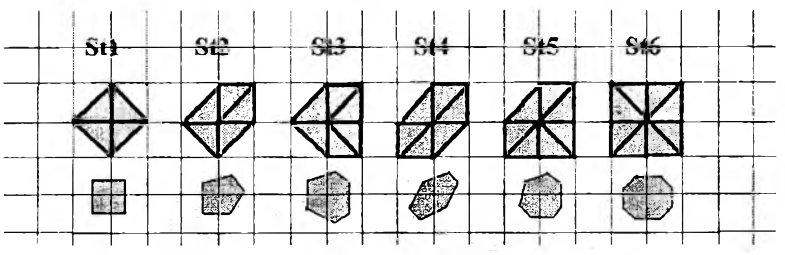

Рисунок 2. Шесть неконгруэнтных типов звезд в DCT.

Среди DCT отметим те, которые для всех внутренних точек имеют один и тот же тип звезды. Такие DCT реализуются на типах st3 или st4. Для st4 такую триангуляцию естественно назвать *транслируемой* (совмещение звезд при любом параллельном переносе на целочисленный вектор), а для st3 просто конгруэнтной. На вопросах других периодических DCT на парах и тройках типов звезд в данной статье мы останавливаться не будем.

На T(S,Up) ,как на комплексе можно задать подкомплекс, задав подмножество целых точек $S1 \in S$. В такой подкомплекс входят все ребра, инцидентные вершины которых в S1 и все треугольники, вершины и ребра которых вошли в подкомплекс. Целые точки из $S2=S \setminus S1$ также образуют подкомплекс. Обозначим соответствующие

подкомплексы как K1=K(S1) и K2=K(S2). Пусть K1 и K2 связные. Множество всех ребер Up можно представить как:

$$Up = U_1 \cup U_2 \cup U_3, \text{ где } U_1 \in K1, U_2 \in K2.$$

U3 — ребра, инцидентные вершины которых принадлежат разным подкомплексам. Такое множество ребер иногда называется водоразделом.

3. На образованной структуре можно задать единичные операции «расширения»(D+) и «сжатия»(C–). Расширение — добавление к множеству S1 одной целой точки **x** из S2 (расширение S1 и сжатие S2), такое ,что сохраняется связность комплексов K1*=K(S1 ∪ **x**) и K2*=K(S2 \ **x**). Аналогична операция сжатия [16].

Поскольку нарушение связности (склейка или разрыв) может произойти только в star(**x**) анализ корректности (в смысле сохранения связности) предполагаемого преобразования проводится только на элементах из star(**x**). Алгоритмы такого анализа можно найти в [16]. Здесь лишь подчеркнем, что число различных вариантов расположения вершин из K1 и K2 в star(**x**) равно 2^r, где r+1 — число вершин в звезде. Поэтому для r<30 все варианты можно заранее просчитать и ответы о возможности преобразования без склеек и разрывов таблично задать в оперативной памяти современных desktop-компьютеров, не говоря уже о кластерных суперкомпьютерах.

На рисунке 4 показаны примеры единичного расширения-сжатия и последовательности таких расширений-сжатий, приводящей к ситуации, когда расширение K1 (темная окраска) дальше невозможно без разрыва K2.

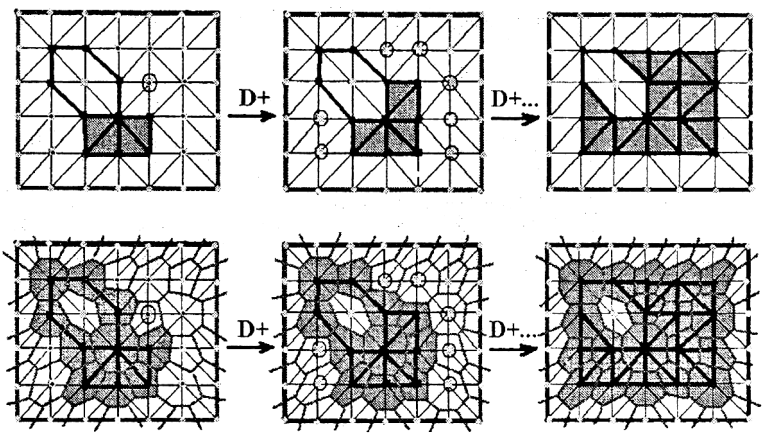

Рисунок 4. Результаты последовательного расширения «темного» комплекса (верхний ряд) и соответствующая картина в двойственных многоугольниках (нижний ряд).

Выбор точек расширения и сжатия может определяться различными целевыми функциями, как зависящими от общих геометрических и топологических свойств «конструируемого» комплекса (полиэдра), так и от «локальных» конфигураций соседних звезд. Целевые функции, опирающиеся на метрические соотношения для невыпуклых полиэдров, как правило, требуют более точной аппроксимации евклидовой метрики, нежели метрика на кратчайших путях, проходящих по ребрам триангуляций T(S,Up).

4. Общий подход к такого рода аппроксимациям состоит в рассмотрении более широкого набора примитивных векторов и построении на нем для каждой целой точки **Z2** решеточного веера. Для случая плоской решетки рассмотрим последовательности Фарея несократимых правильных дробей Φ(k). Каждой несократимой правильной дроби a/b поставим в соответствие вектор((0,0);(b,a)), который является примитивным с евклидовой длиной $\sqrt{a^2+b^2}$. Тогда для заданного Φ(k) и точки(0,0), векторы, соответствующие всем дробям из Φ(k) образуют в секторе $(0;\pi/4)$ одну восьмую часть *веера,* которая естественными симметриями отображается на $(0;2\pi)$, т.е. объединение всех секторов есть все **R2** и веер считается полным. Распространив такое построение для каждой целой точки, можно рассматривать его как неориентированный граф со взвешенными (евклидовой длиной) ребрами. Расстояние между двумя целыми точками определяется, как кратчайший путь по взвешенным (евклидовой длиной)порождающим веер ребрам. В [12] показано, как с увеличением порядка k последовательностей Фарея длина ломаной (кратчайшего пути по ребрам веера) между двумя целыми точками стремится к евклидовой длине. Такой метод позволяет определить расстояния между целыми точками, когда запрещен проход отрезка евклидовой прямой между ними при наличии преграды, которая отображается как дефект в графе, образованном веерами.

Более высокие размерности

Триангуляции и звезды. Роль диагонально-канонической триангуляции как исходной структуры, способной порождать процедурами перестроек, расширений и сжатий широкий набор триангуляций для случая **R2**, побуждает к попыткам экстраполяции этого подхода для более высоких размерностей. Однако уже для **R3** возникают особенности, на которых следует остановиться.

Для **R2** звезду вершины в DCT, как подкомплекс, можно рассматривать как объединение (стыковку) треугольников, образованных проведенными диагоналями в смежных квадратах. При этом для правильности расположения симплексов в комплексе требуется только правильное примыкание квадратов, имеющих общей эту точку.

Для **R3** кубы вокруг вершины звезды (как октанты в **R3**) , поделенные диагоналями триангуляции, должны не только быть сами правильно триангулированы и не просто примыкать друг к другу квадратными гранями, но и иметь в них совпадающие диагонали.

Рассмотрим для единичного куба некоторый вариант проведения диагоналей в его гранях. Тогда каждой вершине поставим в соответствие число диагоналей, инцидентное ей . Такое число может принимать значения i=0,1,2,3. Если обозначить количества вершин с соответствующими значениями через z0, z1, z2, z3, то для них должны быть выполнены следующие *диофантовы уравнения*:

$$\Sigma\, zi = 8;\ \text{(число вершин)}$$

$$\Sigma\, i\, zi = 12;\ \text{(число граней x 2)}$$

Общее множество решений этой системы {(0,6,0,2), (2,0,6,0), (1,3,3,1), (2,2,2,2), (0,4,4,0), (4,0,0,4)}. Для всех решений, кроме (0,4,4,0), существуют неконгруэнтные триангуляции куба. На рисунке 5 показаны все 5 типов неконгруэнтных триангуляций, четыре из которых являются комбинаторно эквивалентными (6 составных тетраэдров, каждый объемом 1/6) и одна из 5 тетраэдров(4 тетраэдра объемом 1/6 и один1/3).

Любая звезда в DCT **R3** может быть правильным объединением только этих типов. Ряд полиэдров, соответствующих звездам триангуляций куба, представлен на рисунке 5. Среди них 3-ий и 4-ый невыпуклые.

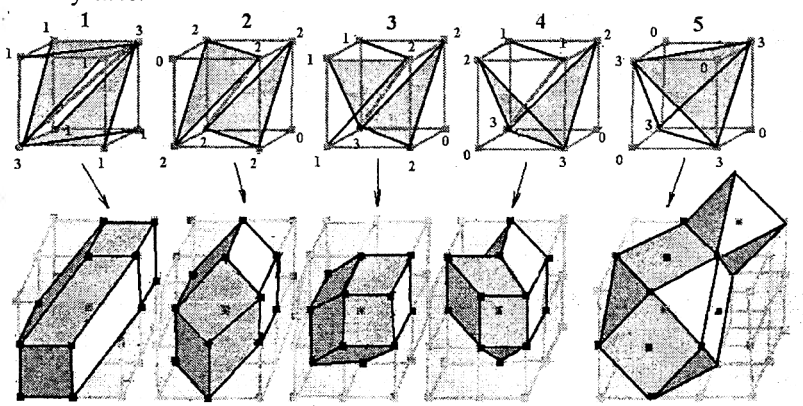

Рисунок 5. Неконгруэнтные триангуляции куба и соответствующие полиэдры звезд.

Каноническая триангуляция куба во всех октантах порождает звезду-комплекс, которой соответствует кубододекаэдр, гомеоморфный ромбододекаэдру [16]. Для **R3** в инструментальной системе этот комплекс играет определяющую роль и его параметры, как полиэдра:

вершин — 15, ребер — 50, граней — 48, симплексов — 24; объем — 4; кубододекаэдра, как многогранника: вершин — 14, ребер — 36 (24), граней — 24 (12). В скобках показаны числа без учета триангуляции граней-параллелограммов (см. рисунок 6).

Каноническая триангуляция куба, определяемая в некоторых работах и для более высоких размерностей как примитивная триангуляция [8] порождает на **Rn** звездчатую структуру, транслируемую на любой целочисленный вектор. Некоторые характеристики транслируемых звезд для ряда размерностей представлены в следующей таблице.

	2d	3d	4d	5d	6d
Число вершин	7	15	31	63	127
Число симплексов	6	24	132	960	8880

Примитивная триангуляция n-куба по определению [8], когда все симплексы имеют объем, равный 1/n!. Генерация примитивной триангуляции **Rn** может основываться на предложенном в [8] методе путевых симплексов, связывающем каждую подстановку из симметрической группы порядка n с конкретным симплексом на n-кубе. Так для **R3** ниже приведены соответствия между подстановками и множеством вершин соответствующих симплексов:

(1 2 3) → (0,0,0);(1,0,0);(1,1,0);(1,1,1);
(1 3 2) → (0,0,0);(1,0,0);(1,0,1);(1,1,1);
(2 1 3) → (0,0,0);(0,1,0);(1,1,0);(1,1,1);
(2 3 1) → (0,0,0);(0,1,0);(0,1,1);(1,1,1);
(3 1 2) → (0,0,0);(0,0,1);(1,0,1);(1,1,1);
(3 2 1) → (0,0,0);(0,0,1);(0,1,1);(1,1,1);

Другой способ основывается на отображениях n-куба с «большой» диагональю на себя (проекции в подпространства, связанные с гранями всех размерностей куба) [20].

В нашем случае мы расширяем класс рассматриваемых триангуляций до триангуляций на примитивных векторах нормы p=1. Так, пятый тип разбиения куба для **R3** не имеет все равные объемы симплексов и не подпадает под принятое определение примитивной триангуляции.

Для **R3** в инструментальной системе были выбраны триангуляции со звездами, соответствующими каноническому разбиению куба (1-ый тип) и 5-ому типу разбиения, полиэдры которых выпуклы и центрально симметричны.

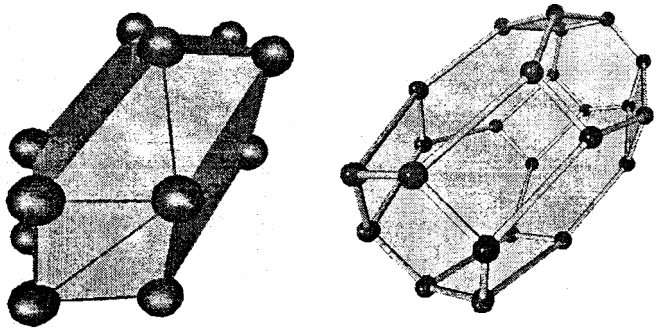

Рисунок 6. Кубододекаэдр, гомеоморфный ромбододекаэдру и двойственный ему многогранник, гомеоморфный усеченному октаэдру.

Решеточный веер и евклидовы приближения. Изложение в этом разделе в основном опирается на понятия веера в **Rn.** Оно практически полностью заимствовано из (13). Пусть **Rn**-евклидово пространство и **Zn** $\in$ **Rn** — стандартная целочисленная решетка. Для набора векторов $\mathbf{l}_1,\mathbf{l}_2,..\ \mathbf{l}_s \in \mathbf{Rn}$ определяется порожденный ими выпуклый многогранный конус σ как:

$$\sigma = \{r_1\mathbf{l}_1+\ldots+r_s\mathbf{l}_s \in \mathbf{Rn};\ r_i>0\}$$

Выпуклый многогранный конус является выпуклым полиэдром и поэтому определены грани выпуклого многогранного конуса. Порождающие векторы в нашем случае являются примитивными, поэтому конус σ является рациональным, строго выпуклым. Веером Σ называется такой набор из конусов, что грань каждого конуса содержится в Σ, и пересечение любых двух конусов из Σ является гранью каждого из них. Веер Σ в **Rn** называется полным, если объединение всех конусов из Σ есть все **Rn**.

В каждом конусе полного веера определим кратчайший путь между вершиной конуса и целой точкой **x**, внутренней для многогранного угла при вершине конуса, по ребрам решетки порождающих векторов обозначим через L.

Определим относительную погрешность между длиной этого пути и евклидовым расстоянием между этими точками:

$$\Delta i = L-Le/Le.$$

Тогда максимум этой величины по всем целым точкам **x**, внутренним в том же многогранном углу можно определить как относительную погрешность для этого конуса.

$$\Delta(C) = \max_i \Delta i;$$

Для каждого конуса эта погрешность может быть различной, поэтому для всего полного веера относительную погрешность определим как:

$$\Delta(\Sigma) = \max_c \Delta(C);$$

Задача состоит в том, чтобы построить набор порождающих векторов полного веера, чтобы: $\Delta(\Sigma) < \Delta_0$, где Δ_0 -заданная величина погрешности.

На основе итерационной процедуры над унимодулярными матрицами, строками которых являются порождающие (для конусов) примитивные векторы, в [19] построен алгоритм генерации такого набора. Так для **R3** итерационная процедура рассматривается вначале внутри трехгранного угла, образованного гранями, определяемыми тройками точек ((0,0,0); (1,0,0); (1,1,0)); ((0,0,0); (1,0,0); (1,1,1)); ((0,0,0); (1,1,0); (1,1,1)), что соответствует 1/48 части сферы. Проекции этих граней на сфере с центром в (0,0,0) образуют сферический треугольник, которому соответствует матрица

$$A0 = \begin{matrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{matrix};$$

Если внутри этого угла $\Delta(\Sigma) > \Delta_0$, то происходит разбиение этого угла (максимальной стороны сферического треугольника). Складываются строки матрицы, относящиеся к этой стороне, в нашем случае 1-ая и 3-ья строки, образуя новую целую точку в веере (2,1,1). В результате исходный треугольник разбит на два с соответствующими унимодулярными матрицами:

$$A1 = \begin{matrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 1 & 1 \end{matrix}; \qquad A2 = \begin{matrix} 2 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{matrix};$$

Разбиение продолжается до тех пор, пока не выполнится $\Delta(\Sigma) < \Delta_0$. Затем полученное построение зеркальными отображениями и поворотами распространяется на всю сферу, т.е. образуется полный веер.

На рисунке 7 показаны проекции ребер и граней конусов соответствующих полных вееров на поверхность сферы (**R2, R3**) и внутрь шара для **R4** (для различных Δ_0).

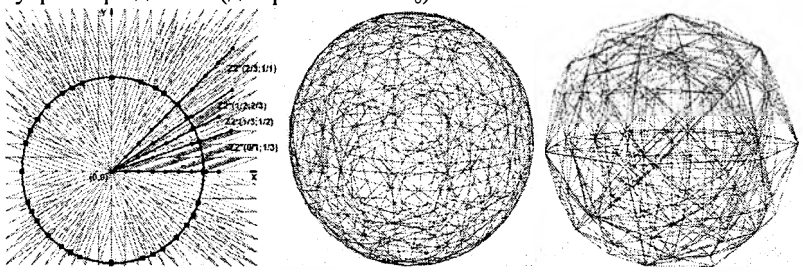

Рисунок 7. Решеточный 2d веер Фарея, проекции 3d веера на сферу и 4d веера внутрь шара.

Инструментальная система

В инструментальной системе, как в прототипе *комбинаторного топологического процессора*, реализуются следующие основные макрооперации:

1. Настройка системы по размерности (2d,3d,4d), размерам общей решетки и заданному типу звезды-полиэдра.
2. Генерация решеточного веера по заданной погрешности Δ [16].
3. Отображения множества преград-дефектов в симплициально-решеточную модель.
4. Прогон метрической волны — маркирование вершин при поиске кратчайших путей от множества-источника и построение *эквидистантного графа* (множество достижимых вершин с пометками и ребра, по которым реализуются кратчайшие пути до источника) [12].
5. Сжатие и расширение симплициальных комплексов без нарушения связности по заданным условиям [14].
6. Выделение триангулированой границы в симплициальном комплексе.
7. Перевод данных в openGL и VRML для визуализации.

Основные особенности для класса компьютеров desktop:

1. Допустимые размеры решеток до 200x200x200.
2. За счет использования симметрий в памяти хранятся по одной копии симплициального комплекса (полиэдра) и 1/48-ой($1/2^3 3!$) части решеточного веера. (Сокращение памяти для хранения по отношению к использованию вещественной арифметики более чем в 100 раз).
3. За счет табличного хранения в оперативной памяти всех возможных (2^{14} = 16 Кбит) ситуаций на границе полиэдра операция топологического анализа по связности выполняется за один такт (общее сокращение операций для 3d более 100 раз).

В качестве этюдного примера вариационной задачи рассмотрим следующую постановку. Построить триангулированную сферу минимального радиуса, как множество целых точек, удаленное на равное расстояние (с учетом обхода заданных преград) от заданного центра (x, y, z) при заданной Δ-относительной погрешности отличия от евклидовой метрики и при условии, что на сфере нет элементов преград. При данной постановке выполняются последовательно все макрооперации топологического процессора 1-2-3-4-5-6-7. Для решетки 50x50x50, центра (21, 18, 21) и двух преград-параллелепипедов на решение этой задачи на PC (Intel Celeron 2,66 GHz, 512 Mb RAM) потребовалось по макрооперациям: 1-3 < 0,01 сек; 4 – 33сек; 5 – 61сек; 6 – 29сек; 7 – 1сек. Один из возможных вариантов решения показан на рисунке 8а. На рисунке 8 также показаны сгенерированные с помощью системы (расширение и сжатие при заданной целевой функции)

структуры незамкнутого тора и приближения к минимальной поверхности Шварца.

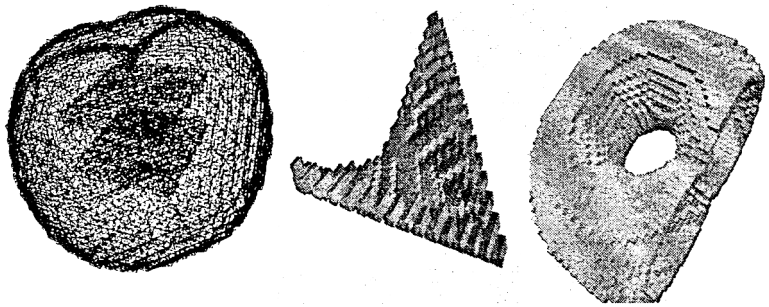

Рисунок 8. Триангулированная «сфера» с центром между прямоугольными преградами, приближение к поверхности Шварца, незамкнутый тор.

Заключение

Работа над симплициально-решеточными моделями в рамках инструментальной системы показала следующее.

1. При определенных допущениях возможно резкое сокращение памяти (около 100 раз) для хранения геометрико-топологических структур такой же сложности.
2. При использовании примитивных и других транслируемых триангуляций за счет табличного хранения результатов топологического анализа комплекса-звезды компьютерное время при реализации преобразований может было сокращено для 3d-случая примерно в 100 раз (для более высоких размерностей еще больше).
3. Для суперкомпьютеров возможно глубокое распараллеливание счета при одновременной обработке непересекающихся комплексов.
4. Возможен в ряде задач подход к их решению, как генерации на симплициальной решетке отметок обо всех возможных метрических решениях и последующего топологически корректного «вырезания» одного из приемлемых решений (самоорганизация процесса решения).
5. Целочисленные симплициально-решеточные модели естественно допускают дальнейшие произвольные линейные преобразования и использование более универсальных (но более емких по памяти) форм хранения и обработки данных.

В целом, представляется, что микропроцессорная реализация комбинаторно-топологических построений и преобразований (в качестве сопроцессора) поможет значительно расширить возможности современных суперкомпьютеров.

Авторы выражают благодарность Е.П.Велихову, Д.В.Аносову, Л.Н.Королеву, Б.Н.Четверушкину, Ю.М.Давыдову, А.В.Тихонравову, Г.И.Рузайкину, А.Н.Томилину за внимание и обсуждения изложенной выше тематики.

Литература

1. Понтрягин Л.С. Основы комбинаторной топологии.М., Наука, 1974.
2. Касселс Дж. Введение в геометрию чисел.М. Мир. 1965.
3. Чандрасекхаран К. Введение в аналитическую теорию чисел. М. Мир. 1974.
4. Гельфанд И.М. Лекции по линейной алгебре. М., МГУ, 1999.
5. Новиков С.П. Топология. Москва-Ижевск. РХД. 2002.
6. Kovalevsky V.A. Finite topology as applied to image analysis // Computer Vision, Graphics and image Processing. 1989.46/141-161.
7. Рябов Г.Г. Модели коммутационных свойств конструкций ЭВМ. М., ИТМиВТ, 1989.
8. Steingrimsson E. Permutations Statistics of Indexed and Poset Permutations. Proc. MIT. 1992.
9. Couprie M., Bertrand G. Simplicity surfaces: a new definition of surfaces in Z3 // Proc. of SPIE.1998.3454. 40-51.
10. Kenmochi Y., Imiya A. Discrete polyhedrization of lattice point set // LNCS, Springer Berlin/Heidelberg. 2001/2243.150-167.
11. Strand R., Borgefors G. Weighted distance transforms for volume images digitized in alongated voxel grids // Pattern Recognition Letters. 2004.25, N 5.571-580.
12. Рябов Г.Г. Маршрутизация на решетчато-клеточных структурах // Вычислительные методы и программирование. МГУ. 2004.3322.164-175.
13. В.М.Бухштабер, Т.Е.Панов. Торические действия в топологии и комбинаторике. М., МЦНМО. 2004.
14. Alliez P., Cohen-Steiner D.,Yvinec M., Desbrun M. Variational tetrahedral meshing // Proc. SIGGRAPH. 2005.24. N3. 617-625.
15. Четверушкин Б.Н. и др. Пакет прикладных программ GIMM для решения задач гидродинамики на многопроцессорных вычислительных системах. // М. Математическое моделирование. т.17 №6. 2005.58-74.
16. Рябов Г.Г. Метрические и топологические волны на решетках. Изд. МГУ. 2005.
17. Klette R., Li F. Shortest paths in a Cuboidal World // LNCS, Springer Berlin, Heidelberg. 2006.4040.415-429.
18. Г.Г.Малинецкий. С.А.Науменко. Вычисления на ДНК. Эксперименты. Модели. Алгоритмы. Инструментальные средства // Информационные технологии и вычислительные системы. №1, 2006, 5-27.

19. Рябов Г.Г., Серов В.А. Отображения целочисленных множеств и евклидовы приближения // Вычислительные методы и программирование. МГУ. 2007. т.8 N1, 10-19.
20. Ryabov G., Serov V. Simplicial-Lattice Model and Metric-Topological Constructions. Proc. conf. Pattern Recognition and Information Processing — PRIP 2007, v.2.135-140.

Н.П. Брусенцов, Ю.С. Владимирова

Конструктная компьютеризация аристотелевой силлогистики

Аристотелева силлогистика не вписывается в современные логические исчисления, и ее неоправданно квалифицируют как «узкую систему», неприменимую ко всем видам рассуждений, например, к математическим доказательствам [4, с. 189]. На самом деле, в этой системе неукоснительно соблюден установленный Гераклитом диалектический принцип сосуществования противоположностей, несовместимый с принятым в 3-м веке до н. э. стоиками «законом исключенного третьего», превратившим трехзначную диалектическую логику в «мертвую схоластику» [5, с. 326]. Возникшая таким образом «классическая» логика и по сей день препятствует развитию адекватного человеческого, а теперь и машинного интеллекта. В ней непредставимо фундаментальное логическое отношение необходимого содержательного следования. В условиях двухзначности оно выродилось в парадоксальную материальную импликацию. Настойчивые попытки устранить парадоксы изобретением строгих, сильных, релевантных и т. п. импликаций проблемы не решили и не могли решить, поскольку непарадоксальное следование — отношение трехзначное, в двухзначной логике не осуществимое.

В силлогистике оно представлено общеутвердительной посылкой «Все x суть y», истолковываемой Аристотелем как содержательное необходимое следование [3]: «Всем x присуще y», «Сущность y содержится в сущности x», «Из x необходимо следует y», «Если есть x, то не может не быть y». В этом и заключается естественноязыковый (здравый) смысл логического следования: оно необходимо предполагает сосуществование противоположностей, т. е. вещей, которым присущи рассматриваемые термины, и вещей, вещам, которым эти термины антиприсущи: x-вещей с x'-вещами, y-вещей с y'-вещами, и необходимо соблюдено, если при этом исключены (не существуют) xy'-вещи. Если существует хотя бы одна xy'-вещь («Некоторые x суть y'»), следование $x \Rightarrow y$ невозможно. Если же существование xy'-вещей не отрицается и не утверждается, то следование $x \Rightarrow y$ возможно, но не необходимо, может быть, а может и не быть (акцидентально). Наконец, при несоблюдении сосуществования противоположностей следование (как и всякое логическое отношение) немыслимо, не существует. Суждение, которым оно выражено, оказывается *химерой*, суждением невесть о чем.

Например, материальная импликация $x \to y$ в угоду предписанной законом исключенного третьего двухзначности

истолковывается как отношение, соблюденное при несуществовании y'-вещей. Полагают, что из несуществующего следует все, что угодно, не могущее не существовать следует из чего угодно, называя это парадоксами материальной импликации. Но ведь здравый смысл следования в том, что нечто определенное (а не «все, что угодно») не может не быть, когда есть то, из чего оно следует, и абсурдно «следование из ничего». Парадоксы импликации возникли в результате стремления сделать логику недопустимо простой – двухзначной, не заботясь о ее адекватности. В двухзначной логике нет места сосуществованию противоположностей, и поэтому в ней исключена не только Гераклитова диалектика бытия (логос), но и первейшее отношение адекватной логики – непарадоксальное следование.

В аристотелевой силлогистике термины x, y, z, ... служат символами качеств, присущностью либо антиприсущностью которых определяются сущности рассматриваемых вещей. Как присущность, так и антиприсущность качества (например, x и x') – существенные особенности вещей. Сущность вещи выражается конъюнкцией (совместностью) ее существенных особенностей. Несущественные особенности в конъюнкцию не включаются (умалчиваются). Например, сущность квадрата – xyz, где x – четырехугольность, y – прямоугольность, z – равносторонность. Сущность ромба – xz, а $xy'z$ – непрямоугольный ромб.

Согласно принятому в аристотелевой силлогистике принципу сосуществования противоположностей, ни одна из особенностей не может быть ни общезначимой, ни бессодержательной. Должны одновременно существовать все вещи: x- и x'-, y- и y'-, z- и z'- (в приведенном примере должны существовать четырехугольные и нечетырехугольные фигуры, прямоугольные и непрямоугольные, равносторонние и неравносторонние). Только в этом случае имеется возможность адекватно рассуждать о рассматриваемых вещах, выявлять подлинные, а не принятые для удовлетворения «потребностей математических применений логики» [6, с. 79] взаимосвязи охарактеризованных такими особенностями сущностей.

В приведенном примере $xz \equiv x(y \vee y')z$, где $y \vee y'$ — сосуществование противоположностей. Сопоставление сущности квадрата xyz с сущностью ромба xz обнаруживает, что вторая из них содержится в первой и следует из нее. Действительно, всякий квадрат необходимо есть ромб. Формально это выражается в том, что конъюнкция xyz и xz тождественна xyz. Содержательно соблюденность необходимого следования $xyz \Rightarrow xz$ проявляется в том, что консеквент xz получается из антецедента xyz удалением термина y, т. е. преобразованием существенной особенности в несущественную. В общем случае следование $A \Rightarrow B$ необходимо соблюдено, если сущность

B образована из сущности *A* переводом некоторых ее особенностей в несущественные, т. е. удалением из нее соответствующих терминов. Обратное следование $A \Leftarrow B$ при этом будет не необходимым («Некоторые *B* суть *A*»). Например, «некоторые ромбы суть квадраты».

Выражающая отношение следования $x \Rightarrow y$ общеутвердительная силлогистическая посылка A*xy* («Все *x* суть *y*»), представима конъюнкцией трех дизъюнктов [2]:

$$Axy \equiv Vxy V'xy' Vx'y',$$

где Vxy – существование xy-вещей, $V'xy'$ – несуществование xy'-вещей, $Vx'y'$ – существование $x'y'$-вещей. Умалчивание существования/несуществования $x'y$-вещей означает несуществовенность его для представленного отношения.

Принятый в универсуме Аристотеля УА [1, с. 91] принцип сосуществования противоположностей выражается конъюнкцией $VxVx'VyVy'$, где Vx, Vx', … - существование x-, x'-, … вещей. Общеутвердительная посылка, преобразованная в минимальную форму имеет вид:

$$Axy \equiv VxV'xy'Vy'.$$

Тогда в УА, т.е. при условии сосуществования противоположностей, A*xy* выражается одним означающим несовместимость *x* и *y*′дизъюнктом $V'xy'$.

Частноутвердительная посылка I*xy* («Некоторые *x* суть *y*»), выражающая не необходимое следование, представляется в УА дизъюнктом Vxy, – совместной данностью *x* и *y*. Общеотрицательная посылка E*xy* «Все *x* суть *y*′» получается из A*xy* инверсией *y*:

$$Exy \equiv Axy' \equiv V'xy'.$$

Соответственно частоотрицательнаая посылка O*xy* как инверсия общеутвердительной A*xy* будет:

$$Oxy \equiv Vxy' \equiv Ixy'.$$

Таким образом, при использовании инверсии терминов в силлогистике достаточно двух функторов – A и I, выражаемых в УА функциями несуществования $V'xy'$ и существования Vxy, которые оказываются трехзначными, принимающими соответственно значения “+” и “-”, а в случае $Vxy \vee V'xy$ – значение “0”.

Компьютеризация аристотелевой силлогистики просто и экономно реализуется при помощи четырехтритных конструктов [7], кодирующих трехтерминные дизъюнкты существования и несуществования. Значение первого (головного) трита указывает тип дизъюнкта: “+” – существование, представляющее частную посылку, “-” – несуществование, общая посылка. Последующие триты сопоставлены терминам *x*, *y*, *z*, указывая их статусы. Например, $Axy \equiv V'xy' \equiv$ (-+-0), $Iyz \equiv Vyz \equiv$ (+0++), $Vxy'z \equiv$ (++-+), $V'xy'z \equiv$ (-+-+).

Заключение из пары посылок реализуется переводом среднего термина в несущественные, т. е. его элиминированием или склеиванием, если оно возможно. Например, из двух общих посылок выводится общая:

$$A{xy}A{yz} \equiv V'{xy}'V'{yz}' \equiv V'(xy' \vee yz') \equiv$$
$$\equiv V'(xy'z \vee xy'z' \vee xyz' \vee x'yz') \equiv V'(xy' \vee xz' \vee yz') \equiv$$
$$\equiv V'{xy}'V'{xz}'V'{yz}' \Rightarrow V'{xz}' \equiv A{xz}.$$

Применительно к представленным тритными конструктами посылкам склеивание реализуется как потритное «логическое сложение» $\oplus$ (рис. 1) соответствующих конструктов. Например,

$$A{xy}A{y} \equiv V'{xy}'V'{yz}' \equiv (-+-0) \oplus (-0+-) \equiv (-+0-) \equiv V'{xz}' \equiv A{xz}.$$

Частное заключение получается склеиванием конструкта, кодирующего частную посылку с инверсией представляющего общую. Например,

$$I{xy}A{yz} \equiv (+++0) \oplus inv(-0+-) \equiv (+++0) \oplus (+0-+) \Rightarrow (++0+) \equiv I{xz}.$$

Если склеивания общей посылки с частной нет (элиминации среднего термина не происходит), то и заключения не существует.

Из двух общих посылок не всегда возможно общее заключение, но непременно есть частное, для получения которого вместо одной из общих употребляются подчиненные ей частные, с одной из которых заключение необходимо будет. Например, из $A{xy}'A{yz}$ общего заключения нет. Посылке $A{xy}'$ подчинены $I{xy}' \equiv V{xy}'$ и $I{x'y} \equiv V{x'y}$.

$$I{xy}'A{yz} \equiv V{xy}'V'{yz}' \equiv (++-0) \oplus inv\,(-0+-) =$$
$$= (++-+) - \text{нет заключения}.$$
$$I{x'y}A{yz} \equiv V{x'y}V'{yz}' \equiv (+-+0) \oplus inv(-0+-) =$$
$$= (+-+0) \oplus (+0-+) \Rightarrow (+-0+) = V{x'z} \equiv I{x'z}.$$

Таким образом, из двух общих посылок заключение есть всегда, но в зависимости от того, являются ли посылки транзитивными, можно сделать заключение либо о необходимом следовании z из x ($A{xz}$), либо о не необходимом следовании ($I{xz}$). В случае одной частной и одной общей посылки следование никогда не будет необходимым, а возможна также ситуация, когда никакого вывода сделать нельзя.

Благодаря принятию взамен закона исключенного третьего принципа сосуществования противоположностей, удалось полностью компьютеризовать аристотелеву силлогистику, причем интеллект реализованной в диалоговой системе структурированного программирования ДССП программы силлогистического вывода значительно превзошел то, что достигнуто «невооруженными» умами людей. Число правильных модусов, составляющее в традиционной логике 19, а в математической логике сокращенное до 15, оказалось равным 128. В каждой из четырех фигур имеется по 32 правильных модуса.

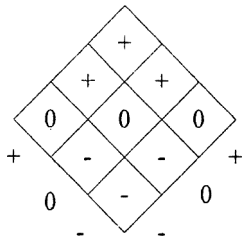

Рисунок 1. Таблица операции ⊕.

Литература

1. Брусенцов Н.П. Искусство достоверного рассуждения. – М.: Фонд «Новое тысячелетие», 1998.
2. Брусенцов Н.П. Трехзначная диалектическая логика. // Программные системы и инструменты: Тематический сборник факультета ВМиК МГУ им. Ломоносова: № 2. Под ред. Л.Н. Королева. – М.: Изда-тельский отдел ВМиК МГУ, 2001. С. 36-44.
3. Аристотель Сочинения в четырех томах. – М.: "Мысль", т. 1 – 1975. С. 173-177.
4. Лукасевич Я. Аристотелевская силлогистика с точки зрения современной формальной логики. – М.: ИЛ, 1959.
5. Ленин В.И. Философские тетради. – М.: Политиздат, 1973.
6. Гильберт Д., Аккерман В. Основы теоретической логики. – М: ИЛ, 1947.
7. Брусенцов Н.П., Владимирова Ю.С. Конструктная компьютеризация силлогистики // Математические методы распознавания образов. ММРО-13. – М.: МАКС-Пресс, 2007. С. 10-13.

Леонов М.В., Иванов А.В., Клеменков П.А., Пейсахов И.Б.

О мобильных практикумах и информационных системах

Введение

Под мобильным практикумом будем понимать переносное запоминающее устройство с программным обеспечением, не требующим инсталляции на жестких дисках компьютера и обеспечивающим среду для выполнения заданий вычислительного практикума. Несколько лет назад в качестве такого переносного устройства нами использовались модифицированные LiveCD с дистрибутивом Knoppix, а теперь, в связи с резким удешевлением флэш-памяти, мы стали использовать так называемые «флэш-брелки», или флэш-карты. Эти носители информации обладают очевидными существенными преимуществами перед CD: во-первых, позволяют записывать на них же результаты работы, а во-вторых, — удобны для модификации, в-третьих, их емкость варьируется в более широких пределах.

Аналогичным образом можно определить и мобильную информационную систему — как переносное устройство с информационной системой, не требующей инсталляции на компьютере. Такие системы удобны для демонстраций на конференциях, а также для использования вне привычного рабочего места.

Требования к вычислительной системе, в которой предполагается использовать такие носители, могут варьироваться. Для некоторых вариантов достаточно возможности загрузки операционной системы с флэш-карты, в других требуется наличие определенной операционной системы. Определенные требования накладываются и на программы, входящие в мобильную систему. Так, если речь идет о Windows, они не должны вносить свои настройки в системный реестр операционной системы компьютера, все временные файлы должны хранить в своем локальном каталоге, и т.д.

1. Постановка и актуальность наших задач

Мы рассматриваем две задачи. Первая из них — подготовка флэш-карты с программной средой для выполнения заданий практикума на языках Паскаль и Си, а также заданий по работе в операционных системах семейства UNIX.

Вторая задача — подготовка флэш-карты для мобильных информационных систем, использующих для своей работы СУБД

MySQL, Web-сервер (Apache или аналогичный) и скриптовый язык PHP.

Есть несколько причин для совместного рассмотрения решения в наших условиях. Кроме свойства мобильности, решения должны быть основаны на свободном программном обеспечении [1], иначе у преподавателя практикума (или разработчика информационных систем) возникают юридические проблемы при распространении подготовленного программного обеспечения. Еще одна общая особенность – необходимость мониторинга практически одних и тех же Интернет-источников для отслеживания новых версий программ и появления новых инструментов.

Актуальность мобильного практикума следует из законного стремления преподавателя к независимости от администратора компьютерного класса, минимизации угрозы срыва занятий, в желании студентов иметь единую среду для выполнения заданий на занятиях в ВУЗе и дома, что особенно актуально для студентов вечерних и заочных отделений. Кроме того, такой подход обеспечивает «мягкое» сосуществование на одном компьютере Windows и Linux, что обычно важно для упомянутой категории пользователей.

Актуальность мобильных информационных систем для работников естественно-научных областей, да и других категорий пользователей, в частности, следует из специфики труда этих пользователей: их работа с базами данных не ограничивается рабочим местом и установленным администрацией графиком работы, а надежность и доступность Интернет-соединений пока рано считать неотъемлемым свойством рабочих мест наших коллег.

2. Выбор инструментария

Мы исходим из необходимости иметь варианты решения как для Windows, так и для операционной системы из семейства UNIX, точнее, ее «подсемейства» Linux. Экспериментальный характер работы и участие в ней студентов спецсеминара позволили опробовать несколько вариантов для операционной системы Linux, в том числе дистрибутивы SLAX[2] и Knoppix [3].

Законченные результаты получены на базе пакета PortableApps [4] для операционной системы Windows, дистрибутивов SLAX и Knoppix. К моменту написания статьи не удалось опробовать вариант отечественной фирмы ALTLinux, хотя планы и надежды остаются.

Из многообразия (только основных не менее десяти) дистрибутивов мы выбрали представителя, основанного на Debian GNU/Linux – LiveCD Knoppix и представителя, основанного на Slackware – LiveCD Slax.

Дистрибутив Knoppix был одним из первых LiveCD дистрибутивов. Он был разработан немецким инженером Клаусом

Кноппером (Klaus Knopper), отличается немецкой основательностью и отвечает заявленной автором универсальности. К примеру, один из его DVD-дистрибутивов содержит более 5 гигабайтов программного обеспечения.

Автор дистрибутива Slax – чешский программист Томас Матейчек (Tomas Matejicek). Одно из его отличий – отсутствие сложного пакетного менеджера для генерации версии, унаследованное от дистрибутива Slackware. Пакетный менеджер Slackware представляет собой скрипт на Bash, который распаковывает tgz архив с перекомпилированным программным обеспечением в дерево каталогов. Просто реализовано и удаление пакетов. Отсюда простота сборки, так как зависимости между пакетами не анализируются, и вся ответственность на программисте (или простом пользователе), осуществляющем сборку.

Отметим одно из основных препятствий подготовки мобильного практикума на флэш-карте. Почти все Live-дистрибутивы (которые берутся за основу для флэш-варианта) предлагают довольно жестко определенный набор программного обеспечения, причем, как правило, без средств программирования на языке Паскаль. Язык Паскаль, по-видимому, не пользуется популярностью у разработчиков мобильных систем, а нам он необходим в соответствии с учебным планом.

3. Пакет «Мобильный практикум» (по мотивам пакета PortableApps)

Пакет PortableApps, у истоков разработки которого был программист Джон Галлер (John Haller), состоит из программ, модифицированных одноименной компанией для запуска с флэш-карты под управлением Windows. Из предлагаемых на сайте компонент мы взяли систему программирования на C/C++ под названием Dev-Cpp и пакет OpenOffice.org – для того, чтобы наш мобильный практикум мог использоваться и для изучения основ информационных технологий в ву-зах со специальностями 030100 (Информатика. Специализация – учитель информатики). Кроме того, была добавлена свободно распространяемая система программирования FreePascal и заменена оболочка меню быстро-го доступа. Эта замена вызвана тем, что программа PortableAppsMenu, к сожалению, не имела опций по добавлению программ, отличных от при-сутствующих на сайте PortableApps. Эта возможность есть у свободно распространяемой программы PStart [5], которой мы и воспользовались для подготовки своего пакета (см. рисунок 1). Программа PStart, кроме всего прочего, позволяет и организовать гипертекстовый справочник: ее конфигурация определяется xml-файлом, открытым для редактирования.

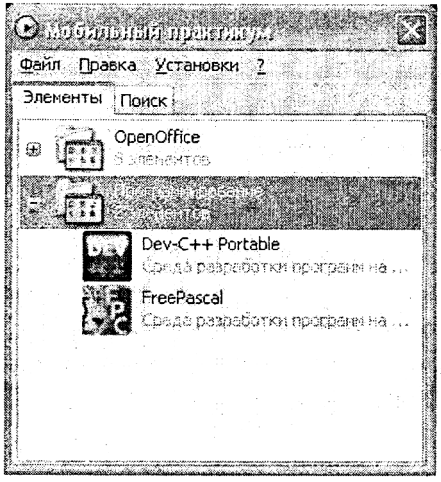

Рисунок 1.

4. Мобильная информационная система на основе пакета PortableApps

В состав входит интегрированный пакет Xampp, включающий в себя «мобильные версии» Web-сервера Apache, СУБД MySQL и интерпретатор PHP. Именно эти компоненты используются на нашем совместном с Ботаническим садом МГУ сайте Umbelliferae Information Server [6] для функционирования нескольких ботанических информационных систем.

Поэтому этот пакет мы взяли в качестве инструмента для подготовки мобильного варианта нашего сайта, размещающегося на флэш-карте объемом в 2 гигабайта.

Справедливости ради отметим, что ранее у нас существовала другая мобильная версия сайта, собранная на основе пакета microweb [7], бесплатно использовать который можно законно лишь строго определенное время.

Фактически наш мобильный сайт включает несколько мобильных информационных систем, в частности, интерактивную карту распространения азиатских видов растений семейства Зонтичных (см. рисунок 2) на территории России.

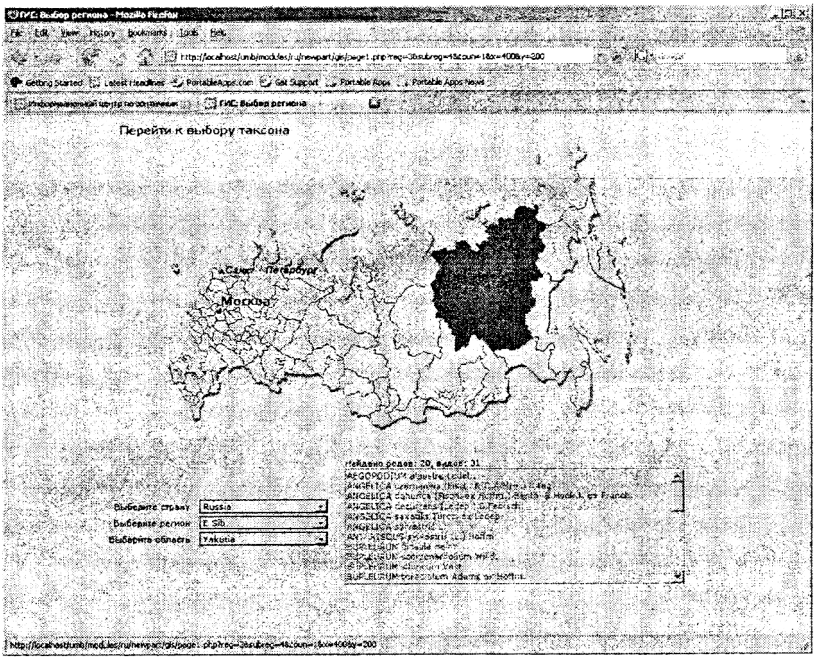

Рисунок 2.

5. Мобильные практикумы на основе LiveCD дистрибутивов Linux

Наш вариант мобильного практикума под названием «SLAX-практикум» основан на версии SLAX Server Edition v 5.1.8.1 с включением нескольких модулей:

gcc_3_4_6.mo,

Lazarus_IDE_Free_Pascal_Compiler_0_9_14_FPC2_02.mo,

Official_Development_module_5_1_4.mo,

Slax_Utilities_0_2-5.mo и некоторых других.

Надо отметить, что дистрибутивы Slax фактически предоставляют пользовательский интерфейс для внесения изменений в конфигурацию системы. Добавление новых модулей сведено к простому копированию в каталог *modules*. Это удобно, так как пользователю, осваивающему все новые и новые инструменты, обычно трудно предугадать заранее, какие модули ему потребуются в будущем.

Кратко алгоритм подготовки версии на основе SLAX – LiveCD сводится к следующему.

На отформатированную под файловую систему FAT32 флэш-карту переносим все файлы из iso-образа LiveCD (с помощью одной из программ для работы с iso-образами, например, Ultra ISO). Из каталога

boot копируем файлы vmlinuz (скомпилированное бинарное ядро) и initrd.qz (архив корневой файловой системы и стартовый скрипт) в корневой каталог. С помощью программы-загрузчика ядра Linux syslinux делаем флэш-карту загрузочной. Заменяем конфигурационный файл isolinux.cfg на syslinux.cfg с опциями для загрузчика syslinux (либо переименовываем и при необходимости редактируем). Дополнительные модули, в нашем случае это модуль системы программирования FreePascal, модули русификации и Development module version 5.1. (содержащий, в частности, редактор связей) переписываем в каталог *modules*.

Создание кастомизированного (от слова customize – изготовлять на заказ) варианта дистрибутива Knoppix у нас состоит из следующих основных этапов (подробнее эта процедура описана, например, в [8] и [9]).

1. Развертывание одной из базовых версий на достаточно мощном компьютере.
2. Создание каталога на свободном разделе жесткого диска с двумя подкаталогами: для дерева создания образа и для результата кастомизации. Перенос необходимых файлов для копии системы Knoppix.
3. Установка новых пакетов, в полученной копии с помощью менеджера пакетов APT, в нашем случае это пакет с системой FreePascal.
4. Подготовка флэш-карты: создание и форматирование раздела под файловую систему FAT32.
5. Установка в созданном разделе загрузчика syslinux.
6. Копирование файлов из нового образа Knoppix в созданный раздел на флэш-карте с переименованием конфигурационного файла isolinux.cfg в syslinux.cfg (с предварительным выполнением команды mount для нового раздела и подготовленного образа Knoppix).
7. После команды umount для флэш-карты она готова для работы.

Более сложная процедура обновления состава модулей связана с тем, что Knoppix, как LiveCD, основанный на Debian, использует мощный менеджер APT, который автоматически разрешает зависимости между пакетами. Поэтому, например, при использовании старой версии Knoppix с текущим репозиторием Debian проявляется место пакетная несовместимость, то есть установка одного нового пакета приводит к обновлению практически всей пакетной базы.

Внешний вид рабочего стола нашей версии мобильного практикума на основе Knoppix изображен на рисунке 3.

В качестве некоторого итога наших экспериментов и практической работы хотелось бы подчеркнуть, что сложность дистрибутива Knoppix с лихвой компенсируется достоинствами:

надежным определением устройств, документированностью и т.д. Зато LiveCD Slax работает, как правило, быстрее, даже на маломощных машинах. Это объясняется тем, что сборка программного обеспечения для LiveCD Slax осуществляется под архитектуру i386.

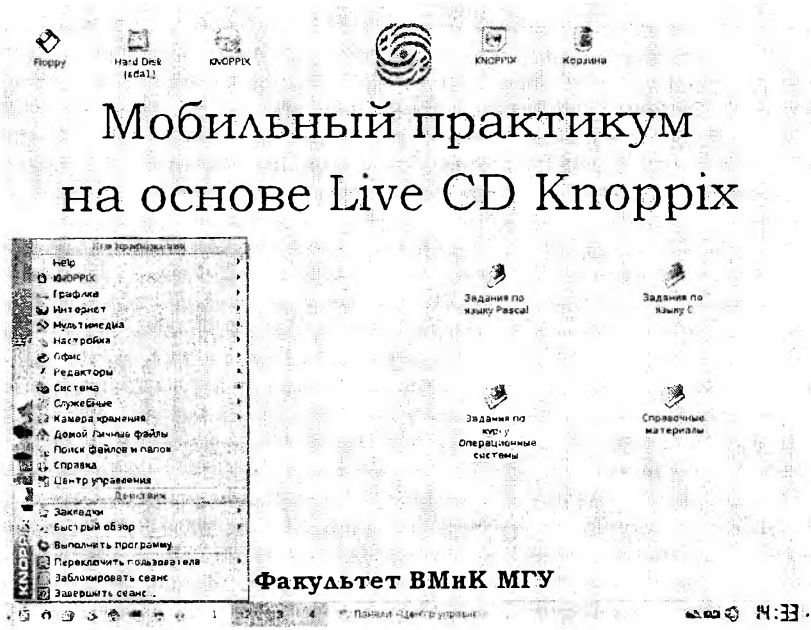

Рисунок 3.

Заключение

Опыт, полученный при подготовке и эксплуатации мобильных практикумов и информационных систем на основе свободного программного обеспечения, позволяет сделать следующие выводы.

Мобильные практикумы могут стать средством повышения эффективности и комфортности работы преподавателя и студентов. В использовании их есть и методологическая ценность: студент получает стимул в совершенствовании и более глубоком осмыслении собственной вычислительной среды.

Мобильные практикумы незаменимы для «гостевых» преподавателей, облегчая проведение занятий в незнакомой обстановке.

Разработка специализированных мобильных информационных систем дает специалистам предметных областей дополнительные степени свободы в исследованиях и представлении своих результатов на конференциях и других мероприятиях.

Однако внедрение этих инноваций требует определенных усилий от преподавателей, в частности, в отслеживании новостей свободного программного обеспечения, а также усилий по подготовке инструктивных материалов для учащихся и студентов.

Авторы выражают глубокую благодарность за поддержку работы профессору Евгению Владимировичу Захарову.

Ссылки

1. http://ru.wikipedia.org/wiki/Opensource.
2. http://www.slax.org.
3. http://www.knoppix.net.
4. http://www.portableapps.com.
5. http://www.pegtop.net/start/.
6. http://www.umbelliferae.cs.msu.su/.
7. http://www.indigostar.com/microweb.htm.
8. http://www.knoppix.net/wiki/Knoppix_Remastering_Howto.
9. http://www.knoppix.net/wiki/Bootable_USB_Key.

Махнычев В. С., Ильичев А. Б.

Распараллеливание задачи о составлении вузовского расписания с использованием распределенного обхода дерева перебора

Введение

Задача о составлении вузовского расписания — это задача о составлении расписания комбинаторного типа с дополнительными условиями оптимальности решения и ограничениями сложной формы. Особенностью задачи является большая многовариантность, не позволяющая перебрать все возможные комбинации для получения точного решения.

Для решения подобных задач применяются комбинации переборных и эвристических алгоритмов. При этом количество рассмотренных в переборе вариантов и – как следствие – точность полученного решения напрямую зависят от мощности используемой вычислительной системы. Таким образом, для получения решения, наиболее близкого к оптимальному, становится актуальным эффективное распараллеливание алгоритма решения задачи.

Для распараллеливания была разработана специальная комбинация переборного и эвристического алгоритма, позволившая применить селективную схему параллельного обхода дерева перебора, ранее проверенную на алгоритме игры в шахматы и подробно описанную в [1]. Особое внимание уделялось оптимизации переборного алгоритма, так как именно он подлежит распараллеливанию и определяет эффективность параллельной схемы. Для испытания реализованной системы использовалось несколько различных компьютеров, объединенных в локальную сеть.

Постановка задачи

В общем случае задачу о составлении вузовского расписания можно сформулировать следующим образом.

Дано множество групп G, множество преподавателей C, множество академических часов (например, за неделю) T и множество аудиторий P. Определим занятие a как четверку (g,c,t,p), $g \in G$, $c \in C$, $t \in T$, $p \in P$, означающую, что преподаватель c проводит занятие в группе g в академический час t в аудитории p. Требуется составить график занятий A (множество занятий) таким образом, чтобы количество часов, проведенных в каждой группе определенным преподавателем, соответствовало заданному для этой группы (исходя из учебного плана).

При этом один преподаватель не может вести занятия в нескольких группах одновременно, то есть

$$\forall a_0 = (g_0, c_0, t_0, p_0) \in A \Rightarrow \bigcup_{\substack{a=(g,c,t,p)\in A,\\ t=t_0, p=p_0, g\neq g_0}} a = \phi$$

Кроме этого, специфика высшего образования вносит дополнительные элементы расписания – такие, как лекции, читаемые одним преподавателем для нескольких групп одновременно. Некоторые виды занятий требуют также аудиторий определенного типа (например, поточные лекции могут проходить только в больших аудиториях, а семинары по программированию должны включать в себя занятия в машинном зале). На аудитории также накладывается ограничение: ни одна аудитория не может быть одновременно занята более чем одним преподавателем.

Также задаются дополнительные условия оптимальности решения, как-то: нежелательность «окон» (пустых пар) между занятиями групп, нежелательность занятий в субботу, индивидуальные пожелания преподавателей и так далее.

Оценка оптимальности решения

Для оценки эффективности каждого полученного решения использовалась система неотрицательных штрафов: для каждого занятия и каждого места в расписании задан штраф за назначение данного занятия на данное место. Легко видеть, что при этом не играет существенной роли, в какой аудитории будет проходить занятие, значение имеет только тип необходимой аудитории. Таблица штрафов строится эвристическим алгоритмом. Отличительной особенностью этой системы является возможность вычислить точный штраф для неполного решения, когда известна лишь часть расписания – например, расписание только для части групп; и, что немаловажно, вычисленный штраф не зависит от изменения остальной части решения (расписания остальных групп), то есть входит в суммарный (полный) штраф в качестве постоянного слагаемого.

Кроме того, для каждой отдельно взятой группы можно получить некоторое число m_i – минимальный штраф для этой группы при условии, что все используемые группой ресурсы (преподаватели и аудитории) не заняты другими группами. В рамках этого предположения задача составления оптимального расписания для одной группы есть не что иное как задача о назначениях. Для ее решения используется венгерский метод.

Таким образом, вычислив суммарный штраф для сформированной части расписания и прибавив к ней минимальные штрафы m_i для тех групп, расписание которых еще не было составлено,

мы получим нижнюю границу полного штрафа (очевидно, что при составлении общего расписания штраф каждой конкретной группы может только увеличиться по сравнению с минимальным штрафом). А так как алгоритм старается составить оставшуюся часть расписания «идеально», то штраф в оставшейся части – в случае успешного решения задачи – близок к сумме минимальных штрафов m_i, и полный штраф близок к полученной нижней границе. Эта особенность – близость предполагаемого штрафа оптимального решения к оценке, полученной быстрыми приближенными вычислениями, – позволила применить метод ветвей и границ в качестве алгоритма перебора.

Переборный блок

При такой системе оценок удобно было выбрать в качестве одного шага ветвления назначение одного занятия. При этом, в чистом виде метод ветвей и границ не применим из-за огромного количества ветвлений (при шести учебных днях и пяти возможных занятиях в день число вариантов расписания для одной группы приближается к факториалу 30, а для нескольких десятков групп – факультета целиком – количество различных вариантов необозримо даже при самых эффективных отсечениях ветвей). Поэтому число вариантов на каждом уровне перебора ограничивалось некоторой функцией, которая зависит от общего числа вариантом в данной вершине, от глубины этой вершины в дереве перебора и от числового параметра, который медленно увеличивался по мере увеличения времени перебора. При таком ограничении числа вариантов имеет смысл рассматривать варианты в порядке уменьшения их перспективности. Поэтому в каждой вершине дерева перебора варианты сортируются в соответствии с их штрафом.

Для удобства применения селективной схемы в переборном алгоритме используется метод итеративного углубления, заключающийся в том, что на первой итерации перебор затрагивает один уровень дерева перебора, на второй итерации – два уровня, и так далее. Это было реализовано путем введения дополнительного параметра в функцию, ограничивающую число вариантов на каждом уровне перебора. Этот параметр определяет глубину в дереве перебора, начиная с которой в каждой вершине рассматривается ровно один вариант. При рассмотрении на каждом уровне ровно одного варианта, переборный алгоритм фактически вырождается в жадный алгоритм. При этом часть дерева выше этого уровня соответствует собственно перебору, а ветвь, расположенная ниже, соответствует решению, которое строит жадный алгоритм.

Эвристический алгоритм

Для построения таблицы штрафов использовался «жадный» алгоритм с некоторыми эвристиками. Общая схема работы алгоритма такова:

1. все занятия сортируются согласно некоторому порядку;
2. для каждого занятия изначально устанавливается высокий штраф для всех мест в расписании;
3. последовательно просматриваются все занятия и для каждого занятия, с учетом оценки на количество свободных ресурсов и общую занятость группы, определяется день, в который это занятие желательно назначить. После чего штрафы для этого дня понижаются;
4. пожелания преподавателей вносятся в таблицу в явном виде.

В алгоритме учитывается также нежелательность «окон» в расписании отдельных групп. Чтобы избежать «окон», предусмотрены следующие меры. Во-первых, порядок рассмотрения занятий таков, что сначала рассматриваются все лекционные занятия. Это существенно, поскольку лекция проходит одновременно у многих групп, и ее назначение затруднительно, когда часть расписания уже построена. Кроме того, при выборе «желательного» дня для каждого занятия наименьший штраф устанавливается для 3-й пары, чуть больше – для 2-й и 4-й, и наибольший – для 1-й и 5-й. Это, с одной стороны, делает окна в расписании невыгодными с точки зрения переборного алгоритма. С другой стороны, поскольку занятия можно назначать как до уже назначенных, так и после, это увеличивает число вариантов для назначения занятий «без окон». В противоположность этому: если мы назначим занятие 1-й парой, то у нас будет только один вариант для назначения следующего занятия в этот день так, чтобы не получилось «окон».

Стоит, кроме того, заметить, что тот же самый порядок занятий используется в переборе: занятия назначаются именно в том же порядке. Это делает возможным не осуществлять перебор из корня дерева, а предварительно назначить несколько занятий эвристически. Это фактически достигается тем, что функция, ограничивающая число вариантов на каждом уровне перебора, задает очень маленькие ограничения на верхних уровнях. Фактически самые верхние уровни перебора, на которых рассматривается только один вариант, соответствуют эвристическому назначению первых занятий, причем в нашем случае это будут лекции.

Применение селективной схемы

Идея селективной схемы параллельного обхода дерева перебора [1] состоит в рассмотрении лишь нескольких, наиболее

перспективных ветвей дерева перебора и наличии в вершинах дерева, соответствующих узлам распределенной системы, проверяющих узлов, с помощью которых производится поиск новых перспективных вариантов. Метод итеративного углубления, применяемый для решения задачи о составлении расписания, хорошо согласуется с селективной схемой, так как позволяет получать уточняющиеся оценки для выбранных ветвей с течением времени.

Применение селективной схемы в чистом виде, начиная с корня дерева перебора, оказалось нецелесообразным. Причина этого в том, что часть расписания может быть легко составлена эвристическим алгоритмом без какого-либо перебора, и для этой части расписания существует еще достаточное количество вариантов составления расписания, при каждом из которых может быть получено оптимальное решение [2]. Таким образом, более целесообразным оказалось зафиксировать положение некоторого (подбираемого эмпирически) числа занятий в расписании и начать применять селективную схему с некоторой внутренней вершины дерева перебора.

В выбранной вершине применяется селективная схема перебора: для выбранной вершины на одном из узлов распределенной системы запускается переборный блок, по промежуточным результатам его работы определяются несколько наиболее перспективных ветвей, и для этих ветвей в свою очередь запускаются переборные блоки на других узлах распределенной системы. Система входит в режим перебора вариантов, при этом оператор системы уведомляется о текущем наилучшем решении. Решение о прекращении перебора принимается оператором системы.

Для ускорения работы алгоритма, все узлы системы информируются о текущем минимальном достигнутом значении полного штрафа. Это позволяет увеличить количество отсечений при переборе.

Результаты испытаний и направления дальнейших исследований

Описанная схема была реализована и испытана на распределенной системе из семи счетных узлов. При таком выборе, для корневой вершины перебора рассматривались две наиболее перспективные ветви; для каждой из этих двух ветвей рассматривалось еще по две наиболее перспективные ветви. Таким образом, селективная схема перебора позволила увеличить глубину просмотра алгоритма на два уровня, что заметно улучшило результаты на различных тестовых наборах, требовавших переборных действий для получения оптимального расписания: по сравнению с однопроцессорной версией того же алгоритма ускорение на тестах составило от 2 до 3,3 раз.

В текущей версии алгоритм не адаптирован к специфике составления расписания для нескольких потоков и курсов одновременно. Специфика состоит в том, что расписания для различных потоков и тем более курсов имеют намного меньше пересечений по общим ресурсам, чем расписания групп одного потока. С учетом этого, в дальнейшем предполагается адаптировать алгоритм, изменив переборный блок таким образом, чтобы минимизировать количество ненужных перестановок (например, при формировании расписания для каждого относительно независимого подмножества групп можно применять ту же стратегию определения точки переборного ветвления, которая применялась в реализованной схеме для всего множества групп).

Литература

1. Махнычев В. С. Распараллеливание сложных интеллектуальных задач, основанных на обходе дерева перебора // Программные системы и инструменты. Тематический сборник №5. – М.: Факультет ВМиК МГУ, 2005. С. 83-91.
2. Панкратьев Е. В., Чеповский А. М., Черепанов Е. А., Чернышёв С. В. Алгоритмы и методы решения задач составления расписаний и других экстремальных задач на графах больших размерностей // Фундаментальная и прикладная математика 2003, т.9, выпуск 1, с.235-251.

Раздел II
Методы обработки экспериментальных данных

Е.А.Попова

Анализ масштабируемости и производительности параллельного алгоритма построения ансамблей деревьев решений для задачи локализации нейронных источников

1. Введение

Задача локализации нейронных источников мозга по электроэнцефалографическим измерениям (ЭЭГ) рассматривалась в традиционных постановках [1]. Обратная задача ЭЭГ имеет неединственное решение и требует модели аппроксимации нейронных источников-диполей, которыми аппроксимируется абстрактная группа активных нейронов, находящаяся в непосредственной близости друг к другу. Решение оптимизационной задачи нахождения параметров диполей-источников, например, градиентным методом, сильно зависит от хорошего начального приближения, медленно сходится и требует больших вычислительных затрат на каждом итерационном шаге, особенно при большом количестве входных параметров задачи. Появление мощных супер ЭВМ позволяет взглянуть на проблему по-другому. В работе [2] был предложен алгоритм локализации нейронных источников на основе построения ансамбля деревьев решений. Алгоритм основан на постановке задачи классификации для нахождения параметров диполей, для которых модельный потенциал на поверхности головы максимально приближен к экспериментальным измерениям. Особое значение имеет случай, когда объем обучающего множества превышает размеры оперативной памяти вычислительной системы. Алгоритмам параллельного построения дерева решений посвящен целый ряд работ [3-6]. В традиционных методах построения деревьев решений, основные вычислительные затраты уходят на следующие шаги алгоритма:

1. Выборка подмножества данных для вычисления текущей точки разбиения (число переменных в выборе точки разбиения пропорционально квадратному корню общего числа переменных в базе данных).

2. Сортировка значений каждого атрибута для последующего вычисления энтропии. Сортировка может быть выполнена независимо для каждого атрибута.
3. Итеративные вычисления изменения энтропии для нахождения лучшего признака для разбиения. Поиск признака осуществляется за один проход отсортированного массива значений.
4. Распределение обучающих примеров по потомкам исходного узла.
5. Другие операции, каждая из которых занимает меньше 1% процессорного времени.

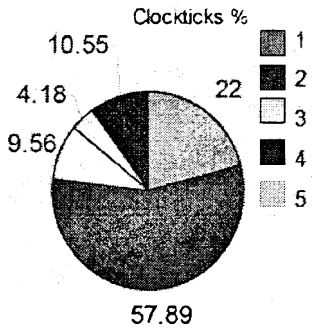

Рисунок 1. Анализ последовательной работы алгоритма построения дерева решений.

На рисунке 1 показано распределение времени работы алгоритма построения дерева решений, измеренного в тактах работы процессора. Показано, что операции (1)-(4) занимают около 90% процессорного времени. На каждом их 4-х этапов осуществляется обработка исходного обучающего множества данных, и если его размер превышает доступный объем оперативной памяти вычислительной системы, то затраты на обмен информацией могут быть очень значительными. В наиболее эффективных из существующих параллельных подходов построения дерева решений снижаются затраты на обмен информацией для вычисления точек разбиения в узлах дерева, оптимизируется работа с обучающим множеством (минимизировать число проходов по базе данных). Алгоритмы SLIQ [3] и SPRINT [4] представляют два основных алгоритма параллельного построения деревьев решений для данных большой размерности. SLIQ сортирует упорядоченные атрибуты только один раз и образует из входного обучающего множества списки атрибутов. Это влечет за собой значительные вычислительные затраты. Сортировка данных, находящихся на внешней памяти, представляет собой трудоемкую задачу, а составление дополнительных списков для атрибутов влечет за собой увеличение времени выполнения до 3-х раз. В дополнении к

этому, SLIQ требует хранения структур данных (называемых списками классов), их размер пропорционален числу записей в исходном обучающем множестве. Списки классов должны храниться в основной памяти, так как с ними осуществляется большая часть операций. Алгоритм не требует построение списков классов, вместо этого формируются хэш-таблицы, которые пропорциональны числу записей, ассоциированных с узлом дерева решений.

Существуют другие методы, которые оптимизируют проходы по обучающему множеству, избегая полной сортировки и перезаписи данных. Например, RainForest [5] не требует сортировки значений атрибутов и формирования списков атрибутов. Объем основной памяти, требующийся для хранения структур данных алгоритма (так называемых AVC групп), пропорционален числу различных значений атрибутов. Но так как количество числовых атрибутов обычно очень велико, то в алгоритме RainForest предлагается подход, основанный на SPIES (Statistical Pruning of Intervals for Enhanced Scalability) [5] - разбиению на интервалы области значений каждого атрибута, нахождения наименее вероятных интервалов, которые могут содержать точку для разбиения и их отбрасывание. В данном подходе не нужна сортировка исходного обучающего множества, объем хранимых в основной памяти структур данных и поток обмена данными минимальны. В алгоритме BOAT [6] за счет итеративного построения дерева полный проход по обучающему множеству делается только 2 раза для построения нескольких уровней дерева решений, что сильно сокращает вычислительные затраты. Для построения дерева на первой итерации используется небольшое подмножество исходного множества данных. Алгоритм позволяет хранить в оперативной памяти основные для построения данные, динамически заменяя их на каждой итерации.

Целью настоящей работы является построение такого параллельного алгоритма, который оптимальным образом разбивал бы исходную задачу на такие подзадачи, которые могли бы независимо выполняться, а их время выполнения было бы равномерно сбалансированного между всеми процессорами. Принимается во внимание природа исходных данных и алгоритм решения прикладной задачи локализации нейронных источников, предложенный в [2]. Опираясь на традиционные схемы параллельного построения деревьев решений, предлагается способ «быстрого построения» множества деревьев-классификаторов, каждое из которых формирует вероятную зону локализации, в зависимости от количества доступных ресурсов вычислительной системы. Разработанный алгоритм адаптирован для вычислительных систем с распределенной памятью.

2. Постановка задачи локализации нейронных источников и ее сведение к задаче классификации

Пусть имеется экспериментальные сигналы ЭЭГ, представляющие запись электрического потенциала $U_{\exp}$ в ряде точек на поверхности головы. Часто используется сферическая модель головы. В этом случае нам дан набор данных $U_{\exp}(\theta_i, \varphi_i, t^n)$. Этот потенциал моделируется потенциалом, который создается внутренними нейронными источниками, представленными здесь электрическими диполями внутри головы. Требуется подобрать такое расположение диполей, которое наилучшим способом аппроксимирует экспериментальные данные.

Внутренняя пространственная трехмерная область покрывается сеткой в сферической системе координат. Шаги сетки $h_r = 1/N_r$, $h_\theta = 1/N_\theta$, $h_\varphi = 1/N_\varphi$. Где N_r, N_θ, N_φ - число точек по каждому из направлений. В каждом узле сети располагается диполь. Каждый диполь характеризуется 6 признаками: три пространственные координаты и три составляющие плотности тока (моменты диполя). Строится трехмерная сетка в области моментов диполей в сферической системе координат, где вектор υ характеризуется тремя параметрами: $\upsilon_r = |\upsilon|$, $\upsilon_\theta = \cos\theta$, $\upsilon_\varphi = \cos\varphi$. Шаги сетки $N_{\upsilon_\theta}, N_{\upsilon_\varphi}$, сетка по параметру υ_r привязана к данному сигналу ЭЭГ. Образ представляет собой точку в шестимерном пространстве. Например, это может быть диполь, расположенный в одном и том же месте, но обладающий разными моментами. Всего образов $N_{DB} = N_r N_\theta N_\varphi N_{\upsilon_\theta} N_{\upsilon_\varphi} N_{\upsilon_r}$.

Обозначим вектор признаков образа через

$$\mathbf{X}^p = \{x_1, x_2, \ldots x_i, \ldots x_M \mid C_k\}^p \tag{1}$$

где p — номер примера, k — номер класса, x_i^p — i-й признак. Для каждого p-го образа вычисляется ошибка по потенциалу, международное обозначение — RRE (Residual Relative Error).

$$\varepsilon^p = \sum_i \sum_j [(U_{\exp}(\theta_i, \varphi_j) - V_K(\theta_i, \varphi_j)] - w^{(p)}(\mathbf{x}^p \mid \theta_i, \varphi_j)]^2 \sin\theta_i h_{\theta_i} h_{\varphi_j} \tag{2}$$

Более подробно постановка задачи обсуждается в работе [10]. Задаем уровень допустимой ошибки ε_{th}, и для каждого момента времени t разделяем параметры диполей на два класса: ниже порога,

когда $\varepsilon^p < \varepsilon_{th}$ (первый класс — C_1) и выше порога (второй класс — C_2). Таким образом, формируется база данных для классификации в один момент времени. Целью является нахождение таких диполей, которые минимизирует ошибку (2).

Основная идея предлагаемого метода локализации источников состоит в решении задачи минимизации методом классификации источников на основе специально сформированной базы данных этих источников. Если число анализируемых временных срезов N_t, то полная база данных содержит $N_t * N_{DB}$ данных. Анализ всей базы данных одним деревом нецелесообразен, поэтому для классификации предлагается построение ансамбля случайных деревьев. Каждый образ в базе данных — это диполь, характеризуемый набором параметров, включающий его координаты. Полное анализируемое множество образов разбивается на независимые подмножества соответствующие определенному моменту времени в ЭЭГ сигнале. Каждое такое подмножество состоит из набора образов - диполей, расположенных некоторым способом внутри пространственной области, назовем его обучающим множеством. В каждый момент времени имеется набор определенного числа диполей внутри области. Далее эти диполи делятся на два класса — C_1, C_2. К классу C_1 относятся те диполи, которые дают ошибку в аппроксимации экспериментальных данных меньшую некоторого заданного порога ε_{th} по потенциалу ε_{th} [10]. К классу C_2 относятся все остальные. Для решения задачи классификации и построения классификатора используется ансамбль случайных деревьев решений. Каждое дерево в ансамбле анализирует пространственные данные (локализации диполя) для определенного момента времени и окончательное решение вырабатывается некоторой процедурой голосования. Задача классификатора состоит в том, чтобы найти наиболее вероятные зоны расположения хороших источников и выбрать среди них лучшего представителя.

3. Параллельный алгоритм построения ансамблей деревьев решений

После формирования обучающего множества формируется ансамбль деревьев решений. Связь каждого дерева и локализуемой области описана в работе [2]. Алгоритм построения одного дерева решений соответствует схеме, относящейся к классу традиционной C4.5 [7]. Поэтому здесь не будут приводиться все детали алгоритма.

Исследования, относящиеся к алгоритмическим особенностям предлагаемого подхода, описаны в работе [2].

Рассмотрим общий алгоритм построения множества деревьев решений — комитета — на вычислительной архитектуре с распределенной памятью, учитывая специфику задачи локализации. При построении ансамбля деревьев решений, каждое дерево (член ансамбля) может быть построен независимо. Взаимодействие всех деревьев необходимо только на последнем этапе построения ансамбля - выработки функции голосования. Голосование деревьев занимает от 7-10% от общего времени решения задачи локализации, поэтому на данном этапе будет исследована только часть алгоритма эффективного решения обратной задачи локализации, а именно этап построения деревьев решений из обучающего множества большого размера.

Так как каждое дерево строится независимо, но объем обучающей выборки превышает локальную память процессорного блока в системе, то необходим метод распределения деревьев между процессорами. Предлагается следующая процедура. Пусть есть множество свободных процессоров P_{free}, а N_t - количество деревьев решений, которые нужно построить для формирования ансамбля. Тогда группа процессоров P_{free} разбивается на блоки по $(P_{free} / N_t) \bmod N_b$, где N_b — количество процессоров в каждом блоке. После этого построение каждого дерева выполняется на одном блоке процессоров.

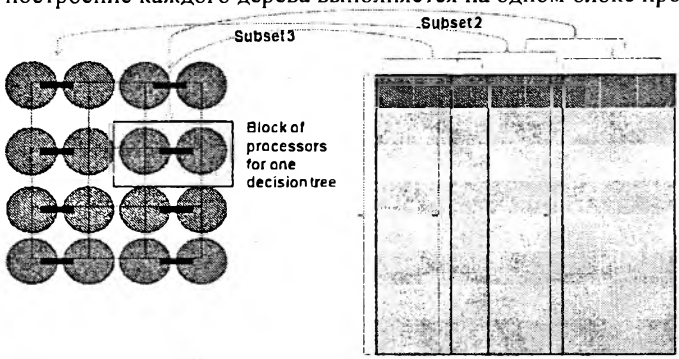

Рисунок 2. Пример распределения примеров обучающего множества между двухпроцессорными блоками.

На рисунке 2 показан пример распределения обучаюшего множества между двухпроцессорными блоками, где каждое дерево будет построено на 2-х процессорах. Ниже будет описан алгоритм построения дерева решений для каждого многопроцессорного блока, каждый из которых выполняется независимо при построении ансамбля.

Мы использовали общие схемы хранения временных "быстрых" структур данных, которые используются в традиционных алгоритмах параллельного построения деревьев решений. В качестве таких структур выступают хэш-таблицы [6], содержащие основные значения распределений классов, значения индексов разбиения, другую необходимую информацию о дереве, или же структуры-списки [8], содержательно представляющие то же, что и хэш-таблицы, однако для работы с ними используются другие методы (обход списка, вставка элемента, удаление, обновление и др.). В этой работе реализован метод формирования "быстрых" структур-списков. Предположим, что количество процессоров в блоке есть степень 2-х и они представляют собой топологию гиперкуба (соответственно алгоритм может быть модифицирован, если общее число P процессор в блоке не степень двойки). Шаги параллельного алгоритма:

1. База данных обучающих примеров дерева решений равномерно распределена среди P процессоров. Таким образом, если N - общее число обучающих примеров в подмножестве для данного дерева решений, то у каждого процессора в локальной памяти будет N/P примеров.
2. Изначально все процессоры принадлежат одному узлу N_0 и работают вместе для вычисления точки разбиения.
    a. Каждый процессор для каждого атрибута, находящегося у него в памяти, вычисляет распределение классов и записывает его в свою копию списков классов (хэш-таблицы).
    b. Каждый процессор обменивается данными о распределении классов с другими процессорами посредством записи в глобальную редукционную переменную (хэш-таблицу или список).
    c. Одновременно процессоры вычисляют энтропию или индекс Гини для каждого атрибута узла и выбирают наилучший атрибут для разбиения на подмножестве, а потом и на всем обучающем множестве посредством записи в редукционную переменную.
    d. Создание потомков рассматриваемого узла, разбиение тренировочного множества на подмножества-потомки.
3. С увеличением глубины дерева, объем собираемой статистики на каждом уровне увеличивается. На каком-то уровне временная стоимость обмена информацией между процессорами становится чрезмерно большой. Тогда в этом случае, группа процессоров, работающих в блоке над каждым узлом, разбивается на 2

подгруппы, которые выполняют построение поддеревьев параллельно.

   a. В гиперкубе, каждый из 2-х подгрупп процессоров соответствует подкубу. Сначала, соответствующие процессоры 2-х подкубов обмениваются нужными данными. После обмена у каждого из подкубов есть вся информация для построения следующего узла дерева. Число обучающих примеров на каждом процессоре может быть от 0 до $2N/P$, каждая подгруппа процессоров балансирует количество узлов на процессор, распределяя обучающие примеры.

4. Далее шаги алгоритма, начиная с п.1 повторяются для созданных подгрупп процессоров, которые могут работать независимо.
5. Если число процессоров в группе становится меньше двух, то эта группа объединяется с группой процессоров, которая имеет такое же число процессоров.

Ключевой момент в данном алгоритме - критерий разбиения группы работающих процессоров на подгруппы. Если делать разбиение слишком часто или редко, то будет неэффективно тратиться время на обмен информации между процессорами. Оптимальная точка разбиения достигается, когда время на передачу информации равно времени пересылки примеров между процессорами на стадии поиска точки разбиения узла.

$$S_{com} \geq M_c + L_c \qquad (3)$$

где S_{com} — общие временные затраты на обмен данными, M_c — затраты на обмен данными между блоками процессоров, L_c — затраты на обмен данными между процессорами одного блока. Дальнейший анализ эффективности предложенного алгоритма будет приведен ниже.

4. Анализ масштабируемости и производительности предложенного алгоритма

Для анализа производительности алгоритма сделаем следующие предположения:

1. Процессоры образуют топологию гиперкуба.

2. Все уравнения, описывающие затраты на передачу данных, написаны для бинарного дерева с 2^L листами на глубине L.

3. Мы предполагаем, что размер дерева решений асимптотически не зависит от размера входного обучающего множества.

Обозначим через A_d — число атрибутов. Тогда для каждого уровня дерева существует A_d таблиц распределения классов, которыми должны обменяться процессоры. Размер каждой из таблиц — произведение числа классов и числа значений атрибутов. Отсюда размер таблицы распределения классов на каждый процессор для каждого узла построения есть

$$S = C * A_d * M \tag{4}$$

где C — число классов, A_d — число атрибутов, M — число различных значений атрибутов (или пороговые значения, позволяющие восстановить значения атрибутов). Число узлов (листов) на уровне есть 2^L, где L — рассматриваемый уровень построения дерева решений. Тогда полный размер таблицы T_s определяется следующим образом

$$T_s = C * A_d * M * 2^L \tag{5}$$

На уровне L, полное время вычислений C_{pl}, включай проходы по обучающему множеству, создание и обновление таблиц распределений классов для каждого атрибута равно

$$C_{pl} = \theta\left(\left(\frac{A_d * N}{P}\right) + C * A_d * M * 2^L\right) t_c = \theta\left(\frac{N}{P}\right) \tag{6}$$

где N — общее число обучающих примеров, P — общее число процессоров.

После локальных вычислений на каждом процессоре выполняется синхронизация процессоров, используя глобальную редукционную переменную — общую таблицу классов распределений. Тогда затраты на передачу Cm_l можно оценить следующим образом

$$Cm_l = (t_s + t_w * C * A_d * M * 2^L) * \log(P_i) \le \theta(\log(P)) \tag{7}$$

где t_s — единица измерения времени вычислений, t_w — временная задержка при передаче одного пакета данных.

Когда группа процессоров разбивается на 2 подгруппы, каждый узел приписывается к подгруппе таким образом, чтобы число обучающих примеров в каждой подгруппе было примерно одинаковым. Для того чтобы подгруппы работали независимо, необходимо, чтобы в каждой подгруппе было необходимое обучающее множество примеров. Чтобы система была полностью сбалансирована необходимо, чтобы на каждом процессоре было N / P примеров обучающего множества. Данные взаимодействия осуществляются в 2 этапа, первый — перемещение каждым процессором необходимых примеров другому

процессору, следовательно, каждый процессор может принять и отправить максимум N/P примеров процессору другой подгруппы. Тогда затраты на синхронизацию C_{mov} можно оценить как

$$C_{mov} \le 2 * \frac{N}{P} * t_w \qquad (8)$$

Второй этап — балансировка внутри каждой подгруппы. К моменту синхронизации подгрупп у каждого процессора может быть от 0 до $2*N/P$ примеров, каждый процессор может получить и отправить максимум N/P примеров. Предполагая отсутствие перегрузок сети, балансировка C_{bal} занимает

$$C_{bal} \le 2 * \frac{N}{P} * t_w \qquad (9)$$

Если же предполагать, что существуют некие перегрузки и задержи в сети, тогда затраты на балансировку подгруппы могут быть оценены [8], [9].

Для анализа масштабируемости задачи мы будем использовать очень полезную метрику, которая обычно применяется при анализе большого числа задач для решения на параллельных коммерческих вычислительных системах. Она определяется следующим образом. Пусть P это число процессоров, а W это размер задачи (то есть полное время выполнения последовательного алгоритма задачи). Если W должно расти как $F_{E(P)}$, для поддержания эффективности W, то тогда $F_{E(P)}$ определяется как изо-эффективная функция для эффективности E и график $E(P)$ является кривой изо-эффективности (постоянной эффективности) для эффективности E.

Предположим, что для построения классификационной модели для базы данных было построено дерево глубиной L_1. Глубина дерева это константа, не зависящая от размера обучающего множество, если данное дерево решений аппроксимирует данные. Затраты на создание новых подгрупп процессоров Cp_{part} — это произведение всего числа подгрупп и затрат на передачу данных для полного формирования одной группы равно $\theta\left(\frac{N}{P}\right)$ (8),(9). Общее число разбиений, в котором может участвовать процессор при построении всего дерева, равен его глубине - L_1.

$$Cp_{part} \le L_1 * \theta\left(\frac{N}{P}\right) \quad (10)$$

Затраты на обмен данными на каждом уровне дерева описываются уравнением (8)(и равны $\theta(\log P)$). Общее время передачи данных при построении всех уровней $Call_l$ описывается уравнением

$$Call_l \le L_1 * \theta(\log P) = \theta(\log P) \quad (11)$$

Цена всех коммуникаций между процессорами C_{total} — это сумма затрат на создание разбиения группы на подгруппы и затрат на обработку таблиц распределений классов

$$C_{total} = \theta(\log P) + \theta\left(\frac{N}{p}\right) \quad (12)$$

Общее время выполнения T_{total} равно

$$T_{total} = \theta\left(\frac{N}{P}\right) \quad (13)$$

Время выполнения параллельного построения $T_{par} = C_{total} + T_{total}$.

$$T_{par} = \theta(\log P) + \theta(\frac{N}{P}) \quad (14)$$

В случае последовательного выполнения программы, проход по обучающему множеству выполняется один раз при построении каждого уровня и время на построение равно

$$T_{ser} = \theta(N) * L_1 = \theta(N) \quad (15)$$

Для подсчета эффективности использования параллельных вычислений приравняем время выполнение последовательной программы и время выполнения параллельной программы, умноженное на количество использованных процессоров.

$$\theta(N) = P * \theta(\log P) + \theta(\frac{N}{P}) \quad (16)$$

Таким образом, мера эффективности равна $\theta(\log P)$, учитывая, что перегрузок или других задержек, связанных с коммуникационной сетью, нет. Это говорит о том, что аналитически мы можем сказать, что задача построение ансамбля деревьев решений является масштабируемой задачей с ростом процессоров.

5. Результаты обработки сигнала ЭЭГ с помощью параллельного алгоритма построения деревьев решений

Анализировались реальные данные ЭЭГ, снятые с пациентов во время определенного эксперимента. Так как данное исследование направлено на изучение масштабируемости параллельного алгоритма построения ансамблей деревьев решений при обработке данных ЭЭГ, сформированных в базу данных специальным образом, то полное описание экспериментов испытуемых, первичную обработку электроэнцефалограммы, выборки окон и временных точек можно найти в работах [1], [2]. Здесь приведен анализ исходных обучающих множеств для построения деревьев решений, их размерности и некоторые характеристики.

Таблица 1. Характеристики баз данных для обучения ансамбля деревьев решений.

№ базы данных в таблице	Кол-во зон для локализации	Число атрибутов	Число примеров во множестве
1	1	6	7^6
2	1	6	12^6
3	2	12	7^{12}
4	2	12	12^{12}
Число деревьев в ансамбле	Формирование базы данных (% от общего времени выполнения)	Формирование ансамбля (% от общего времени выполнения)	Голосование (% от общего времени выполнения)
250	54	31	15
500	58	35	7
250	62	25	13
500	69	19	12

В Таблице 1 описано несколько характеристик баз данных, которые использовались при построении ансамбля деревьев решений. Параметры формирования обучающих множеств - количество примеров в обучающем множестве, количество деревьев решений при построении ансамбля, количество зон локализации, параметры сетки при решении прямой задачи [10]. Исследовались две задачи нахождения одной зоны активности, при нескольких параметрах сетки внутри головы, и две задачи нахождения 2-х зон активности также при разных параметрах сетки. В таблице отражено количество деревьев, построенное в данном ансамбле и число обучающих примеров в базе данных. В первой части

данного параграфа будут приведены результат работы параллельного алгоритма построения одного дерева решений - члена комитета. Во второй части будут показаны результаты построения ансамбля деревьев решений при их параллельном исполнении.

6. Анализ построения ансамбля деревьев решений

Алгоритм построения ансамбля деревьев решений опирается на распределение всего количества процессоров по группам и построение своей группой процессоров одного члена-дерева ансамбля. Время выполнения построения классификатора зависит от размерности обучающего множества.

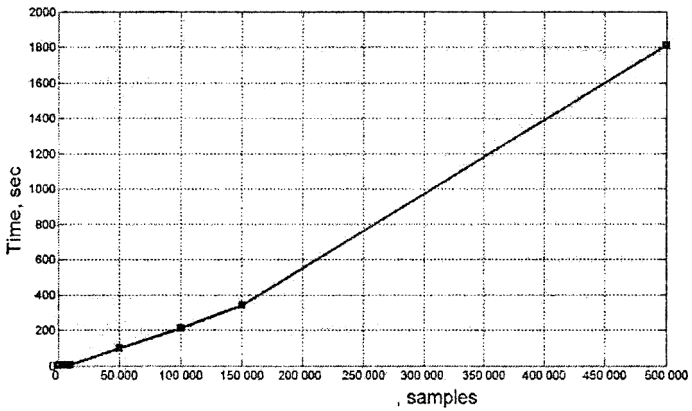

Рисунок 3. Зависимость времени выполнения последовательной программы — Time — от размера входного множества — N.

На рисунке 3 показана зависимость времени выполнения последовательной программы от размера входного множества. Ключевым параметром при распределении деревьев является количество процессоров в группе. Из экспериментальных исследований следует, что параллельный алгоритм построения одного дерева является масштабируемым и адаптируем к числу процессоров. Конечно, наше исследование ориентировано на массивно-параллельные архитектуры, с тысячами узлов. Аналитически показано, что эффективность может увеличиваться с ростом процессоров. Например, для построения ансамбля деревьев решений для базы данных, приведенной в таблице, нужно построить 500 деревьев. Каждое дерево, как показано экспериментально, может быть эффективно построено вплоть до 16 процессоров, а теоретически и намного больше. Таким образом, минимум процессорных узлов, необходимых для построения такого ансамбля, нужно 8000 (если увеличить число процессоров до 100-200, то можно еще больше ускорить построение). Как показано в работе [2],

точность задачи локализации можно увеличивать, уточняя входные параметры прямой задачи и параметры ансамбля. Эти характеристики сильно увеличивают входное обучающее множество, поэтому использование больших вычислительных массивно-параллельных комплексов будет сильно сокращать временные затраты. Ниже приведены результаты ускорения выполнения параллельного алгоритма и его масштабируемости.

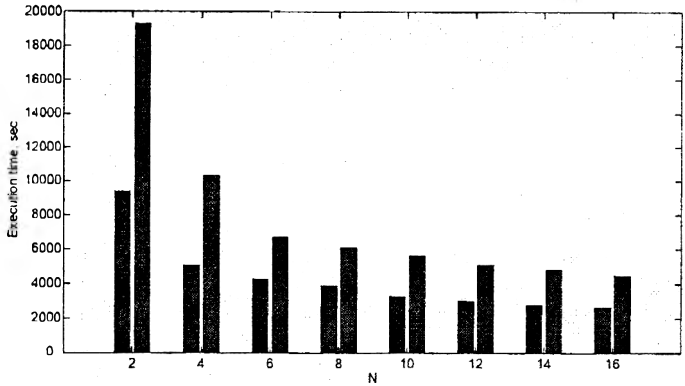

Рисунок 4. Ускорение времени построения полного ансамбля деревьев решений — Speedup — для различных тренировочных множеств Таблицы 1.

На рисунке 4 представлены результаты построения ансамбля деревьев решений для баз данных, описанных в таблице, с распределением по 4 процессора на группу.

7. Анализ построения одного дерева решений

При анализе масштабируемости рассматриваемой задачи построения ансамбля деревьев решений (17), было выявлено, что необходимо найти оптимальную точку при построении дерева для выполнения разбиения группы процессоров на независимые подгруппы. Также было показано, что эта точка характеризуется отношением

$$R = \frac{\sum C_{com}}{C_{mov} + C_{bal}} \qquad (17)$$

На рисунке 5 показано ускорение времени работы алгоритма построения одного члена ансамбля при разбиении группы процессоров на подгруппы на каждом уровне построения дерева, при отсутствии разбиения на всех этапах и при разбиении в зависимости от значения порога. Результаты показали, что теоретические выводы, сделанные при анализе в предыдущем параграфе, верны. Когда разбиение выполняется слишком часто, размер поддерева сложно предсказать на этапе

построения, то алгоритм начинает терять эффективность, когда число процессоров возрастает от 8 до 16. Затраты на передачу данных и постоянное обновление необходимых хэш-таблиц очень сильно сказывается на времени выполнения. При отсутствии каких-либо разбиений вообще, эффективность повышается при работе 2-х процессоров в группе, после этого начинает снижаться из-за увеличения буферов обмена данными и количества узлов для построения на большой глубине дерева. Из рисунка видно, что при оптимальном пороге разбиения, можно наблюдать ускорение времени выполнения задачи. Выявить аналитическим путем оптимальный порог разбиения достаточно сложно. Приведем результаты обработки конкретных исследуемых обучающих множеств и экспериментальное определение оптимального порога разбиения. По некоторым оценкам, приведенным в работах [12]-[14], интервал нахождения оптимального значения отношения (17) лежит около единицы. Мы исследовали время выполнения задачи для нескольких значений порога разбиения, лежащих около единицы.

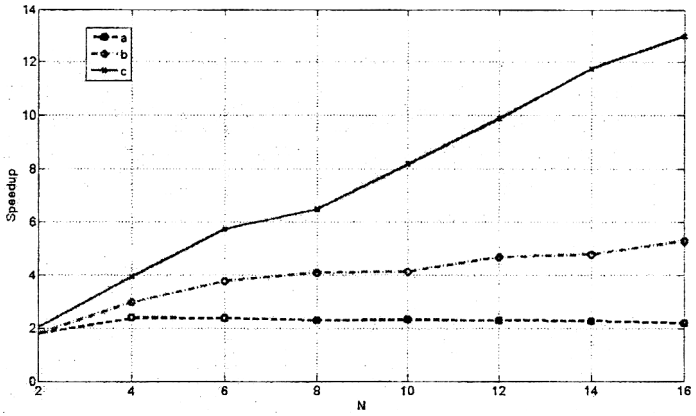

Рисунок 5. Среднее ускорение времени работы алгоритма — Speedup — построения одного члена ансамбля при разбиении группы процессоров на подгруппы: a) при отсутствии разбиения на всех этапах b) на каждом уровне построения дерева, c) при разбиении в зависимости от значения порога.

На рисунке 6 показаны зависимости среднего времени выполнения задачи построения одного дерева решений от порога для разбиения процессоров на подгруппы. Из рисунка видно, что действительно минимум достигается приблизительно около 1 и близок к оптимальному значению, указанному в других работах [12], [17].

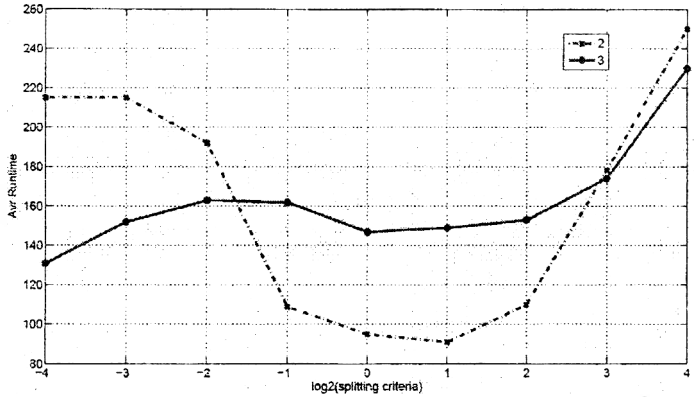

Рисунок 6. Зависимости среднего времени выполнения задачи Av_{TIME} построения одного дерева решений от порога для разбиения процессоров на подгруппы — Split.

На рисунке 7 показано ускорение работы предложенного алгоритма параллельного построения дерева решений для различных обучающих множеств, описанных в Таблице 1. При изучении масштабируемасти задачи, мы фиксировали одно обучающее множество, состоящее из 7 миллионов примеров, и увеличивали количество процессоров.

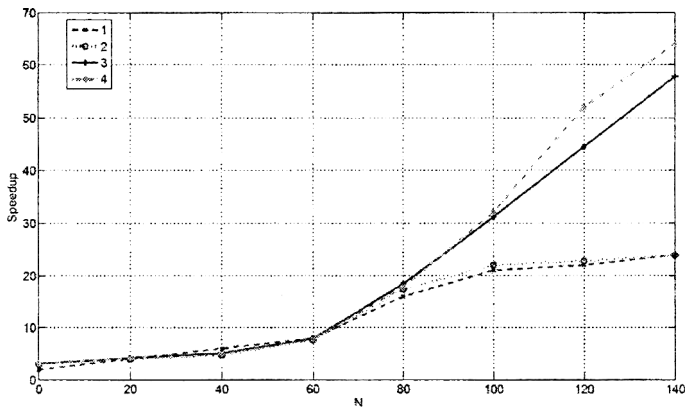

Рисунок 7. Ускорение работы предложенного алгоритма Speedup параллельного построения дерева решений для различных обучающих множеств, описанных в Таблице 1.

Результат, представленный на рис.8, показывает, что мера эффективности построения одного дерева решений равна $O(P \log P)$.

Данные эксперименты не охватывают все возможные случаи формирования базы данных ЭЭГ-сигнала и возможные отклонения будут предметом дальнейших исследований.

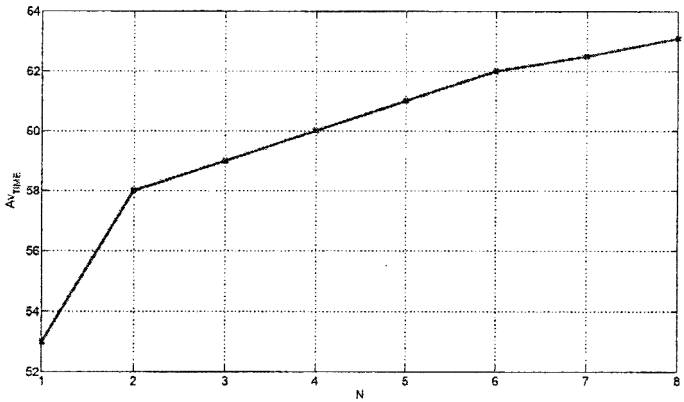

Рисунок 8. Масштабируемость задачи — Scaleup — построения дерева решений.

8. Выводы

Из многочисленных исследований алгоритма построения деревьев решений [7], [16], [17] можно указать основные параметры, отвечающие дереву такие как: максимальная глубина, количество узлов, количество листьев, сбалансированность. Существуют оценки сложности деревьев решений, и такие понятия как «сила» дерева и др.[7]. Ключевым параметром предложенного параллельного алгоритма является пороговое значение для разбиения группы процессоров на подгруппы. Оно оценивается исходя из соотношения (17), которое учитывает временные затраты на обмен данными и выполнение построения дерева, но не берет во внимание «оценку» дерева решений (риск, глубину, количество узлов и т.д.) на этапе его построения. Можно предполагать, что априорная оценка размерности, сложности дерева решений, его влияние на распределение задач по процессорам может значительно улучшить эффективность, при выполнении условия, что оценка дерева занимает незначительные затраты по времени.

В настоящем исследовании проведен полный анализ производительности и масштабируемости алгоритма построения ансамбля деревьев решений для задачи локализации нейронных источников мозга. Исследование показало, что задача масштабируется с ростом процессоров и может быть оптимизирована по времени за счет параметров разбиения процессоров на подгруппы. Исследование было проведено для массивно-параллельных архитектур, получивших широкое развитие в последнее время. В дальнейшем, планируется

модифицировать оптимизационные параметры задачи и привязать их к характеристикам формируемого дерева. Необходимо также исследовать максимальную загруженность каждого процессора при построении ансамбля для прогнозирования максимального количества используемых алгоритмом ресурсов вычислительной системы.

Литература

1. EEG and MEG: Forward. J.C. Mosher, R.M. Leahy and P.S. Lewis. 1999, IEEE Trans. Biomed. Eng., pp. 245-259.
2. Ансамбль деревьев решений для локализаии нейронных источников мозга. Е.А., Попова. 2007, М.: Вестник МГУ, pp. 23-35.
3. SLIQ: A fast scalable. M. Mehta, R. Agrawal, and J. Rissanen. Avignon, France : s.n., 1996. the Fifth Int'l Conference on Extending Database Technology.
4. SPRINT: A scalable. J. Shafer, R. Agrawal, and M. Mehta. 1996. In Proc. of the 22nd.
5. Communication and Memory Efficient Parallel Decision Tree Construction. R.Jin, G. Agrawal. 2002. In Proc. of 2002 Internaitonal COnference on Parallel DM.
6. K. Alsabti, S. Ranka, and V. Singh. CLOUDS: Classification for large or out-of-core datasets. 1998.
7. Random Forests. L.Breiman. 2001, Machine Learning, pp. 5-32.
8. Catlett, J. Megainduction: Machine Learning on Very Large Databases. Sydney : University of Sydney, 1991.
9. Metalearning for multistrategy learning and parallel learning. Stolfo, Philip K. Chan and Salvatore J. 2003. In Proc.Second Intl. Conference on Multistrategy Learning. pp. 150-165.
10. Численное решение обратных математических задач электроэнцефалографии. К.Хоффманн, А.М.Попов, И.А.Федулова, С.Е.Певцов. 2004, Вестник МГУ, pp. 16-27.
11. С, Певцов. Разработка методов решения обратных задач, возникающих в биомедиуине. Москва : Москва, 2007.
12. Parallel Formulations of Decision-Tree Classification Algorithms. A.Srivastava, E.Han, V. Kumar, V. Singh. 2002. In Proc. of 2001Internaitonal COnference on Parallel Processing.
13. ScalParC: A new scalable and efficient parallel classification alg orithm for mining large datasets. M.V. Joshi, G. Karypis, and V. Kumar. 1998. In Proc. of the International Parallel Processing Symposium.
14. V. Kumar, A. Grama, A. Gupta, and G. Karypis. Introduction to Parallel Computing: Algorithm Design and Analysis. Redwod : Benjamin Cummings/ Addison Wesley, 1994.
15. Unstructured tree search on simd parallel computers. Kumar, G. Karypis and V. 1994, Parallel and Distributed Computing, pp. 379-391.

16. Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. Monterey, Calif. : U.S.A.: Wadsworth, 1984.
17. Bagging predictors. Breiman, L. 1996, Machine Learning, pp. 123-140.

Иванов С.А.

Методы поиска закономерностей в символьных последовательностях*

Введение

Последовательности являются одними из классических объектов математики и их различные виды: функциональные, числовые, строковые, символьные, графовые, рассматриваются практически во всех современных её разделах. В прикладных задачах очень часто объектом исследования являются символьные последовательности. Их анализ позволяет получить ответы на различные вопросы в широком спектре научных областей — от теоретического программирования до биологии и прикладной лингвистики. Последовательности этого вида возникают в таких задачах, где рассматриваются объекты, допускающие декомпозицию на некоторое число фрагментов более простой структуры, позволяющих выделить среди фрагментов одинаковые. Причём «одинаковость» фрагментов носит достаточно условный характер, поскольку определяется из решаемой проблемы человеком, проводящим исследование. Например, при анализе поведения некоторого животного, различные действия могут считаться «одинаковыми». Так же, в генетическом коде человека выделяются так называемые «некодирующие участки»[1)], состоящие из различных генов, однако практически всегда гены, находящиеся на таких участках, не представляют интереса для исследования и считаются «одинаковыми». Фрагменты, полученные при декомпозиции объекта исследований, кодируются при помощи символов некоторого алфавита и, таким образом, первоначальный объект представляется просто как символьная строка. Единственное условие, которому должен удовлетворять кодирующий алфавит — мощность алфавита[2)] должна быть больше или равна количеству различных полученных фрагментов. При таком подходе конкретная область исследований, из которой была получена задача, не имеет принципиального значения, будь то исследование поведения животного или исследование ДНК человека, поскольку все объекты будут в конечном итоге представлены в виде строк из символов кодирующего алфавита, анализ которых и будет производиться. При

* Работа выполнена при поддержке гранта РФФИ № 05-07-90238.

1) Считается, что данные участки генетического кода человека не несут информации о строении человека.

2) Под мощностью алфавита понимается количество символов, из которых он состоит.

анализе может учитываться некоторая экспертная информация из предметной области, которая может быть представлена различными способами. Например, экспертная информация может представляться неравнозначностью символов кодирующего алфавита[3)], так же может использоваться некоторая информация о символьных подстроках, например, об их возможном взаимном расположении[4)]. Целью подобных исследований часто является выделение внутренней структуры кодирующей последовательности, позволяющей сделать некоторые выводы о структуре исходного объекта исследований.

Подходы к пониманию термина «Структура»

В исследованиях термин "структура" понимается по-разному. И это приводит к различному его определению. В основном рассматривают три варианта: композиционная иерархическая структура, таксономическая структура и структура как набор функциональных зависимостей. Рассмотрим каждый из этих вариантов подробнее.

Таксономическая структура

В данном случае структура представляется в виде иерархии отношений вида «класс — представитель класса», то есть каждая сущность более низкого уровня является частным случаем сущности более высокого. При данном подходе дочерние узлы образуют множества, поэтому дуги в графе идущие от одной вершины не упорядочены (или идущие к одной вершине), и две различные вершины графа могут соединяться только одной дугой.

Данная структура представляет собой качественный состав последовательности, и её построение представляет собой классификацию образцов, входящих в последовательность.

Существуют различные подходы к решению задачи построения подобной структуры последовательности, некоторые из них описаны в [2].

[3)] Например, в последовательностях представляющих ДНК человека вероятность появления различных букв кодирующего алфавита различна.

[4)] Примером этому может послужить последовательность дествий животного. Если рассматривать собаку, то можно утверждать что при достаточно большой частоте фиксации действий, подпоследовательность, описывающая животного в состоянии сна, не может идти непосредственно перед подпоследовательностью, описывающей бег собаки, очевидно, что должна быть некоторая промежуточная подпоследовательность.

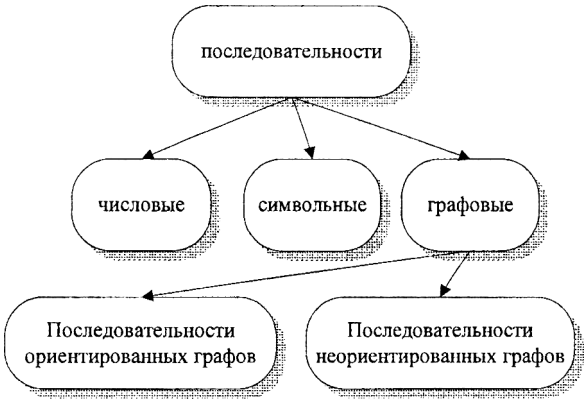

Рисунок 1. Пример таксономической иерархии.

Композиционная иерархическая структура

При таком подходе структура представляет собой иерархию, в которой высокоуровневые сущности представляют собой комбинации низкоуровневых элементов. Типичным примером такой структуры является словосочетание: словосочетание состоит из слов, слова состоят из слогов, слога состоят из букв, которые находятся на самом низком уровне иерархии.

Композиционные иерархические структуры обычно представляются в виде направленного ациклического мультиграфа с дугами, идущими от родительских узлов к дочерним. Дуги упорядочены, причём порядок определяется порядком декомпозиции родительской сущности. В качестве листьев выступает популяция атомов[1)]. Уровнем выше находятся некоторые комбинации входных элементов. Каждый последующий уровень содержит сущности, являющиеся агрегациями сущностей более низких уровней.

[1)] При рассмотрении символьных последовательностей атомами или входными элементами данных будут символы кодирующего алфавита.

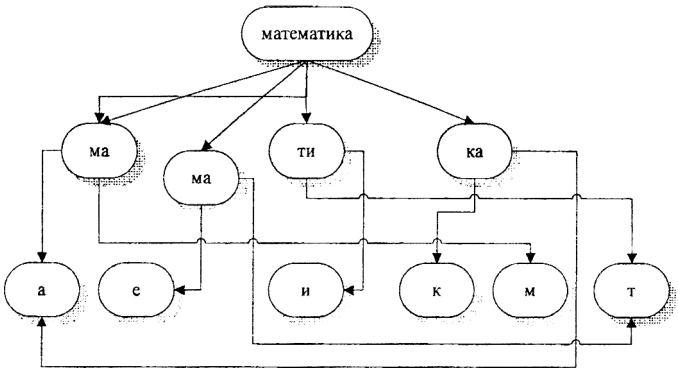

Рисунок 2. Пример композиционной иерархии.

Данный подход к пониманию "структуры" достаточно подробно рассматривается в работе [1], где предлагаются два варианта построения решения поставленной задачи — рекуррентные нейронные сети и модифицированный метод n-грамм.

Структура как набор зависимостей

Это понимание структуры кажется наиболее понятным и логичным. В этом подходе целью исследований является нахождение набора правил, позволяющих определять очередную букву по её предшественницам. Предположим, что дана некоторая последовательность букв:

$a_1, a_2, a_3 ... a_k$, причём $a_i \in \sum$ при $i = \overline{1..k}$, где $\sum$ некоторый алфавит[1)].

Примерами подобных правил могут служить:

- если на позиции L последовательности стоит символ "a", то на позиции L+3 будет стоять символ "c";
- если на позиции L последовательности стоит символ "b", а на позиции L+2 стоит символ "c", то на позиции L+4 будет стоять символ "a".

Так же очередной символ может определяться нечётким множеством, например:

- если на позиции L последовательности стоит символ "a", то на позиции L+3 будет стоять "примерно символ "c"", то есть символ на позиции L+3 будет принадлежать нечёткому множеству, например такому {0,2/"a"+0,3/"b"+1/"c"}.

[1)] Например, можно рассматривать английский алфавит.

Подобная структура символьной последовательности позволяет сделать вывод о структуре объекта предметной области, анализ которого производится. Выделение подобных зависимостей – задача достаточно часто встречающаяся на практике, в таких областях как биоинформатика, нейрофизиология и прочие. Решению данной задачи посвящено много исследований. Статистические методы нахождения структуры символьной последовательности как набора зависимостей достаточно подробно рассмотрены в [3]. Классический метод построения подобной структуры, основанный на переборе возможных шаблонов, рассматривается в работах [4] и [5].

Одной из задач биоинформатики является определение структуры ДНК человека. Первичная структура ДНК — это линейная последовательность нуклеотидов в цепи, которые разделяются на группы по азотистым основаниям четырёх видов:

1. Гуанин, обычно обозначается символом "G".
2. Аденин, обычно обозначается символом "A".
3. Тимин, обычно обозначается символом "T".
4. Цитозин, обычно обозначается символом "C".

Таким образом, ДНК человека представляется последовательностью символов из алфавита {"A", "C", "G", "T"}, называемого обычно генетическим. Очень часто этот алфавит расширяют символами, описывающими подмножества символов генетического алфавита, и рассматривают расширенный алфавит {"A", "C", "G", "T", "W", "R", "K", "S", "Y", "M", "N"}[3)]. В исследованиях рассматриваются соответствия исходной строки ДНК подстрокам из расширенного генетического алфавита. Считается, что подстрока соответствует строке из расширенного алфавита, если символ, стоящий на i-ой позиции подстроки, принадлежит множеству, описываемому i-ым символом подстроки расширенного алфавита. Например:

- подстрока "ACGT" удовлетворяет "ANST",
- подстрока "AAAA" удовлетворяет "AAAW" и "AWAA".

Зависимости в нуклеотидных цепочках ДНК человека помогают определять аномалии, которые в дальнейшем могут привести к развитию заболеваний, к которым человек генетически расположен. Из-за того, что за последнее время достигнуты значительные успехи в диагностике и профилактике инфекционные и вирусных заболеваний, более серьёзно стало заметно влияние генетических факторов. Например, в Канаде, согласно статистическим данным, у 5% населения в возрасте до 25 лет обнаруживаются наследственные дефекты, приводящие к инвалидности, а у более чем 50% в течение жизни

[3)] Значения символов расширенного алфавита: "W"="A" или "T", "R"="A" или "G", "K"= "G" или "T", "S"= "C" или "G", "Y"= "C" или "T", "M"="A" или "C", и "N"= любой из "A","C","G","T".

развивается заболевание, имеющее в той или иной степени наследственную природу. В настоящее время более половины случаев обращения в детские лечебные учреждения связано с генетическими заболеваниями [7]. Так, например, в геноме человека довольно часто встречаются микро и мини сателлиты. Микро и мини сателлитами называются *тандемные повторы*. Тандемным повтором называют последовательность нуклеотидов, которую можно представить, как некое слово, повторяющееся одно за другим без делеций и вставок символов[1)], но с возможными ошибками. Найдено уже 14 неврологических заболеваний вызванных экспансией сателлитов. Более полную информацию о микро и мини сателлитах можно получить в [6].

Исследованию решения задачи поиска структуры последовательности, как набора зависимостей посвящено много исследований. Существуют два основных подхода к решению данной задачи, основанных на переборе возможных вариантов: алгоритм Магнуссона и статистический подход. Эти подходы будут описаны в работе ниже. Они достаточно активно используются в современных исследованиях. Однако их применение к задачам, подразумевающим исследование очень длинных символьных последовательностей[2)], становится проблематичным из-за большого времени анализа, вызванного необходимостью перебора возможных вариантов зависимостей. Целью данной работы является разработка метода определения структуры последовательности, основанного на кластеризации подстрок определённой длины. Строки, образующие кластер, определяют шаблон[3)], который в свою очередь и определяет зависимости между символами последовательности. Таким образом, зависимости строятся исходя из построенных кластеров, что позволяет избежать перебора возможных зависимостей, и делает метод более подходящим для анализа очень длинных последовательностей, чем алгоритм Магнуссона и статистические алгоритмы. Метод, представляемый в данной работе использует кластеризацию подстрок. Кластеризация – это очень популярный способ выделения общего у группы объектов.

Обычно выделяют два типа кластеризации: иерархическую кластеризацию и декомпозицию. При декомпозиции множество всех объектов разбивается на подмножества, каждый объект принадлежит только лишь одному подмножеству. При иерархической кластеризации множества, которым принадлежит объект, образуют иерархию, в

[1)] В специализированной литературе под делецией символа подразумевают удаление.

[2)] Например, ДНК человека, кодируются последовательностью длиной 3 миллиарда символов.

[3)] Понятие шаблона вводится в следующей главе.

которой большее по размеру множество состоит из некоторого количества меньших по размеру множеств. Оба подхода достаточно часто используются на практике для получения информации о структуре взаимосвязей исследуемых объектов.

Целью кластеризации является построение отображения, которое по анализируемому объекту выдаёт номер кластера, к которому данный объект относится. Изучению методов кластеризации посвящено много исследований, построено много методов, среди наиболее популярных методов: K-средних, X-средних[1)], генетический алгоритм, самоорганизующиеся карты Кохонена, иерархические алгоритмы и прочие подходы. Метод представляемый в данной работе подразумевает разбиение всех подстрок на некоторое предопределённое количество кластеров. В данной работе рассматривается применение методов K-средних и генетических алгоритмов для построения подобного разбиения.

Постановка задачи поиска структуры в последовательностях

Формализация основных понятий

Для чёткой формализации понятия "структура", рассматриваемого в данной работе, необходимы базовые определения. Одним из них является понятие шаблона.

Определение: Пусть задан алфавит $\sum = \{a_1, a_2, a_3, ..., a_t\}$ и пусть символ $"-" \notin \sum$. Тогда алфавит $\sum_1 = \{a_1, ..., a_t, "-"\}$ называется расширенным, а алфавит $\sum$ — основным.

Определение: Шаблоном длины L — это упорядоченная последовательность из L символов расширенного алфавита.

Пусть исходный алфавит $\sum = \{a, b, c\}$, тогда расширенным алфавитом будет $\sum_1 = \{a, b, c, "-"\}$. Тогда шаблонами длины 3 будут: "abc", "a-b", "a--", "---". Примерами шаблонов длины 4 могут служить: "abbb", "a--b", "ac--".

Определение: Пусть шаблон T длины L такой, что $\exists\ 0 \le N < L : \forall i = 0..N-1$ T[L-i] = "-", тогда для фиксированного $0 \le m \le N$ m-сдвигом вправо шаблона T называется

[1)] Метод X-средних представляет собой модификация метода K-средних, позволяющий подбирать количество кластеров автоматически.

шаблон S такой, что: $\forall i = m+1..L$ S[i] = T[i-m] и $\forall i = 1..m$: S[i] = "–".

Определение: Пусть шаблон T длины L такой, что $\exists\ 0 \le N < L : \forall i = 0..N-1$ T[i] = "–", тогда для фиксированного $0 \le m \le N$ m-сдвигом влево шаблона T называется шаблон S такой, что: $\forall i = 1..L-m$ S[i] = T[i+m] и $\forall i = 0..m-1$:S[L-i] = "–".

Определение: Шаблон S длины L называется сдвигом шаблона T длины L, если $\exists\ 0 \le m < L$ такое, что S является m-сдвигом вправо или влево шаблона T.

Нетрудно заметить, что шаблон совпадает со своим 0-сдвигом вправо и 0-сдвигом влево.

Определение: Подстрока S длины L соответствует шаблону T длины L, если существует шаблон T_1 — сдвиг шаблона T такой, что для $\forall i = 1..L$ верен следующий предикат:

$$\{T_1[i] \equiv "-"\} OR \{T_1[i] = S[i]\}.$$

Определение: Функцией качества шаблона T длинной L относительно последовательности S называется отношение количества встреченным подстрок в последовательности, удовлетворяющих шаблону T, к его априорной оценке, полученной в предположении, что символы в последовательности распределены независимо. То есть функция качества определяется формулой:

$$F(T,S) = \frac{A}{P(T) \cdot M},$$ где A — количество подстрок последовательности S, соответствующих шаблону T, M — количество подстрок длиной L в последовательности S, $P(T)$ — вероятность того, что L подряд идущих произвольных символов последовательности S соответствуют шаблону T.

Определение: Шаблон T определяет зависимость в последовательности S для некоторого порогового значения α, если значение функции качества шаблона T относительно последовательности L превышает пороговое значение.

Определение: Шаблон называется вырожденным, если количество символов исходного алфавита, содержащихся в этом шаблоне меньше 2.

Поясним смысл понятия вырожденности шаблона более конкретно. Предположим, что шаблон T длины L вырожден и определяет зависимость в последовательности S. Следовательно, в данном шаблоне может либо стоять только один символ основного алфавита, либо не стоять ни одного такого символа вообще. Если в шаблоне отсутствуют символы основного алфавита, то это значит, что

любая подпоследовательность длины L исходной последовательности будет соответствовать шаблону. То есть такому шаблону удовлетворяет любая последовательность, а значит, такой шаблон не несёт в себе никакой информации. Если в шаблоне T стоит всего лишь один символ основного алфавита, то это даёт только лишь информацию о частоте встречи данного символа в последовательности, что тоже не составляет интереса. Если же шаблон не вырожден, то информация о том, что строка S удовлетворяет шаблону, уже говорит о взаимном расположении символов в последовательности, что соотвстствует пониманию структуры как набора зависимостей.

Определение: Структурой последовательности называется множество невырожденных шаблонов, которым удовлетворяет данная последовательность с заданным пороговым значением α [1].

Постановка задачи поиска структуры как набора зависимостей в последовательностях

Задана последовательность символов S алфавита $\sum = \{a_1, a_2, a_3, ..., a_t\}$:

$$x_1, x_2, x_3, ...x_n, x_i \in \sum .$$

Для заданного порогового значения α и целых положительных k и l необходимо найти k шаблонов длины l[2], которым последовательность S соответствует с наибольшим значением функции качества шаблона относительно строки S.

Более формально можно представить задачу следующим образом:

$$\sum = \{a_1, a_2, a_3, ..., a_t\}, \qquad S = x_1, x_2, x_3, ...x_n, x_i \in \sum ,$$

α — значение порога,

$$M = \{T \mid T - шаблон\ длины\ L,\ F(T,S) \geq \alpha\},$$

необходимо найти:

---

[1] Предполагается, что исследования структуры происходят всегда при заданном значении α, то есть оно всегда определено как некоторое константное значение.

[2] Данную формулировку следует понимать следующим образом: если количество шаблонов, определяющих зависимости в последовательности S, не меньше k, то найти k шаблонов, иначе найти все шаблоны, определяющие зависимости в последовательности S.

$$Arg\left(\underset{\substack{M_i \subset M,\ T \in M_i \\ |M_i| \le k}}{Sup} \sum F(T,S)\right)$$

Алгоритм Магнуссона

Еще одним подходом к решению данной задачи является алгоритм Магнуссона. Данный алгоритм был предложен в 2000 году исландским исследователем для анализа скрытых шаблонов поведения животных. Более подробно можно ознакомиться с алгоритмом в статье Магнуссона [4]. В последнее время было предпринято несколько попыток улучшить данный алгоритм. Одна из таких попыток описана в [5].

Для детального пояснения алгоритма Магнуссона вводится понятие критического интервала.

Определение: критическим интервалом относительно символа C исследуемой последовательности для символа D называется пара положительных целых чисел m и n*, таких, что если в исследуемой последовательности символ C встречается на позиции L, то на позициях L+n…L+m встретится как минимум один символ D.

Определение: длиной критического интервала считается значение m-n+1.

Определение: минимальный критический интервал – критический интервал с минимальной длиной.

В критический интервал (m,n) попадают все символы исследуемой последовательности, отстоящие от ближайшего предшествующего символа C на количество символов, которое не превосходит числа n и не меньше числа m[1)].

Определение T-pattern.

Предположим, что исследуется последовательность символов из алфавита $\sum = \{a_1, a_2, ...a_n\}$. Тогда T-pattern являются конструкции следующего рода:

1. T = a_i, где $a_i \in \sum$

---

* Обычно считается, что $m \le n$, в противном случае интервал получается пустым и не сможет содержать хотя бы один символ D и тем самым не будет критическим интервалом.

1) То есть если символ C стоит в последовательности на позиции L, то все символы, находящиеся на позициях L+n…L+m, попадут в критический интервал.

2. T = T_1, d_1, d_2, T_2, где T_1 и T_2 являются T-pattern-ами, а d_1 и d_2 — положительными числами, $d_1 \le d_2$.

Критический интервал для паттернов определяется аналогично критическому интервалу для символов[2)].

Основная идея алгоритма состоит в иерархическом построении t-pattern-ов по исходной последовательности. Первоначально рассматривается взаимное расположение символов алфавита. Для каждой пары символов (C, D) алфавита делается попытка построить минимальный критический интервал относительно символа C для символа D. Если построить такой критический интервал невозможно[3)], то делается заключение о том, что символ D не зависит от символа C. Если минимальный критический интервал построен, то анализируются вхождения символа D в критический интервал. Первоначально считается, что символы в последовательности распределены независимо с частотой равной: $\frac{N_{символа}}{N_{общее}}$, где $N_{символа}$ - количество вхождений символа в последовательность, а $N_{общее}$ - общее количество символов в последовательности. Исходя из соотношений частоты вхождения символа D в критический интервал с его априорной оценкой[4)], делается вывод о том, что вхождение символа D зависит от вхождения символа C и зависимость определяется критическим интервалом. Зависимость определят t-pattern $T = Cd_1d_2D$, где числа d_1 и d_2 определяют критический интервал для символа C относительно символа D.

После рассмотрения всех комбинаций символов алфавита получается набор t-pattern-ов. Далее производятся аналогичные рассуждения для комбинаций паттернов, полученных на предыдущем шаге друг с другом, паттернами, полученными на более ранних шагах, и символами алфавита. Таким образом, получаются паттерны следующего шага.

Построение паттернов происходит до тех пор, пока не станет невозможным построить ни одного нового паттерна. В результате работы алфавита получается набор паттернов, организованных в

[2)] Для этого нужно лишь в определении критического интервала для символов заменить все вхождения слова “символ” на слово “t-pattern”.

[3)] Простейшим примером, когда построение критического интервала невозможно может служить отсутствие символа D в последовательности после символа C.

[4)] Априорная оценка делается исходя из начального предположения, описанного ранее.

иерархическую структуру, каждый из которых описывает структуру как набор функциональных зависимостей.

Рассмотрим шаг определения зависимости между двумя символами исходной последовательности более подробно на примере.

Пусть, дана последовательность, состоящая из символов алфавита {“A”, “B”, “C”, “D”}:

AABAABABABABABCDDDDAABABDBABCDBADBCBAB

DDDDDABDCADBCDDCDADBCDBCBDBCBDCDCCDC

Рассмотрим зависимость между символами “A” и “B” в последовательности.

Легко видеть, что минимальным критическим интервалом будет (1,2). Априорная вероятность появления символа “B” в последовательности:

$$P_b = \frac{N_b}{N_{общее}} = \frac{21}{76} \approx 0{,}276315 .$$

Таким образом, количество символов “B”, попавших в критический интервал – 17, а априорная оценка: $N_{apriority} = 17 \cdot 0{,}47628 \approx 8$. Видно, что априорная оценка более чем в два раза меньше результата, значит символы “A” и “B” находятся в зависимости, определяемой паттерном: $T = A,1,2,B$.

Можно заметить, что понимание шаблона, рассматриваемое в данной работе, соответствует t-pattern-у, в котором во всех критических интервалах начало критического интервала совпадает с концом, то есть паттерну, в котором всегда $d_{1,i} = d_{2,i}$ для всех $d_{1,i}$ и $d_{2,i}$ входящих в паттерн. Таким образом, алгоритм Магнуссона позволяет найти структуру исследуемой последовательности в понимании, рассматриваемом в данной работе. Однако сложность алгоритма Магнуссона по времени, как это показано в [5], зависит от длины анализируемой последовательности и длины искомых зависимостей, что делает проблематичным использование данного алгоритма на очень длинных последовательностях, например, таких как геном человека, чья длина составляет более трёх миллиардов нуклеотидов. Так же метод Магнуссона подразумевает хранение всей входной последовательности, что так же проблематично при анализе очень длинных последовательностей. Время работы алгоритма Магнуссона пропорционально квадрату длины искомой зависимости и длине исследуемой строки, что делает проблематичным использование данного метода в задачах подобного рода.

Данный подход применяется при анализе последовательностей программным средством THEME. Данное программное средство было

создано под руководством Магнуссона, и является реализацией его алгоритма. Программное средство THEME, в основном, применяется в психологии и нейрофизиологии для исследования последовательностей, описывающих последовательность действий животного.

Построение кластерного решения задачи поиска структуры последовательности, как набора зависимостей

Современные методы осуществляют прямое построение шаблонов, которым удовлетворяет последовательность, посредствам перебора возможных вариантов. В данной работе предложен альтернативный метод построения шаблонов длины L, основанный на анализе подстрок длины L. Рассмотрим его более подробно.

Представление шаблона кластером подстрок

Предположим, что анализируется символьная последовательность S:

$$S = x_1, x_2, x_3, \ldots x_n, \text{ где } x_i \in \sum = \{a_1, a_2, a_3 \ldots a_m\}.$$

Предположим, что нам известно, что в последовательности S шаблон T длины L определяет зависимость. Тогда, шаблон T определяет некоторое множество B подстрок длины L, удовлетворяющих данному шаблону. Однако множество B может соответствовать нескольким шаблонам, определяющим зависимость в исследуемой последовательности S. Покажем это на простом примере.

Пусть задана последовательность из символов алфавита $\sum = \{A, B, C, D\}$:

$$ADBDADBDBCBCBDADBDADCDCBADBBDCADADACA.$$

В этой последовательности шаблону $T = "A--D"$ соответствует множество подстрок длины 3: $\{"ADBD", "ADCD", "ADAD"\}$. Нетрудно видеть, что данное множество так же соответствует шаблону $T' = "AD-D"$. В зависимости от выбранного порогового значения α [1], как шаблон T', так и шаблон T могут определять зависимости в исследуемой последовательности S. В указанном примере шаблон T' более точен,

---

[1] Значение порога задаётся заранее и используется для определения того, задаёт ли шаблон зависимость в исследуемой символьной последовательности. Здесь и далее считается, что шаблон определяет зависимость в символьной последовательности согласно определению данному в главе 2.

чем шаблон T и поэтому более интересен для определения структуры последовательности. Но возможны и не столь простые случаи. Например, если получившееся множество подстрок совпадает с $\{"ABDCA","BACDC","AABDC"\}$, то шаблонами, определяющими зависимость в последовательности S, будут: $T_1 = "A - D - -"$ и $T_2 = "B - C - -"$.

Множество подстрок B определяет множество M_B шаблонов, определяющих зависимость в последовательности S, таких, что для любого $T_i \in M_B$ любая подстрока из B соответствует T_i. Так как известно, что T определяет зависимость, то шаблон T также будет принадлежать множеству M_B. Рассмотрим шаблон: $T_B = Arg(\sup_{T_i \in M_B} F(T_i, S))$. Из того, что $T \in M_B$ непосредственно следует, что $F(T,S) \le F(T_B,S)$. Будем называть шаблон T_B уточнением шаблона T, а множество B' подстрок анализируемой последовательности S, соответствующих шаблону T_B, уточнением множества B. Из построения понятно, что $B \subseteq B'$. Множества могут совпадать. Операция уточнения шаблона увеличивает значение функции качества шаблона, то есть по заданному шаблону строит шаблон, описывающий лучше зависимости в символьной последовательности.

Если множество B совпадает со своим уточнением, то шаблон T, определяющийся множеством B, совпадает со своим расширением. То есть шаблон T не может быть улучшен операцией расширения. Таким образом, построение n шаблонов, определяющих зависимости в исследуемой последовательности, соответствующих n наибольшими значениями функции качества, может быть представлено как построение множеств, определяющих эти шаблоны, для каждого из которых, его уточнение совпадает с самим множеством.

Рассмотрим более подробно подобные множества. Каждое из них представляет собой некоторое подобие кластера – поскольку является множеством, элементы которого обладают общим свойством – соответствием шаблону, определяемому самим множеством. Однако рассматриваемые множества имеют несколько существенных отличий от кластеров. Первым является то, что не все подстроки заданной длины попадут в подобные множества. Так же такие множества могут пересекаться, то есть одна подстрока может соответствовать нескольким шаблонам.

Предположим, что задано множество непересекающихся множеств подстрок, каждое из которых совпадает со своим уточнением,

то есть: $B = \{B_1, ...B_n\}$, такое, что $B_i \equiv B_i'$ [1] и $B_i \cap B_j = 0$ при условии $i \neq j$. Каждое из этих множеств определяет шаблон T_{B_i}, как показано выше. Разобьём множество всех подстрок исследуемой последовательности S на множества M_i такие, что множество M_i содержит подстроки, для которых ближайшим является шаблон T_{B_i}, где расстояние от шаблона до подстроки считается равным минимальному количеству замен, которое необходимо произвести в подстроке для соответствия подстроки шаблону. Подобное разбиение множества всех подстрок соответствует разбиению на кластеры. Однако требование того, чтобы не пересекались множества $B_1 ... B_n$, ведёт к тому, что определяемые ими n шаблонов, могут не быть шаблонами с n наибольшими значениями функции качества. Поэтому при данном подходе получается не точное решение, а его некоторое приближение. Построению кластерного разбиения посвящено множество исследований. Одними из популярных методов являются методы K-средних и метод, основанный на применении генетического алгоритма, описание применения которых к решению рассматриваемой задачи ниже.

Формирование кластеров подстрок методом K-средних

Подход определения структуры символьной последовательности, представленный выше подразумевает кластеризацию подстрок определённой длины как декомпозицию всего множества подстрок на k подмножеств, где k – количество правил, которые необходимо получить[2]. Алгоритм K-средних один из наиболее простых и эффективных, позволяющих разбить исходное множество объектов на k кластеров.

Рассмотрим более подробно этот алгоритм. Каждый кластер представляется своим центром, и распределение объектов по кластерам происходит в зависимости от соотношений расстояний от объекта до центров кластеров. А именно, объект относится к кластеру, расстояние до центра которого минимально. Таким образом, для распределения объектов по кластерам необходимо лишь знать центры кластеров.

Центры кластеров определяются итерационным алгоритмом. Первоначально они определяются либо произвольным образом, либо,

---

[1] B_i' - множество, уточняющее множество B_i.

[2] см. главу 2.

исходя из самих анализируемых объектов[3]. Далее происходит итеративное уточнение положения центров кластеров. А именно, для каждого кластера определяют принадлежащие ему объекты и переопределяют центр кластера таким образом, чтобы сумма квадратов расстояний от центра кластера до принадлежащих ему объектов была минимальной. Алгоритм останавливается либо по прошествии некоторого заранее определённого количества итераций[4], либо, когда получаемые центры кластеров перестают сдвигаться.

Основной целью данного алгоритма является минимизация суммарного квадратичного отклонения объектов от центров кластеров, к которым они принадлежат. То есть, если происходит разделение n объектов $X_1, X_2, X_3, ...X_n$ по k кластерам $K_1, K_2, K_3, ...K_k$, с центрами $\mu_1, \mu_2, \mu_3, ...\mu_k$, и заданы расстояния $\rho(\mu_i, X_j)$, то происходит минимизация функционала: $\sum_{i=1}^{k} \sum_{X_j \in K_i} \rho^2(\mu_i, X_j)$.

Рассмотрим предлагаемое в данной работе применение идеи метода K-средних к задаче поиска зависимостей в символьных последовательностях. Прежде всего, определим центры кластеров. Центрами кластеров будем считать шаблоны, задача поиска которых решается.

Определим более формально, чем в предыдущей главе, расстояние от шаблона до подстроки[2].

Пусть задана некоторая подстрока s из символов алфавита $\sum$ длины L и некоторый шаблон T. Тогда элементарным расстоянием от подстроки s до шаблона T будем называть количество позиций, для которых занимающий их символ последовательности, не соответствует символу шаблона T. То есть количество позиций i таких, что: $(s[i] \neq T[i])\ and\ (T[i] \neq "-")$. Расстоянием от шаблона T до

---

[3] Обычно пытаются выбрать центры первоначальных кластеров таким образом, чтобы разнести их друг от друга на как можно большее расстояние.

[4] Обычно это делают тогда, когда критичным является время работы алгоритма. Тогда довольствуются некоторым промежуточным результатом, который может быть достаточно далёк от результата, но в какой-то степени отражает структуру взаимосвязей между исследуемыми объектами.

[2] Определение понятия расстояния происходит не с остальными определениями в главе 2, а здесь, поскольку это понятие используется только в алгоритме K-средних.

подстроки s будем называть минимальное из элементарных расстояний между сдвигами шаблона T и подстрокой s.

Таким образом, мы определили центры кластеров, расстояние от объекта до центра кластера. Для полного определения метода K-средних осталось лишь определить метод построения уточнения центра кластера по подстрокам, принадлежащим кластеру. Уточнение центра кластера B предлагается определять как шаблон, обладающий максимальным значением функции качества среди шаблонов, которым соответствуют некоторые[3)] подстроки, принадлежащие данному кластеру.

Предположим, что центр кластера определяется шаблоном T. Из определения расстояния от шаблона до подстроки видно, что расстояние от шаблона до подстроки, удовлетворяющей шаблону, рано нулю. Таким образом, все строки, удовлетворяющие шаблону, будут заведомо отнесены к кластеру K. Далее из способа уточнения центра кластера следует, что функция качества для шаблона T', являющегося уточнением центра кластера K, будет больше или равна значению функции качества для шаблона T. Таким образом, суммарное значение функции качества для центров всех кластеров не уменьшается на последующих шагах алгоритма.

Алгоритм останавливается, когда центры кластеров перестают сдвигаться. В данный момент каждый из центров кластеров характеризуется не уточняемым шаблоном. Таким образом, строятся шаблоны, каждый из которых не может быть улучшен операцией уточнения.

Формирование кластеров подстрок генетическим алгоритмом

Генетический алгоритм — это подход к решению оптимизационных задач, имитирующий эволюционный процессы развития решения. Предположим, что нам необходимо решить следующую оптимизационную задачу:

$$X_m = Arg(\max_{X \in B} F(X))$$

Каждый элемент множества B может оказаться искомым решением. Множество B представляет собой множество потенциальных решений. Каждое потенциальное решение (элемент множества B) кодируется в виде генома – последовательности генов. Ограничений того, чем может являться ген, нет, поэтому ген может представлять собой натуральное число, букву или что-то другое. Обычно геном служат числа 0 или 1, таким образом, каждое потенциальное решение

[3)] Возможно не все.

кодируется как последовательности 0 и 1. Исследователь сам выбирает то, каким образом кодировать потенциальные решения.

Для геномов определяется функция качества, позволяющая оценить решение. Обычно чем больше значение функции качества для генома, тем больше решение, кодируемое геном, соответствует искомому результату.

Первоначально инициализируется множество геномов, называющееся популяцией.

Далее, на каждом шаге из уже существующих геномов строятся новые. По двум произвольно выбранным геномам, называемым родительскими, посредствам их комбинаций и мутаций строятся два новых генома, называемых дочерними. Под комбинацией понимается то, что новый геном строится из частей родительских. Под мутацией понимается случайное (обычно с некоторой заданной вероятностью) изменение одного или нескольких генов результата комбинации двух старых геномов.

После порождения новых геномов происходит процесс отбора, во время которого погибают, то есть исключаются из рассмотрения, геномы с наименьшим значением функции качества. Таким способом поддерживается постоянный размер[1)] рассматриваемой популяции.

Процесс порождения новых геномов и отбора лучших представителей продолжается до тех пор, пока значение функции качества не будет удовлетворять некоторому условию[2)], либо не будет исчерпано максимально возможное количество итераций, которое определяется заранее.

Рассмотрим применение данного метода к решению задачи поиска зависимостей в символьных последовательностях путём кластеризации подстрок.

Возможное решение представляет собой разделение подстрок исходной последовательности на k множеств. Пусть в исследуемой строке S существует ровно m различных подстрок длины l, где l – длина искомого шаблона, определяющего зависимость. Пронумеруем все подстроки в алфавитном порядке числами от 1 до m. Представим возможное решение массивом целых чисел, в котором на позиции i стоит номер кластера, к которому относится подстрока с номером i.

[1)] Размер популяции – количество геномов в популяции. Формально размер должен не превышать некоторого заранее определённого числа. Но на практике обычно размер популяции оставляют равным этому числу.

[2)] Например, нас может интересовать не точка, в которой достигается абсолютный максимум, а точка значение в которой превосходит некоторого значения.

Таким образом, геном определяет разбиение подстрок исследуемой последовательности по кластерам. Каждый кластер характеризуется его центром, который вычисляется как шаблон, обладающий максимальным значением функции качества среди шаблонов, которым соответствуют некоторые[1)] подстроки, принадлежащие данному кластеру. Предположим, что геном Γ определяет кластеры $K_1, K_2, ..., K_k$ с центрами $\mu_1, \mu_2, \mu_3, ...\mu_k$, функций качества генома, будем считать:

$$F(\Gamma) = \sum_{i=1}^{k} F_{gen}(\mu_i, S),$$

где F_{gen} — это функция качества шаблона μ_i относительно кластера K_i, и

$$F_{gen} = \frac{A}{P(\mu_i) \cdot M},$$

где A – количество подстрок кластера K_i, соответствующих шаблону μ_i, M — количество подстрок длиной L в исследуемой последовательности S, $P(T)$ — вероятность того, что L подряд идущих произвольных символов последовательности S соответствуют шаблону T.

Выводы

Основной проблемой существующих методов поиска зависимостей в последовательностях является то, что они предполагают построение этих зависимостей путём перебора анализа их возможных вариантов. Это делает временную сложность алгоритмов построенных на основе существующих подходов зависимой от длины анализируемой последовательности, что приводит в свою очередь к проблематичности применения данных алгоритмов к решению задач, предполагающих анализ последовательностей больших размеров. В данной работе предложены методы, дающие приближённое решение рассматриваемой задачи. Предложенные методы рассматривают входную последовательность как набор подстрок, что делает их независимыми от размера входной последовательности и применимыми для решения задач подразумевающих анализ последовательностей большой длины.

[1)] Возможно не все.

Литература

1. Karl R. Pfleger, "On-line learning of predictive compositional hierarchies" Ph.D. dissertation. Стэндфордский университет, 2002, 247 с.
2. Оссовский С. Нейронные сети для обработки информации. М.: Финансы и статистика, 2004, 344 с.
3. Bakeman R., Deckner D. F., Quera V. Analysis of Behavioral Streams // Handbook of research methods in developmental psychology. Oxford, UK: Blackwell Publishers, 2005, P. 394–420.
4. Magnusson M.S. Discovering Hidden Time Patterns in Behavior: T-Patterns and their Detection // Behavior Research Methods, Instruments and Computers. 2002. 32. N 1. P. 93-100.
5. Jiong Yang, Wei Wang, Philip S. Yu Mining Asynchronous Periodic Patterns in Time Series Data // IEEE Transactions on Knowledge and Data Engineering. 2003. 15. N 3. P. 613-628.
6. В.А. Боева, М.В. Фридман, В.Ю. Макеев "Эволюция микро- и минисателлитов в геноме человека." // Биофизика. 2006. 51. № 4. С. 650-655.
7. Глик Б., Пастернак Дж. "Молекулярная биотехнология". М.: МИР, 2002, 589 с.

Нестеров С. В.

Исследование ошибок измерения в задаче усвоения метеоданных со спутников-скаттерометров[1]

1. Введение

В области численного предсказания погоды прогноз базируется на процедуре усвоения данных. Усвоение данных об исследуемом объекте производится с целью коррекции состояния модели объекта. В области прогнозирования погоды одним из таких объектов является земная атмосфера, а данные представляют собой показания температуры, давления, скорости и направления ветра, влажности и т.п., полученные с различных измерительных приборов.

1.1. Процедура усвоения данных

В основе процедуры усвоения лежит итеративный процесс, в наиболее общем случае реализующий фильтр Калмана и действующий по принципу – «прогноз – коррекция». Подробно о задаче усвоения можно прочитать в работе [1].

В упрощенном виде этот процесс можно описать следующим образом:

- Сначала строится некоторая модель наблюдаемого объекта. Эта модель зависит от некоего набора параметров $\mathbf{X}$.
- В каждый момент времени $\mathbf{n}$ строится предварительный прогноз $\mathbf{X}_n^b$ по предыдущему состоянию модели $\mathbf{X}_n^b = \mathbf{F}(\mathbf{X}_{n-1})$. Прогноз делается с некоторой ошибкой ε относительно истинного состояния наблюдаемой среды $\mathbf{X}_n^t$:
  $\mathbf{X}_n^b = \mathbf{X}_n^t + \varepsilon$; матрица ковариации ε – $\mathbf{B}$.
- Производится замер параметров объекта $\mathbf{Y}_n$ с некоторой ошибкой η: $\mathbf{Y}_n = \mathbf{H}(\mathbf{X}_n^t) + \eta$, где $\mathbf{H}$ – оператор проекции состояния модели на измеряемые параметры; матрица ковариации η – $\mathbf{R}$.
- С учетом предварительно прогноза $\mathbf{X}_n^b$ и полученных измерений $\mathbf{Y}_n$ методом максимального правдоподобия строится новое состояние модели $\mathbf{X}_n$ – величина, максимизирующая $\mathbf{P}(\mathbf{X}_n^t \mid \mathbf{Y}_n)$.

Поиск $\mathbf{X}_n$ ведется исходя из следующих формул (временные индексы опущены для удобства восприятия):

$\mathbf{P}(\mathbf{X}|\mathbf{Y}) \sim \mathbf{P}_a(\mathbf{X})\mathbf{P}(\mathbf{Y}|\mathbf{X})$, где $P_a(X)$ – априорная вероятность состояния X среды.

---

[1] Работа выполнена при поддержке гранта РФФИ № 05-07-90238.

$$P(Y|X) = P(H(X) + \eta = Y|X) = P(\eta = Y - H(X)|X) = P_\eta(Y-H(X))$$
$$P_\eta(Y-H(X)) \sim \exp[-1/2 * (Y-H(X))^T R^{-1}(Y-H(X))].$$

$$P_a(X) = P(X^b - \varepsilon = X) = P(\varepsilon = X^b - X) = P_\varepsilon(X^b - X)$$
$$P_\varepsilon(X^b - X) \sim \exp[-1/2 * (X - X^b)^T B^{-1}(X - X^b)].$$

Таким образом, требуется минимизировать функционал:

$$J = -\ln P(X|Y) \sim (X - Xb)TB\text{-}1(X - Xb) + (Y\text{-}H(X))TR\text{-}1(Y\text{-}H(X)). \quad (1)$$

При вычислении функционала использовались плотности распределения ошибок измерения и прогнозирования. При этом использовались предположения фильтра Калмана о том, что эти ошибки распределены по нормальному закону и взаимно независимы.

При таком виде функционала он минимизируется за одну итерацию методом Ньютона. Но в случае невыполнения одного из предположений формула усложняется, и процесс минимизации может также заметно усложниться.

Данная процедура применяется и для усвоения метеоданных в задаче прогнозирования погоды. Одним из новых и перспективных источников данных для последующего усвоения являются спутники - скаттерометры. Они позволяют получить данные о приморских и океанических ветрах. Ветер является одним из важнейших параметров, от которых зависит состояние атмосфера, а моря и океаны занимают большую часть поверхности земного шара. Проверка выполнимости для скаттерометров предположений, на которых базируется фильтр Калмана, представляет собой весьма важную задачу, так как Гидрометцентр России планирует использовать скаттерометры в качестве источника данных для своей системы усвоения.

1.2. Скаттерометрия

Основными принципами работы скаттерометров являются следующие:

- Спутник излучает микроволновую энергию.
- Длина волны примерно 5см.
- Данное излучение почти не взаимодействуют с атмосферой.
- Волны на водной поверхности отражают излучение с равной длиной волны.
- Волны на водной поверхности создаются ветром на высотах до 10м.
- По отраженному от волн сигналу путем решения обратной задачи можно судить о ветре, их породившем.

Спутники-скаттерометры обладают рядом неоспоримых преимуществ по сравнению с кораблями и буями, которые до этого были основным средством получения информации о приводных

морских и океанических ветрах, как-то: наблюдение может вестись почти в режиме реального времени, большие возможности по охвату территории и разрешению получаемых данных.

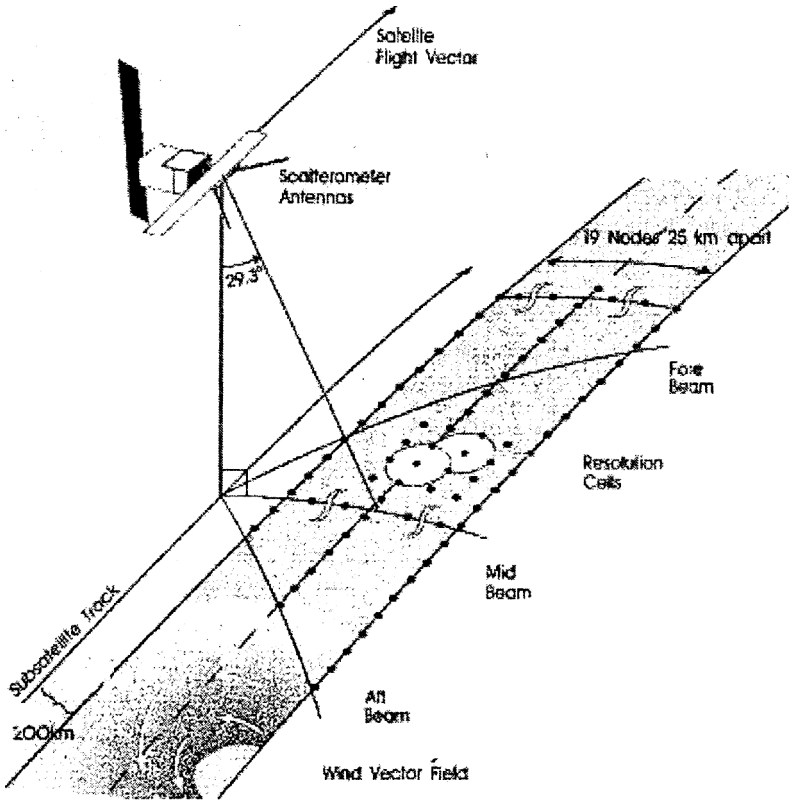

Рисунок 1. Схема работы спутника — скаттерометра.

Самым новым из действующих спутников-скаттерометров является принадлежащий NASA спутник QuickScat. Его показания транслируются через соответствующие каналы передачи метеоданных по всему миру. Гидрометцентр планирует в ближайшее время начать использовать данные с этого спутника для получения актуального состояния атмосферы в районах поблизости от обширных водных поверхностей.

1.3. Контактные данные

В распоряжении ГидроМетЦентра имеется база так называемых контактных данных. Это замеры температуры, давления, скорости и направления ветра и т.д., сделанные в некоторый момент времени с плывущего корабля или стационарного буя. Закон

распределения ошибок таких измерений для компонент вектора скорости известен и может быть выражен явной формулой:

$$P(\bar{u}) = (2\pi)^{-m/2}|\sum|^{1/2}\exp[-1/2*(\bar{u} - \acute{\eta})^{T}\sum^{-1}(\bar{u} - \acute{\eta})],$$

где $\acute{\eta}$ – математическое ожидание, а $\sum$ - матрица ковариации ошибок.

Так как рассматриваются 2-хмерные данные и известно, что ошибки контактных измерений имеют нулевое мат. Ожидание, то данное выражение принимает вид:

$$P(\bar{u}) = (2\pi)^{-1}|\sum|^{1/2}\exp[-1/2*(\bar{u})^{T}\sum^{-1}(\bar{u})],$$

где матрица $\sum$ является диагональной с одинаковыми диагональными элементами, равными 6,25 $м^2\backslash с^2$. На рисунке 2 изображена гистограмма распределения ошибок измерения компонент контактных ветра из данных. Вдоль осей X и Y отложены измерения соответствующих компонент, которые изменяются от -15 до +15 м\с, а вдоль оси Z отложена соответствующая вероятность.

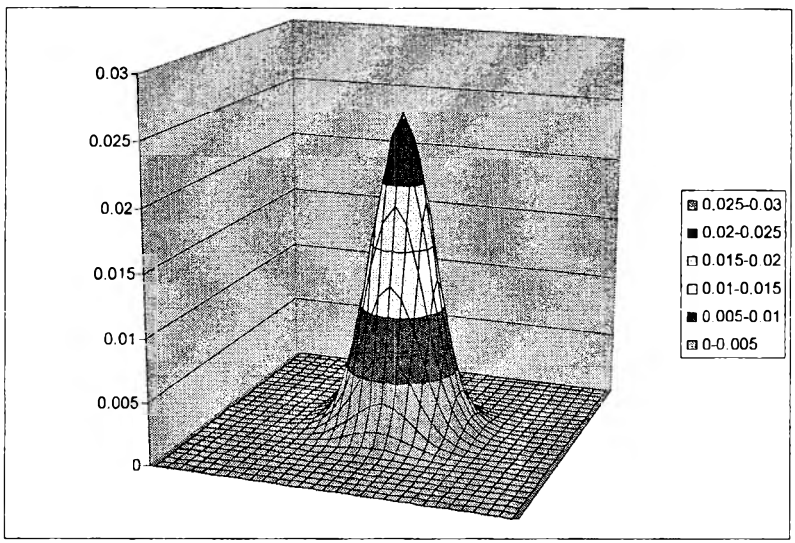

Рисунок 2. Гистограмма распределения ошибок измерения контактных данных.

Ранее контактные данные сами использовались как источник данных, используемых для прогнозирования погоды. Но на текущий момент по всему миру контактные измерения используются по большей части для верификации данных, полученных из других источников.

1.4. Статистический аппарат, используемый в работе

1.4.1. Характеристические функции

В теории вероятности характеристическая функция любой случайной величины полностью определяет закон распределения этой величины. На вещественной прямой характеристическую функцию можно выразить формулой:

$$\Phi_X(t) = \mathrm{E}(e^{itX}),$$

где t — вещественное число, а **E** обозначает математическое ожидание.

В многомерном случае для непрерывной случайной величины характеристическая функция задается следующим образом:

$$\mathrm{E}(e^{itX}) = \int_{\Omega} e^{i\langle t,x\rangle} dF_X(x),$$

где i — это квадратный корень из -1 (то есть мнимая единица), F_X — функция распределения случайной величины, а знак <,> обозначает скалярное произведение. Данное выражение можно переписать в терминах плотности вероятности $f_X(x)$ как:

$$\mathrm{E}(e^{itX}) = \int_{\Omega} e^{i\langle t,x\rangle} f_X(x)dx$$

Для дискретной случайной величины, заданной вероятностями $P(X = x_i) = p_i$, i = 1,..., n, характеристическая функция принимает вид:

$$\mathrm{E}(e^{itX}) = \sum_{i=1}^{n} e^{i<t,x_i>} p_i$$

Свойства характеристических функций:

1. Характеристическая функция однозначно определяет распределение. Пусть X,Y суть две случайные величины, и $\Phi_X(t) = \Phi_Y(t)$. Тогда закон распределения первой случайной величины совпадает с законом распределения второй. В частности, если обе величины абсолютно непрерывны, то совпадение характеристических функций влечёт совпадение плотностей. Если обе случайные величины дискретны, то совпадение характеристических функций влечёт совпадение функций вероятности.
2. Характеристическая функция всегда ограничена: $|\Phi_X(t)| < 1$.
3. Характеристическая функция в нуле равна единице: $\Phi_X(0) = 1$.
4. Характеристическая функция всегда непрерывна: $\Phi_X \in C(\mathbb{R})$

5. Характеристическая функция как функция случайной величины однородна:
$$\phi_{aX}(t) = \phi_X(at),\ \forall a \in \mathbb{R}$$
6. Характеристическая функция суммы независимых случайных величин равна произведению их характеристических функций. Пусть $X_1,\ldots,X_n$ есть независимые случайные величины. Обозначим $S_n = \sum_{i=1}^{n} X_i$. Тогда:
$$\phi_{S_n}(t) = \prod_{i=1}^{n} \phi_{X_i}(t)$$

Характеристическая функция $\varphi_X(t)$ тесно связана с непрерывным преобразованием Фурье, так как она совпадает с комплексным сопряжением результата применения непрерывного преобразования Фурье к функции плотности распределения соответствующей случайной величины:
$$\varphi_X(t) = \langle e^{itX} \rangle = \int_{-\infty}^{\infty} e^{itx} p(x)\,dx = \overline{\left(\int_{-\infty}^{\infty} e^{-itx} p(x)\,dx \right)} = \overline{P(t)},$$
где P(t) обозначает непрерывное преобразование Фурье функции плотности вероятности.

Подобным же образом функция плотности вероятности может быть получена их характеристической функции с помощью обратного преобразования Фурье:
$$p(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{itx} P(t)\,dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{itx} \overline{\varphi_X(t)}\,dt.$$

1.4.2. Проверка гипотез о распределении. Критерий согласия Пирсона (χ^2)

Критерий согласия Пирсона – это один из наиболее часто применяемых критериев для проверки гипотез о предполагаемом законе распределения.

Алгоритм критерия следующий:

- По выборке строится интервальный статистический ряд и гистограмма.
- По виду гистограммы выдвигаются гипотезы:
  H_0 – случайная величина X распределена по такому-то закону: $f(x) = f_0(x)$
  H_1 - случайная величина X не распределена по такому-то закону: $f(x) \neq f_0(x)$,
  где f(x) и $f_0(x)$ – функции плотности вероятности.
- Вычисляется значение критерия по формуле:

$$\chi^2 = n\sum_{j=1}^{M}\frac{(p_j - p_j^*)^2}{p_j} = \sum_{j=1}^{M}\frac{(v_j - np_j)^2}{np_j},$$

где p_j^* — эмпирическая вероятность попадания случайной величины в j-ый интервал, полученная по вборке, а p_j — аналогичная теоретическая вероятность при условии, что гипотеза H_0 верна:

$$p_j = P(A_j < X < B_j) = \int_{A_j}^{B_j} f_0(x)dx = F_0(B_j) - F_0(A_j),$$

где A_j и B_j – это границы j-го интервала, а F_0 — теоретический закон распределения.

Величина χ^2 распределена по закону, который называется распределением χ^2 (хи-квадрат). Так как аналитическое выражение плотности распределения χ^2 является довольно сложным, то на практике используется таблицу значений $\chi^2_{a,k}$, рассчитанных из уравнения $p(\chi^2 > \chi^2_{a,k}) = a$. В данном выражении индекс k является параметром распределения χ^2 и называется числом степеней свободы, а число a называется уровнем значимости и соответствует вероятности отвергнуть гипотезу H_0 притом, что она верна.

- Из таблицы распределения χ^2 выбирается значение $\chi^2_{a,k}$.
- Если вычисленное значение критерия больше, чем соответствующее табличное значение, то гипотеза H_0 отвергается, в противном случае нет оснований ее отклонить.

2. Постановка задачи

Имеется набор текстовых файлов, в каждом из которых в формате ASCII содержится массив данных следующего вида:

$$\mathbf{Y[j]=(a_{yj},b_{yj},t_{yj},w_{yj},v_{yj})}, \qquad (*)$$

где **a** и **b** — пространственные координаты (широта и долгота) точки измерения данных, **t** — время замера, а **w** и **v** — модуль и направление вектора измеренной скорости соответственно. Каждая из этих величин представляет собой либо целое число, либо вещественное с двумя цифрами после запятой.

Требуется проверить выполнимость предположения фильтра Калмана о нормальности распределения ошибок измерения для скаттерометрических показаний. В фильтре Калмана используются компоненты вектора ошибок, а не его модуль и направление. Поэтому задачу формально можно записать следующим образом:

По выборке данных скаттерометрических измерений **Y** вида (*) необходимо построить алгоритм для вычисления функции плотности распределения ошибок измерения величин

$r = w*\cos(v)$ и $s = w*\sin(v) - f_{r,s}(x,y)$.

3. Обзор существующих решений рассматриваемой задачи

Большинство работ в области усвоения данных посвящено вычислению функционала из формулы (1) раздела 1.1. Если известны $\mathbf{B}^{-1}$ и $\mathbf{R}^{-1}$, то данный функционал в таком виде минимизируется за одну итерацию методом Ньютона. Но основная проблема состоит в том, что количество параметров модели среды и количество данных, получаемых в результате измерений, достаточно велико, и размеры матриц **B** и **R** достигают нескольких тысяч по обоим измерениям. В дальнейшем при увеличении разрешения измерительных устройств, количество данных, а значит, и размер соответствующих матриц будет только возрастать. Причем этот рост будет опережать увеличение производительности вычислительной техники, поэтому эффективность расчетов по минимизации соответствующего функционала и в будущем останется актуальной проблемой.

Относительно же скаттерометров, как источников информации, основная масса исследований направлена на восстановление ветра по отраженному сигналу от волн на водной поверхности, а именно устранение неоднозначностей и повышение точности результатов. Для решения обратной задачи восстановления ветра используются различные методы: нейросетевой подход [2], байесовский подход [3]. В результате решения такой задачи могут возникать неоднозначности. Обзор методов борьбы с ними дается в работе [4].

Работ же, касающихся непосредственно тематики диплома — исследованию распределения ошибок показаний восстановленных ветров, почти нет. Одной из таких работ является [5]. Но и в ней лишь предлагается модель расчета скорости ветра, в которой используются случайные процессы, порождающие ошибки с нормальным распределением. Таким образом, сложилась ситуация, когда отсутствуют какие-либо исследования допустимости использования данных со спутников-скаттерометров для задачи усвоения данных с использованием фильтра Калмана.

4. Построение алгоритма нахождения эмпирического закона распределения ошибок спутниковых измерений

Для выделения ошибок измерения из имеющихся данных воспользуемся имеющейся базой контактных измерений. Контактные данные можно считать в том же формате, что и спутниковые:

$\mathbf{X[i]} = (\mathbf{a_{xi}}, \mathbf{b_{xi}}, \mathbf{t_{xi}}, \mathbf{w_{xi}}, \mathbf{v_{xi}})$,

где аналогично **a** и **b** – пространственные координаты (широта и долгота) точки измерения данных, **t** – время замера, а **w** и **v** – модуль и направление вектора измеренной скорости соответственно.

Итого имеем 2 массива **X[i]** и **Y[j]**. Необходимо найти пары измерений из двух массивов, которые достаточно близки по времени и пространственному расположению для того, чтобы их можно было считать замерами одних тех же показаний. Тогда для такой пары можно получить разницу ошибок измерений.

Для всех i и j проверяются условия:

1. $|a_{xi} - a_{yj}|^2 + |b_{xi} - b_{yj}|^2 < L^2$, в нашем случае **L** = 12,5 км – половина ширины покрытия территории лучем спутника за один замер.
2. $|t_{xi} - t_{yj}| < T$, где $T = L / 35 \approx 20$ мин. 35 км / ч – характерная для атмосферы скорость изменения состояния.

Все найденные пары подвергаются первоначальному контролю качества — проверке на выбросы: если векторная разность скоростей в паре по модулю превышает 15 м/с (величина, используемая для тех же целей в ГидроМетЦентре), то пара не учитывается.

Для всех найденных пар вычисляется векторная разность контактных и спутниковых показаний ветра, то есть строится вектор **g**, компоненты которого равны:

$$g_1 = W_Y * \cos V_Y - W_X * \cos V_X$$
$$g_2 = W_Y * \sin V_Y - W_X * \sin V_X$$

Полученный вектор является разностью компонент измеренных векторов, к котором истинное значение вектора ветра пропадает, и остаются только чистые ошибки измерения:

$Y_j - X_i = Z^t + r_j - Z^t + q_i = r_j - q_i$, где r_j – ошибки спутниковых измерений, а q_i – соответственно ошибки контактных, а Z^t – истинное состояние вектора ветра.

Но интерес представляет распределение именно спутниковых ошибок. Получить соответствующий закон распределения из распределения разностей позволяет аппарат характеристических функций. Используя взаимосвязь характеристической функции и преобразования Фурье, можно посчитать характеристическую функцию разности ошибок непосредственно по данным. Характеристическая функция ошибок контактных измерений может быть непосредственно подсчитана, так как закон распределения компонент вектора ошибки контактных данных известен.

Воспользовавшись свойствами характеристических функций можно записать следующее:

$$\varphi_{r-q}(t) = \varphi_r(t) * \varphi_q(-t).$$

Откуда следует, что можно получить характеристическую функцию ошибок спутниковых измерений:

$$\varphi_r(t) = \varphi_{r-q}(t) / \varphi_q(-t).$$

Непосредственное аналитическое выражение для плотности распределения разности ошибок неизвестно, а для анализа имеется лишь соответствующая эмпирическая плотность вероятности, которую можно рассчитать на сетке с произвольным шагом. Поэтому необходимо воспользоваться дискретным вариантом преобразования Фурье для вычисления характеристической функции разности ошибок. Преобразование производится для величин, заданных разбиением области определения эмпирической плотности вероятности.

Характеристическая функция ошибок контактных измерений может быть рассчитана в аналитическом виде. Для нее имеем:

$$\varphi_q(-t) = \exp[-1/2*<t,\sigma>],$$

где σ – вектор дисперсий по каждой из компонент. В данном случае обе его компоненты равны 6,25 (величина, полученная из ГидроМетеоЦентра). Расчет данной функции производится на координатной сетке с количеством узлов равным количеству узлов разбиения области определения эмпирической плотности вероятности разности спутниковых и контактных измерений. При этом компоненты параметра t изменяются в пределах от t_{min} до t_{max}, где $-t_{min} = t_{max} = \pi / step_{emp}$, где $step_{emp}$ — шаг, соответствующей каждой компоненте из разбиения области определения эмпирического распределения разности ошибок.

После деления двух вышеуказанных характеристических функций, заданных на одинаковом количестве узлов, для полученной эмпирической характеристической функции спутниковых ошибок с помощью обратного преобразования Фурье можно получить эмпирическую плотность вероятности этих ошибок.

5. Заключение

В данной дипломной работе получены следующие основные результаты:

1. Разработан алгоритм получения эмпирической аппроксимации закона распределения ошибок одного источника по известному закону распределения другого и эмпирическому распределению их суммы или разности
2. Построенный алгоритм был применен в рамках задачи выяснения справедливости базовых предположений процедуры усвоения данных для источника метеоданных - спутника – скаттерометра QuickScat
3. Выполнена программная реализация построенного алгоритма получения закона распределения ошибок
4. По результатам вычислительного эксперимента сделан вывод о требующейся корректировке процедуры усвоения для обработки данных с указанного спутника

5. В процессе исследований были выдвинут ряд гипотез об исследуемых ошибках, часть из которых была подтверждена, а часть помогла в выяснении связей между компонентами ошибок.

Литература

1. Stephen E. Cohn. An Introduction to Estimation Theory // DAO Ofiice Note 97-01.
2. Cornford. D., Nabney I. T., Bishop. Neural Network-Based Wind Vector Retrieval from Satellite Scatterometer Data // C. M. Neural Computing and Applications. 8. pp 206-217. Neural Computing Research Group. Aston University. BIRMINGHAM B4 7ET. UK.
3. Dan Cornford, Ian T. Nabney, David J. Evans. Bayesian Retrieval of Scatterometer Wind Fields // pp 22. Neural Computing Research Group. Aston University. BIRMINGHAM B4 7ET. submitted to Journal of Geophysical Research.
4. Stoffelen A., D. Anderson. Ambiguity Removal and Assimilation of Scatterometer Data Quart // J. Royal Meteor. Soc. 1997. 123. 491-518.
5. Freilich M. H., B. A. Vanhoff. QuikSTAT vector wind accuracy: Initial esimates. Proc. QuikSTAT // Cal/Val Early Science Meeting. Pasadena. CA. 1999. Jet Propulsion Laboratory.
6. Официальный сайт библиотеки FFTW [HTML](http://fftw.org/).

Позднеев А.В.

Параллельный код частиц в ячейке для моделирования процессов в масс-спектрометре[1]

1. Введение

Работа посвящена численному моделированию экспериментов по измерению масс, осуществляемых посредством масс-спектрометра ионного циклотронного резонанса с преобразованием Фурье (Fourier transform ion cyclotron resonance, FTICR) [1]. Целью является вычисление траекторий ионов, нахождение ICR-сигнала и его преобразование в масс-спектр. Масс-спектрометрия — физический метод установления отношения массы (m) ионов к их заряду (q); термин «масса» в масс-спектрометрии часто употребляется в значении отношения m/q. В основу работы FTICR масс-спектрометра положено явление ионного циклотронного резонанса; определение массы состоит в детектировании ICR-сигнала ионов, в постоянном магнитном поле **B** движущихся с циклотронной частотой [1]

$$\omega_c = \frac{qB}{m} \qquad (1)$$

В настоящее время методы масс-спектрометрии приобрели важное значение в протеомике [2, 3]. FTICR масс-спектрометры обеспечивают непревзойденную точность измерения масс и разрешающую способность [4], необходимые в задачах идентификации белков и определения состава биомолекул [2].

Ионная ловушка FTICR масс-спектрометра обычно имеет цилиндрическую или кубическую форму и линейные размеры около 2,5 см. Ионы анализируемого вещества удерживаются сильным (4,7–12,0 Тл [5]) магнитным полем, направленным по ее оси. К торцевым электродам прикладывается небольшое (порядка 1 В) напряжение для создания вдоль магнитного поля потенциальной ямы. Перпендикулярное магнитному радиочастотное электрическое поле $\mathbf{E}^{\mathrm{rf}}(t) = \mathbf{E}_0^{\mathrm{rf}} \cos\gamma t$ возбуждает только те ионы, циклотронная частота ω_c которых совпадает с частотой $\gamma = \gamma(t)$. В результате циклотронного возбуждения частицы начинают двигаться в фазе, образуя ионное облако. Массы определяют по сигналу от возбужденных ионов на детектирующих пластинах [1].

[1] Работа выполнена при поддержке гранта РФФИ № 05-07-90238.

В FTICR масс-спектрометрии проблема заключается в том, что измеренная частота отличается от циклотронной, вычисляемой по формуле (1). Вызываемые кулоновскими взаимодействиями несовпадения значительно затрудняют точное определение масс [1, 4, 6] Одним из способов исследования влияния подобных, скрытых для экспериментатора параметров, на масс-спектр является компьютерное моделирование.

Отраслевым стандартом в масс-спектрометрическом сообществе является программа SIMION [7]. Код позволяет отслеживать траектории нескольких десятков ионов, исследовать влияние столкновений с нейтральными частицами на масс-спектр, рассматривать вопросы оптимизации геометрии ловушки. Из других близких по функциональности пакетов можно выделить ITSIM [8] и ISIS [9]. Делались попытки использования многопроцессорных компьютеров для моделирования взаимодействующих в FTICR масс-спектрометре ионов по методу частица-частица [10]. В работе [11] было предложено применять метод «частиц в ячейке» (particle-in-cell, PIC) [12] к проблеме исследования кулоновских эффектов в ICR-ловушке.

Трудность моделирования FTICR эксперимента по измерению масс заключается в том, что для достижения в расчете параметров реального инструмента, необходимо интегрировать уравнения движения для сотен тысяч взаимодействующих частиц на протяжении миллионов шагов по времени. Значительно сократить время вычислений можно, используя возможности современных параллельных компьютеров и высокопроизводительных научных библиотек.

Целью данной работы является разработка параллельного трехмерного кода моделирования FTICR масс-спектрометра в реальной кубической геометрии для большого числа взаимодействующих частиц на основе метода «частиц в ячейке».

Далее в работе приводится постановка задачи, описание метода решения, результаты исследования зависимости ускорения разработанного кода от числа процессоров и анализ его масштабируемости.

2. Постановка задачи

Движение p-го иона с массой m_p, несущего заряд q_p, в магнитном поле $\mathbf{B}$ определяется действием силы Лоренца:

$$\begin{cases} m_p \dot{\mathbf{v}}_p = q_p \mathbf{E}(\mathbf{r}_p, t) + q_p [\mathbf{v}_p \times \mathbf{B}], \\ \dot{\mathbf{r}}_p = \mathbf{v}_p, \end{cases} \tag{2}$$

где $\mathbf{r}_p = \{x_p, y_p, z_p\}$ — радиус-вектор p-го иона, $\mathbf{v}_p$ — его скорость, и электрическое поле $\mathbf{E}(\mathbf{r},t)$ определяется потенциалом $\Phi^{\text{tr}}(\mathbf{r})$ удерживающего поля ловушки, потенциалом $\Phi^{\text{rf}}(\mathbf{r},t)$ радиочастотного возбуждающего поля и потенциалом $\Phi^{\text{cl}}(\mathbf{r},t)$ кулоновского поля:

$$\mathbf{E}(\mathbf{r},t) = -\nabla\Phi(\mathbf{r},t) = -\nabla\left(\Phi^{\text{tr}}(\mathbf{r}) + \Phi^{\text{cl}}(\mathbf{r},t) + \Phi^{\text{cl}}(\mathbf{r},t)\right).$$

Рассмотрим ловушку с квадратным сечением, торцевые электроды которой находятся на расстоянии a друг от друга, расстояние между парами возбуждающих и детектирующих электродов — d. Удерживающее поле возникает за счет приложения постоянного потенциала V^{tr} к торцевым электродам, возбуждающее поле определяется потенциалами $\pm V^{\text{rf}} \cos\gamma t$, приложенными к двум противоположным продольным пластинам. Кулоновское поле создается ионами анализируемого вещества и зарядами, которые они индуцируют на стенках ловушки:

$$\Phi^{\text{cl}}(\mathbf{r},t) = \frac{1}{4\pi\varepsilon_0}\sum_{p=1}^{N_p}\frac{q_p}{|\mathbf{r}-\mathbf{r}_p(t)|} + \Phi^{\text{wall}}(\mathbf{r},t).$$

Здесь N_p — общее число частиц, $\Phi^{\text{wall}}(\mathbf{r},t)$ — потенциал поля индуцированных зарядов, $\varepsilon_0 = 8{,}85\cdot10^{-12}$ Ф/м — электрическая постоянная. Если ввести пространственную плотность $\rho(\mathbf{r},t)$ заряда и учесть, что потенциалы $\Phi^{\text{tr}}(\mathbf{r})$ и $\Phi^{\text{rf}}(\mathbf{r},t)$ удовлетворяют уравнению Лапласа, то потенциал $\Phi(\mathbf{r},t)$ поля внутри ловушки можно искать как решение первой краевой задачи для уравнения Пуассона с неоднородными граничными условиями:

$$\begin{cases} \Delta\Phi = -\rho/\varepsilon_0, \\ \Phi|_{x=\pm d/2} = \pm V^{\text{rf}}\cos\gamma t, \\ \Phi|_{y=\pm d/2} = 0, \\ \Phi|_{z=\pm a/2} = V^{\text{tr}}. \end{cases} \qquad (3)$$

Математическая постановка задачи сводится к интегрированию уравнений движения ионов и вычислению заряда, индуцированного на детектирующих пластинах.

3. Метод решения задачи.

3.1. Общая схема алгоритма

Чтобы проинтегрировать уравнения движения (2), необходимо знать силу, действующую на частицу со стороны электрического поля. Значительную трудность представляет вычисление кулоновской составляющей. При небольшом числе частиц возможно определить их взаимодействие, воспользовавшись законом Кулона, но остается проблема учета влияния индуцированных зарядов. Вычислительные затраты схемы частица-частица растут квадратично с увеличением числа ионов, однако особый интерес представляют именно расчеты для большого числа частиц, когда существенными становятся кулоновские эффекты.

Для решения поставленной задачи был выбран метод «частиц в ячейке» [12]. Уравнения движения решаются в области, а для расчета полей вводится сетка. Потенциалы и силы вычисляются посредством интерполяции по массиву сеточных значений. Сеточная плотность заряда находится взвешиванием частиц на сетке. Итерация алгоритма выглядит следующим образом:

1. Интерполяция заряда на сетку.
2. Нахождение потенциала (решение задачи (3) для уравнения Пуассона).
3. Вычисление напряженности поля в узлах сетки.
4. Интерполяция поля на частицы.
5. Интегрирование уравнений движения.

Шаг τ по времени определяется максимальной циклотронной частотой $\omega_{c,\max}$ ионов анализируемой смеси. Общее число итераций выбирается исходя из разрешения частоты, которое ожидается получить. Разрешение (resolution) $\Delta\omega_{50\%}$ в частотной области и $\Delta m_{50\%}$ на шкале масс определяется как ширина спектрального пика. Для гармонического сигнала принимают $\Delta\omega_{50\%} = 2\pi / T_{\text{acq'n}}$, где $T_{\text{acq'n}}$ — время детектирования. Необходимым условием разрешимости двух пиков, которым отвечает разность частот $\Delta\omega$, является условие $\Delta\omega_{50\%} < \Delta\omega$. Время детектирования определяется неравенством $T_{\text{acq'n}} > 2\pi / \Delta\omega$. Если выбирать $\tau = 2\pi / n\omega_{c,\max}$, где произведение $n\tau$ дает значение минимального циклотронного периода ионов смеси, то общее число итераций N, которые необходимо выполнить для разрешения масс m и $m + \Delta m$,

$$N > n\frac{\omega}{\Delta\omega} \approx n\frac{m}{\Delta m} \qquad (4)$$

Разрешающая способность (resolving power) $m/\Delta m_{50\%}$ (или $\omega/\Delta\omega_{50\%}$) современных коммерческих устройств лежит в диапазоне 80 000–1 130 000 [5]. Неравенство (4) показывает, что необходимо выполнить миллионы итераций, чтобы приблизиться к условиям реального эксперимента.

3.2. Схема интегрирования уравнений движения

Движение ионов в ловушке FTICR масс-спектрометра определяется действием силы Лоренца. Чтобы точно воспроизводить циклотронное вращение, в схеме интегрирования нужно, во-первых, разделить члены, отвечающие циклотронному вращению, и члены, определяющие движение в электрическом поле, и, во-вторых, использовать коррекцию численной частоты. Необходимость коррекции частоты следует из конечности во времени шага интегрирования. Так как электрическая сила, действующая на частицу, на каждом шаге находится из решения уравнения Пуассона, невозможно использовать стандартные подпрограммы, которые принимают для вычисления силы в качестве аргумента аналитическую формулу.

Для интегрирования уравнений движения (2) была выбрана хорошо зарекомендовавшая себя во многих приложениях центрированная по времени конечно-разностная аппроксимация, которую предложил Бунеман [12]:

$$\begin{cases} m\dfrac{\mathbf{v}^{n+0.5}-\mathbf{v}^{n-0.5}}{\alpha\tau} = q\mathbf{E}^n + q\left[\dfrac{\mathbf{v}^{n+0.5}+\mathbf{v}^{n-0.5}}{2}\times\mathbf{B}^n\right], \\ \dfrac{\mathbf{r}^{n+1}-\mathbf{r}^n}{\tau} = \mathbf{v}^{n+0.5}. \end{cases} \qquad (5)$$

Непрерывные функции $\mathbf{v}$ и $\mathbf{r}$ заменяются их значениями в дискретные моменты времени; координаты частиц и поля вычисляются на «целых» временных слоях ($t = 0,\ \tau,\ 2\tau,\ 3\tau,\ \ldots$), скорости — на «полуцелых» ($t = \tau/2,\ 3\tau/2,\ 5\tau/2,\ \ldots$). Введение поправочного множителя

$$\alpha = \frac{2}{\omega_c\tau}\operatorname{tg}\frac{\omega_c\tau}{2}$$

позволяет точно воспроизводить при интегрировании циклотронную частоту [12]. Для разделения в первом уравнении (5) членов, связанных с движением в магнитном поле $\mathbf{B}^n = \{0,0,B\}$, и членов, описывающих

ускорение в электрическом был использан метод, предложенный Борисом [12]: из уравнений полностью исчезает $\mathbf{E}^n$, и получаемое соотношение будет представлять из себя чистое вращение.

3.3. Интерполяция заряда и поля

Сеточная плотность заряда в узле (i_1, i_2, i_3), создаваемая зарядами q_p, расположенными в точках (x_p, y_p, z_p), находится по формуле

$$\rho_{i_1 i_2 i_3} = \frac{1}{h_1 h_2 h_3} \sum_{p=1}^{N_p} q_p W_{i_1 i_2 i_3}(x_p, y_p, z_p).$$

В качестве $W_{i_1 i_2 i_3}(x_1, x_2, x_3)$ могут быть использованы различные функции; в данной работе применяется интерполяция вида

$$W_{i_1 i_2 i_3}(x_1, x_2, x_3) = W_{i_1}(x_1) W_{i_2}(x_2) W_{i_3}(x_3) \qquad (6)$$

где

$$W_{i_\alpha}(x_\alpha) = \begin{cases} 1 - |x_\alpha - x_{i_\alpha}|/h_\alpha, & |x_\alpha - x_{i_\alpha}| < h_\alpha, \\ 0, & |x_\alpha - x_{i_\alpha}| \geq h_\alpha, \end{cases} \quad \alpha = 1,2,3.$$

Для интерполяции поля с узлов сетки на частицы используются те же весовые функции (6) с точностью до множителя $h_1 h_2 h_3$:

$$\mathbf{E}(x_p, y_p, z_p) = \sum_{i_1=0}^{N_1} \sum_{i_2=0}^{N_2} \sum_{i_3=0}^{N_3} W_{i_1 i_2 i_3}(x_p, y_p, z_p) \mathbf{E}_{i_1 i_2 i_3}.$$

3.4. Численное решение уравнения Пуассона

Для решения поставленной задачи на каждом временном шаге PIC-алгоритма необходимо решать задачу (3) для уравнения Пуассона. Так как рассматриваемая область является параллелепипедом, то для численного решения дифференциальной задачи (3) был выбран прямой метод, основанный на комбинации серии двумерных быстрых преобразований Фурье (БПФ) и метода прогонки по третьему измерению. При этом обобщается на трехмерный случай алгоритм, изложенный в работе [13].

Двумерные преобразования Фурье являются независимыми и могут быть выполнены параллельно. Для этого используется функция ***fftw_plan_many_r2r*** (с типом преобразования ***FFTW_RODFT00*** по обоим направлениям) параллельной библиотеки FFTW [14]. Так как в ходе численного эксперимента приходится многократно решать уравнение Пуассона с разными правыми частями, а коэффициенты α метода прогонки не зависят от правой части уравнения, то они могут быть вычислены заранее [13]. Обращение к массивам Фурье- и

прогоночных коэффициентов осуществляется исходя из соображений оптимального использования кэш-памяти.

4. Анализ ускорения и масштабируемости параллельного кода

Метод был реализован программно на языке Fortran 90. Код распараллелен с использованием директив OpenMP. Расчеты проводились на 16-процессорной SMP-системе IBM pSeries 690 Regatta (факультет ВМК МГУ).

При анализе производительности код получал эксклюзивный доступ ко всем 16 процессорам системы. Выполнялось 100 итераций, и на каждой из них фиксировалась продолжительность шагов PIC-алгоритма. Замеренные времена сортировались и из отсортированного массива исключалось 12 наименьших и 13 наибольших значений (25%), по оставшимся элементам вычислялось среднее.

4.1. Анализ ускорения

Анализ зависимости ускорения кода от числа процессоров осуществлялся на основании моделирования движения 1 600 000 частиц на сетках 32^3, 64^3 и без учета кулоновского взаимодействия с использованием сетки 32^3. Расчеты проводились для $p = 1, 2, ..., 16$, процессоров.

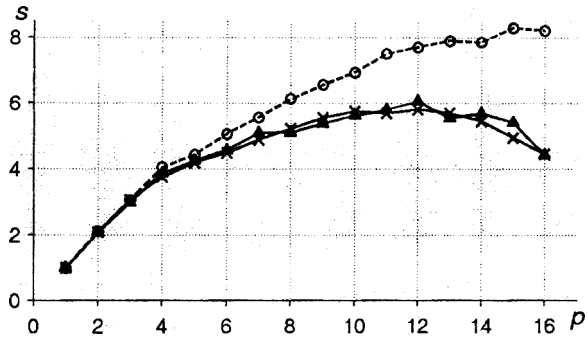

Рисунок 1. Ускорение итерации PIC-алгоритма для сетки 32^3 (маркер ×), 64^3 (маркер ▲) и без учета кулоновского взаимодействия на сетке 32^3 (маркер ○).

Общий вывод, который можно сделать из рисунка 1, заключается в том, что при моделировании взаимодействующих частиц имеется максимум ускорения, так что использование в расчетах более 12 процессоров становится нецелесообразным. Рассмотрим, что может являться причиной падения ускорения.

На рисунках 2 и 3 приведены зависимости ускорения от числа процессоров для процедур вычисления весов интерполяции, интерполяции поля и интегрирования уравнений движения при расчетах на сетках 32^3 и 64^3. Как и следовало ожидать, при распараллеливании по частицам мы получаем гарантированное ускорение с увеличением числа процессоров. При этом имеется незначительная зависимость от размера сетки: процедура интерполяции поля на сетке 32^3 работает несколько быстрее с увеличением числа процессоров, чем на сетке 64^3. При интерполяции заряда на сетку (см. рисунок 4) эта разница проявляется более заметно. В качестве объяснения логично предположить, что при использовании сеток с меньшим числом узлом уменьшается число промахов кэша.

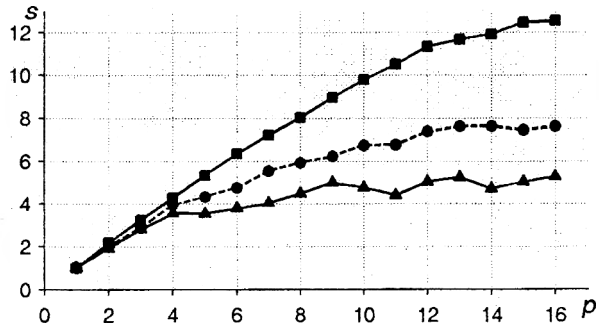

Рисунок 2. Ускорение процедур вычисления весов интерполяции (маркер •), интерполяции поля (маркер ■), интегрирования уравнений движения (маркер ▲) при расчетах на сетке 32^3.

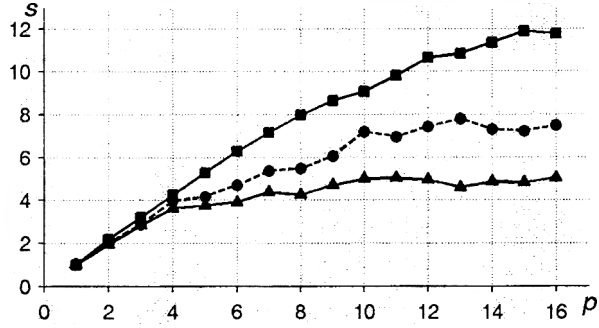

Рисунок 3. Ускорение процедур вычисления весов интерполяции (маркер •), интерполяции поля (маркер ■), интегрирования уравнений движения (маркер ▲) при расчетах на сетке 64^3.

Несмотря на то, что теоретически процедура решения уравнения Пуассона, основанная на комбинации серии независимых двумерных БПФ и метода прогонки, хорошо должна поддаваться распараллеливанию, на практике, с использованием текущей версии библиотеки FFTW (3.2a3) не удается получить более чем четырехкратного ускорения процедуры (см. рисунок 5). Проблема заключается не только в планировщике FFTW, который на использованной для тестирования машине оказывается неспособным определить оптимальное число параллельных нитей для своего выполнения. При числе процессоров, большем четырех, производительность процедуры прогонки, являющейся составной частью подпрограммы решения уравнения Пуассона, также катастрофически снижается (рисунок 6). Таким образом, это приводит к тому, что с увеличением числа процессоров доля времени, затрачиваемого на решение уравнения Пуассона (рисунок 7), и на вычисление поля объемного заряда вообще, постоянно возрастает, что, в конечном счете, и является причиной обсуждаемого падения ускорения (рисунок 1).

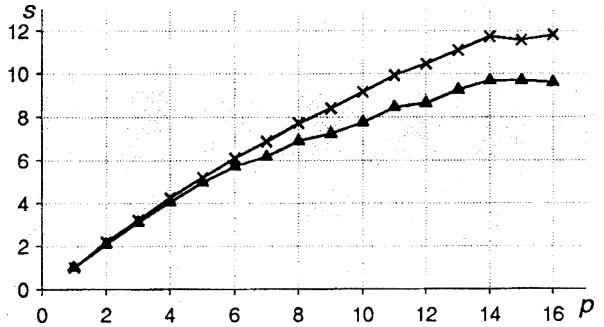

Рисунок 4. Ускорение процедур интерполяции заряда при расчете на сетках 32^3 (маркер ×), 64^3 (маркер ▲).

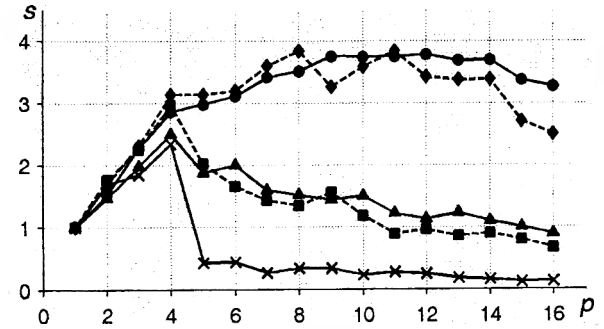

Рисунок 5. Ускорение процедуры решения уравнения Пуассона при расчете на сетках 323 (маркер ×), 483 (маркер ■), 643 (маркер ▲), 963 (маркер ♦) 1283 (маркер ●).

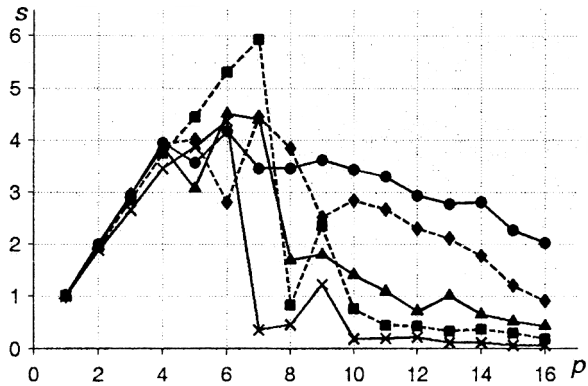

Рисунок 6. Ускорение процедуры прогонки в решении уравнения Пуассона при расчете на сетках 32^3 (маркер ×), 48^3 (маркер ■), 64^3 (маркер ▲), 96^3 (маркер ♦) 128^3 (маркер ●).

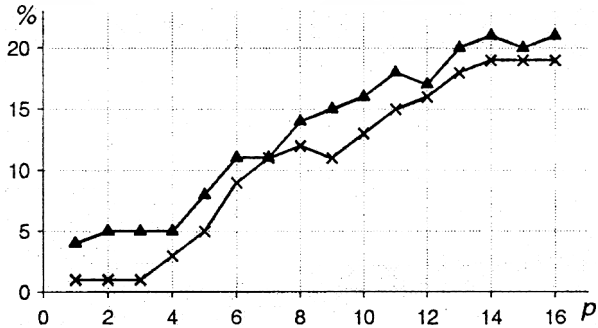

Рисунок 7. Доля времени, затрачиваемого на решение уравнения Пуассона от общей продолжительности итерации, при расчете на сетках 32^3 (маркер ×), 64^3 (маркер ▲).

4.2. Анализ масштабируемости

Под *масштабируемостью* будем понимать зависимость времени работы программы при увеличении размера задачи и пропорциональном увеличении числа задействованных для ее решения процессоров. В рамках данной работы в качестве характеристики размера задачи выберем число частиц, находящихся в ловушке масс-спектрометра.

За исходное время была взята продолжительность расчетов для 100 000 частиц с использованием одного процессора на сетках 32^3, 64^3

и, чтобы отследить влияние процедуры решения уравнения Пуассона, расчета на сетке 32^3 без учета кулоновского взаимодействия. Далее проводились расчеты для $100\,000 \cdot p$ частиц на $p = 2, 3, \ldots, 16$, процессорах. На рисунке 8 приведены средние времена, нормированные на продолжительность итерации при расчете на одном процессоре.

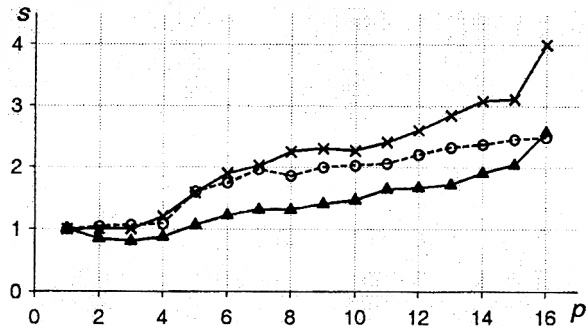

Рисунок 8. Нормированное время расчета для $100\,000 \cdot p$ частиц на сетках 32^3 (маркер ×), 64^3 (маркер ▲) и без учета кулоновского взаимодействия на сетке 32^3 (маркер ○).

График расчета без кулоновского взаимодействия показывает, что, когда число процессоров не превосходит четырех, код демонстрирует практически идеальную масштабируемость: увеличение числа частиц при пропорциональном увеличении числа процессоров не вызывает изменения времени выполнения. Но при числе процессоров, большем четырех, масштабируемость кода начинает ухудшаться. Если в случае расчетов с использованием кулоновского взаимодействия это может быть вызвано процедурами вычисления поля объемного заряда, то при моделировании невзаимодействующих частиц такое поведение можно списать лишь на затраты по синхронизации и/или насыщение шины памяти. Более детальный анализ показывает, что этот результат в обоих случаях связан с процедурой интегрирования уравнений движения: на пяти процессорах для 500 000 частиц эта подпрограмма работает в 2 раза медленнее, чем для 100 000 частиц на одном процессоре, в то время как на четырех процессорах эти времена сравнимы.

Необходимо отметить при использовании до четырех процессоров на сетке 64^3 мы получаем при большем числе частиц меньшую длительность итерации. Причина заключается в том, что при небольшом числе частиц значительное время тратится на решение уравнения Пуассона, которое неплохо распараллеливается при таком числе процессоров. Далее же, число частиц на процессор остается тем

же самым (100 000), а характеристики процедуры решения уравнения Пуассона ухудшаются.

5. Заключение

В работе представлен трехмерный код для моделирования процессов в масс-спектрометре. Предложено использовать алгоритм «частиц в ячейке» для учета влияния поля объемного заряда на траектории ионов.

Проведен анализ параллельной реализации метода. Показано, что при использовании более четырех процессоров сдерживающим ускорение фактором является процедура решения уравнения Пуассона. Возможное решение проблемы заключается в использовании не основанного на БПФ алгоритма. При числе процессоров, не превосходящем четырех, продемонстрирована практически идеальная масштабируемость в смысле неизменности времени выполнения при одновременном увеличении числа частиц и пропорциональном изменении числа процессоров. На основе проведенного анализа сделан вывод, что эффективность распараллеливания процедур интерполяции и интегрирования уравнений движения ограничена пропускной способностью шины памяти. Необходимо реорганизовать алгоритм таким образом, чтобы максимально загрузить процессор вычислениями на то время, пока из памяти поступают свежие данные.

С использованием разработанного кода уже был получен ряд важных с экспериментальной точки зрения результатов [15], и дальнейшее улучшение временных характеристик программы позволит осуществить более сложные расчеты и глубже проникнуть в суть процессов, происходящих в масс-спектрометре.

Литература

1. Fourier transform ion cyclotron resonance mass spectrometry: a primer. A.G. Marshall, C.L. Hendrickson and G.S. Jackson. 1998, Mass Spectrom. Rev., 17, pp. 1–35.
2. Mass spectrometry-based proteomics. R. Aebersold and M. Mann. 2003, Nature, 422, pp. 198–207.
3. Proteomics by FTICR mass spectrometry: top down and bottom up. B. Bogdanov and R.D. Smith. 2005, Mass Spectrom. Rev., 24, pp. 168–200.
4. Accurate mass measurements by Fourier transform mass spectrometry. L.-K. Zhang, R. Rempel, B.N. Pramanik and M.L. Gross. 2005, Mass Spectrom. Rev., 24, 286–309.
5. FTMS: Overcoming challenges. K. Cottingham. 2006, Anal. Chem., 78, pp. 655–657.

6. Experimental evidence for space-charge effects between ions of the same mass-to-charge in Fourier-transform ion cyclotron resonance mass spectrometry. R.L. Wong and I.J. Amster. 2007, Int. J. Mass Spectrom., 265, pp. 99–105.
7. SIMION for the personal computer in reflection. D.A. Dahl. 2000, Int. J. Mass Spectrom., 200, pp. 3–25.
8. Ion trajectory simulation for electrode configurations with arbitrary geometries. G. Wu, R.G. Cooks, Z. Ouyang, M. Yu, W.J. Chappell and W.R. Plass. 2006, J. Am. Soc. Mass Spectrom., 17, pp. 1216–1228.
9. Simulation of ion trajectories in a quadrupole ion trap: a comparison of three simulation programs. M.W. Forbes, M. Sharifi, T. Croley, Z. Lausevic and R.E. March. 1999, J. Mass Spectrom., 34, pp. 1219–1239.
10. Application of a parallel computer to simulation of an ion trajectories in an ion cyclotron resonance spectrometer. N.V. Miluchihin, K. Miura and M. Inoue. 1993, Rapid Commun. Mass Spectrom., 7, pp. 966–970.
11. Realistic simulation of the ion cyclotron resonance mass spectrometer using a distributed three-dimensional particle-in-cell code. D.W. Mitchell. 1999, J. Am. Soc. Mass Spectrom., 10, pp. 136–152.
12. Численное моделирование методом частиц. Р. Хокни, Дж. Иствуд, 1987, М.: Мир.
13. Методы решения сеточных уравнений. А.А. Самарский, Е.С. Николаев. 1978, М.: Наука.
14. The design and implementation of FFTW3. M. Frigo and S.J. Johnson. 2005, Proc. IEEE, 93, pp. 216–231.
15. Realistic modeling of ion cloud motion in a Fourier transform ion cyclotron resonance cell by use of a particle-in-cell approach. E.N. Nikolaev, R.M.A. Heeren, A.M. Popov, A.V. Pozdneev, K.S. Chingin. 2007, Rapid Commun. Mass Spectrom., 21, pp 3527–3546.

Семенов А. С., Эйсымонт Л. К.

Параллельное умножение матриц на суперкомпьютере с мультитредово-потоковой архитектурой

Введение

Прикладные задачи можно условно разделить на два класса: задачи CF-класса (cache-friendly) с высокой пространственно-временной локализацией обращений к памяти, что позволяет эффективно использовать кэш-памяти данных, и задачи DIS-класса (data intensive systems) с плохой локализацией. Характерный уровень реальной производительности на задачах CF-класса – более 70% от пиковой производительности. Известные примеры задач CF-класса – HP Linpack (рейтинговый тест формирования списка top500), умножение плотнозаполненных матриц. При решении задач DIS-класса кэш-память не позволяет снизить потери, возникающие из-за больших задержек выполнения операций обращения к памяти. По этой причине реальная производительность для этого класса задач обычно низкая, менее 5-10% от пиковой. В некоторых важных прикладных областях доля задач DIS-класса уже сейчас достигает 30-50%, причем их важность увеличивается год от года.

В ОАО «НИЦЭВТ» ведется исследовательский проект создания суперкомпьютера с мультитредово-потоковой архитектурой и аппаратной поддержкой распределенной общей памяти (МТП-суперкомпьютера), предназначенного для эффективного решения задач DIS-класса. Важно, чтобы создаваемый МТП-суперкомпьютер не оказался таким, что при эффективном решении DIS-задач эффективность решения задач CF-класса на нем оказалась бы ощутимо хуже, чем на суперкомпьютерах с традиционной архитектурой. В данной статье рассматриваются возможности эффективного решения на МТП-суперкомпьютере одной из задач CF-класса – умножения матриц.

1. Архитектура МТП-суперкомпьютера

МТП-суперкомпьютер содержит множество вычислительных узлов, соединенных коммуникационной сетью с огромной суммарной пропускной способностью при передаче коротких пакетов. В качестве варианта реализации такой сети в описываемых в статье исследованиях использовали сеть 4D тор с адаптивной бездедлоковой передачей пакетов.

Вычислительный узел МТП-суперкомпьютера содержит: сетевой адаптер/маршрутизатор; оригинальный многоядерный

мультитредово-потоковый микропроцессор (варианты J7 и J10) с аппаратной поддержкой трансляции адресов глобально адресуемой памяти и передачи коротких пакетов, реализующих такие обращения; локальную память с большим расслоением на базе стандартных DRAM-модулей.

В процессе исследований рассматривались базовые конфигурации микропроцессоров J7 и J10. Параметры этих вариантов представлены в таблице 1.

Табл. 1. Основные характеристики вариантов многоядерных мультитредово-потоковых микропроцессоров МТП-суперкомпьютера.

	J7-1	J7-2	J7-3	J10-1	J10-2	J10-3	J10-4
Freq	0.5	0.5	0.5	1	1	2	2
Количество ядер/тредов	2/64	2/64	4/64	8/64	8/64	8/128	8/128
Количество FXU/FPU	1/1	2/2	4/4	2/2	4/4	2/2	4/4
ILP	2	4	8	4	8	4	8
Peak, ГФлопс	2	4	16	32	64	64	128
Размер кэша, МБ/way	1/4	1/4	1/4	2/8	2/8	4/16	4/16
Кэш-память BW, ГБ/сек	64	64	64	128	128	256	256
DRAM BW, ГБ/сек	25.6	25.6	25.6	51.2	51.2	102.4	102.4
BWnet,ГБ/сек	12	12	12	25	25	64	64

Обозначения: Freq – частота процессора (ГГц); ILP - количество команд, выдаваемых в ядре на выполнение за такт; Peak - пиковая производительность процессора (ГФлопс); BWnet – пропускная способность сетевого интерфейса.

Для понимания излагаемого далее материала важно будет знать особенности отображения (распределения) адресов глобально адресуемой виртуальной памяти на локальную физическую память каждого из узлов.

Во-первых, возможно использование циклического отображения с выбором узла по младшим разрядам адреса со скремблированием, при котором каждые следующие восемь слов последовательности подряд идущих виртуальных адресов отображаются на физическую память выбираемого псевдослучайным образом узла.

Во-вторых, среди нескольких доступных вариантов отображения виртуальных адресов на физические существует блочное распределение, когда номер узла задается старшими битами

виртуального адреса. При этом блочно распределенная последовательность виртуальных адресов (сегмент) распределяется равными порциями (блоками) по узлам: в физической памяти первого узла первый блок подряд идущих виртуальных адресов, в физической памяти второго узла – второй блок, на последнем узле – последний блок.

Для оценки производительности МТП-суперкомпьютера на тестовых оценочных программах в ОАО «НИЦЭВТ» на языке Charm++ разработана параллельная потактовая идеализированная модель, которая работает и хорошо масштабируется по производительности до 256 узлов на системах МВС-15000 и МВС-50000, имеющихся в МСЦ РАН. Временные диаграммы работы сети и команд работы с памятью в модели максимально приближены к реальным.

2. Умножение матриц на одном процессоре

Рассмотрим различные варианты реализации теста умножения матриц. На рисунке 1 умножение матриц записано непосредственно по определению.

```
for (i = 0; i < N; ++i)
        for (j = 0; j < N; ++j)
                c[i,j] = 0
                for (k = 0; k < N; ++k)
                        c[i,j] += a[i,k] * b[k,j]
```

Рисунок 1. Умножение матриц C = A * B, записанное непосредственно по определению.

На рисунке 2 приведена программа, получающаяся посредством двукратной раскрутки двух внешних циклов программы на рисунке 1. При этом на каждой итерации внутри двух внешних циклов получается блок 2x2 результирующей матрицы C. Иллюстрацию работы с матрицами в этой программе можно увидеть на рисунке 3 (r = s = 2).

Отличие приведенных программ по производительности можно оценить посредством подсчета количества операций с плавающей точкой, приходящихся на одну команду загрузки из памяти. В первой программе на каждые два элемента $a_{i,k}$ и $b_{k,j}$, загруженных из памяти, приходятся две операции с плавающей точкой – умножение и сложение. При этом считаем, что $c_{i,j}$ будет находиться на регистре, поэтому обращения к памяти на каждом такте не потребует. Тогда получаем соотношение арифметических операций и операций с памятью 2/2 = 1. Во второй программе во внутреннем цикле вычисляется сразу блок 2*2 матрицы C. На каждой итерации самого внутреннего цикла загружаются $a_{i,k}$, $a_{i+1,k}$, $b_{k,j}$, $b_{k,j+1}$, но для них выполняется уже 8 операций сложения и умножения. Таким образом, получаем соотношение

арифметических операций и операций обращения к памяти 8/4 = 2. Переиспользование данных из памяти повысилось в два раза, поэтому производительность этой программы должна быть выше.

```
for (i = 0; i < N; i=i+2)
    for (j = 0; j < N; j=j+2) {
        d11 = d12 = d21 = d22 = 0;
        for (k = 0; k < N; ++k) {
            d11 += ai,k * bk,j
            d12 += ai,k * bk,j+1
            d21 += ai+1,k * bk,j
            d22 += ai+1,k * bk,j+1
        }
        ci,j    = d11
        ci,j+1  = d12
        ci+1,j  = d21
        ci+1,j+1 = d22
    }
```

Рисунок 2. Умножение матриц C = A * B, полученное при помощи двукратной раскрутки двух внешних циклов.

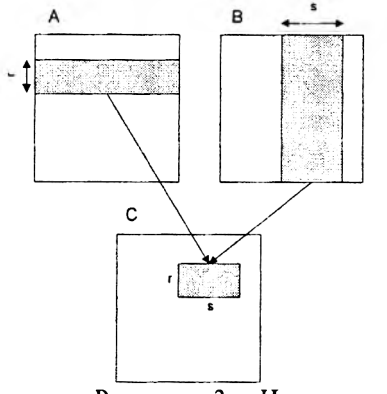

Рисунок 3. Используемый способ выделения тредового вычисления при умножении матриц.

Далее рассмотрим, как с использованием продемонстрированного метода развертки циклов можно получить мультитредовую программу для многоядерно-потокового микропроцессора вычислительного узла МТП-суперкомпьютера. Степень развертки циклов определяется количеством регистров, доступных в архитектуре. Для МТП-суперкомпьютера была произведена четырехкратная развертка первого и второго циклов. Внутренний цикл — это вычисление, выделяемое для одного треда при

доступных в архитектуре. Для МТП-суперкомпьютера была произведена четырехкратная развертка первого и второго циклов. Внутренний цикл — это вычисление, выделяемое для одного треда при распараллеливании цикла на множество тредов. При этом получается, что для одного треда требуется 4*4 + 4 + 4 = 24 регистра с плавающей точкой.

Для заданных матриц A, B и C, которые могут быть достаточно большими, показанный метод распараллеливания может потребовать выполнения большого количества тредов (тредовых вычислений), которое значительно превосходит количество тредовых устройств микропроцессора. При составлении мультитредовой программы получающееся множество тредовых вычислений статически равномерно делилось между тредовыми устройствами всех ядер. Таким образом, каждое тредовое устройство заканчивая одно тредовое вычисление, переходит к другому из выделяемого ему подмножества.

Таблица 2. Сравнение различных процессоров на тесте DGEMM.

	Cell+	Cell	X1E	Opteron	Itanium2	J7-2	J10-4
Freq		3.2	1.13	2.2	1.4	0.5	2
P	51.1	14.6	16.9	4.0	5.4	3.79	90.62
Peak	51.2	14.63	18	4.4	5.6	4	128
Эфф.,%	99.8%	99.7%	93.8%	90.9%	96.4%	94.7%	70.7%

Обозначения: Freq — частота процессора (ГГц); P – уровень реальной производительности теста (ГФлопс); Peak — пиковая производительность процессора (ГФлопс); Эфф. = P/Peak * 100%.

Тест DGEMM основан на умножении матриц, на нем мы будем сравнивать производительность процессоров (см. таблицу 2), данные по процессорам взяты из [**Ошибка! Источник ссылки не найден.**]. Конфигурация J7-2 (см. таблицу 1) сопоставима с Opteron и Itanium2, хотя частота J7-2 (500 МГц) значительно ниже в сравнении с этими процессорами. Конфигурация J10-4 значительно превосходит остальные процессоры. Отметим высокую эффективность современных коммерчески доступных процессоров фирм Intel, AMD и, IBM. Такая высокая эффективность получается на чрезвычайно сложных глубоко оптимизированных программах из библиотек от производителей процессоров. На процессорах J7/J10 высокий результат получается при помощи нехитрых преобразований, которые в состоянии выполнить любой оптимизирующий компилятор с поддержкой автоматического распараллеливания циклов на множество тредов. Наличие в ядрах микропроцессоров J7/J10 большого количества одновременно работающих тредовых устройств (J7-2 – 128, J10-4 – 1024) позволяет добиться, во-первых, толерантности, т. е. нечувствительности к

позволяет использовать высокий параллелизм выполнения команд от разных тредов (показатель ILP,

Табл. 1), который в целом по всем ядрам в J7-2 – 8 команд за такт, а в J10-4 – 64 команды за такт. Это и позволяет получить высокую реальную производительность в J7/J10.

3. Многопроцессорный вариант теста умножения матриц

Механизм распределенной общей памяти, реализованный аппаратно в МТП-суперкомпьютере, позволяет с небольшими изменениями запустить мультитредовую программу, разработанную для одного процессора на множестве процессоров. Изменения касаются разделения работы между процессорами узлов. Столбцы матриц B и C, а также строки матрицы A и соответствующие тредовые вычисления делятся теперь не только между ядрами и тредовыми устройствами, но и между узлами.

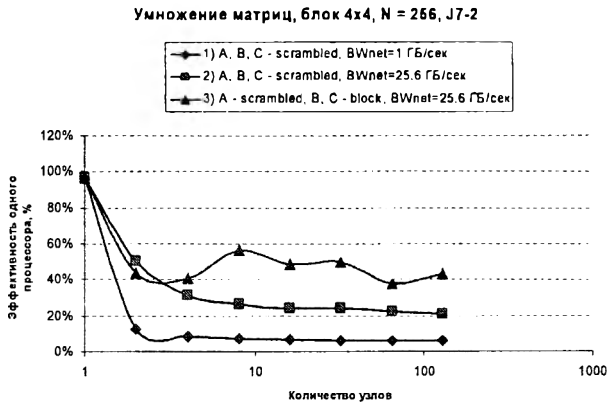

Рисунок 4. Эффективность одного узла на тесте умножения матриц в зависимости от количества узлов.

Во всех вариантах организации программ, рассмотренных далее, матрицы A, B и C находятся каждая в своем сегменте, что позволяет независимо определять свойства распределения по узлам каждой из матриц. Матрицы хранятся по столбцам.

На рисунке 4 представлена приведенная к одному процессору эффективность в зависимости от количества узлов при выполнении данного алгоритма умножения матриц на МТП-суперкомпьютере с узлами на базе J7-2. Пропускная способность одного сетевого интерфейса узла выбрана в одном варианте 1 ГБ/сек, а в двух других вариантах – 25.6 ГБ/сек.

Первый график соответствует пропускной способности интерфейса в 1 ГБ/сек. На 128 узлах эффективность одного процессора падает до 12%, и как следствие, суммарная производительность также оказывается низкой. В сегментах с матрицами A, B и C включено скремблирование. Средства профилирования модели МТП-суперкомпьютера показывают, что при выполнении теста загруженность сетевых интерфейсов достигает предела. Таким образом, именно пропускная способность сети является узким местом и ограничивает производительность.

При повышении пропускной способности сети до 25.6 ГБ/сек происходит повышение производительности (второй и третий графики). Сеть также загружена практически полностью, но уже не является узким местом. Тем не менее, производительность на один узел в среднем для второго и третьего графиков составляет примерно 25% и 50% соответственно.

Отличие между этими вариантами состоит в количестве арифметических операций, приходящихся на одну загрузку элемента из удаленного узла. Действительно, в третьем варианте для повышения локализации в сегментах, в которых находятся матрицы B и C, скремблирование было заменено на блочное распределение. Так как при этом матрицы хранятся по столбцам, то при использовании специального статического распределения работы, удается добиться, что треды процессоров каждого узла работают с частями матриц B и C, находящихся в памяти своего узла. Однако при этом элементы матрицы A в любом случае требуется пересылать через коммуникационную сеть. Для 128 узлов суммарная производительность лучшего варианта составляет 220 ГФлопс.

Приведенные результаты показывают, что в данном случае непосредственный запуск однопроцессорной программы в многопроцессорном режиме при использовании распределенной общей памяти хоть и позволяет достаточно просто получить высокие результаты по эффективности, но они ниже тех, которые могут быть получены для одного узла. Значительное ограничение здесь – пропускная способность сети.

Использование памяти в режиме скремблирования, то есть с отображением в псевдослучайном порядке на разные узлы – это модель равнодоступной памяти (UMA). В данном случае ее применение недостаточно эффективно, хотя очень просто позволяет получить параллельную программу. С другой стороны, приведенные выше результаты показали, что даже незначительный учет возможностей локализации (блочный способ отображения массивов B и C) позволил заметно увеличить эффективность в третьем варианте программы.

Отход от абсолютизации модели памяти UMA мы видим в эволюции систем Tera MTA, Cray MTA-2 [2, 3] до системы Cray XMT

(Eldorado) [4], в которой каждый узел имеет локальную память и добавлены способы управления локализацией через свойства сегментов. В МТП-суперкомпьютере приняты меры по дальнейшему развитию этого направления в сторону систем с памятью с неоднородным доступом (NUMA), в котором также сохраняется модель UMA.

Задача умножения матриц позволяет воспользоваться для повышения реальной производительности специальными алгоритмами преднакачки данных в памяти с разным быстродействием. Для NUMA/UMA систем, таких как Cray XMT, МТП, Cray X1E, SGI Altix это локальная память узла, в которую могут подкачиваться данные из глобально адресуемой памяти. Работа с такой иерархией памяти напоминает использование соответственно кэш-памяти процессора и оперативной DRAM-памяти узла, где находится этот процессор.

Может возникнуть вопрос, что нового для реализации такого алгоритма с подкачками данных может дать мультитредовая архитектура. Далее будет показано, что удается на части аппаратных тредов выполнять программируемые подкачки данных, а на другой части – выполнять вычисления. Это позволяет получить очень высокие результаты по эффективности.

Далее рассматривается разработанный многопроцессорный алгоритм MMult блочного умножения матриц, который реализован по аналогии с блочным алгоритмом SRUMMA, описанным в [5]. Высокоуровневое описание алгоритма дано на **Ошибка! Источник ссылки не найден.**.

Матрицы A, B и C делятся на блоки определенного размера. Названия матриц с индексами на рисунке 5 означают соответствующие блоки матриц. Блоки матрицы C статически делятся поровну между всеми ядрами процессоров всех узлов. Ядра между собой не взаимодействуют и не синхронизируются.

Для того, чтобы посчитать каждый блок матрицы C, сначала в локальную память подкачивается первый блок строки матрицы A, первый блок столбца матрицы B, а также обнуляется локальный буфер блока-результата $C_{i,j}$. Подкачки и обнуление выполняются параллельно и распараллеливаются сами при помощи несложного выделения тредовых вычислений. Затем происходит барьерная синхронизация всех участвующих тредов. После этого на каждой итерации цикла происходит локальное выполнение умножения подкачанных блоков со сложением с уже полученным результатом. На фоне умножения происходит асинхронная подкачка следующих двух блоков в другой набор локальных буферов. Затем снова происходит барьерная синхронизация и смена используемых буферов. После этого вычисления производятся над буферами с подкаченными данными, а в буфера, ранее использовавшиеся для вычислений, подкачиваются новые блоки

данных. После выхода из цикла происходит последнее вычисление и запись блока-результата $C_{i,j}$ в матрицу C.

foreach C_{ij} { // для всех блоков, выделенных данному ядру
copy($A_{i,0}$) //подкачка блока $A_{i,0}$ в первый локальный буфер A
copy($B_{0,j}$) // подкачка блока $B_{0,j}$ в первый локальный буфер B
$C_{i,j} = 0$ //обнуление локального блока-результата
barrier() // барьерная синхронизация трех предыдущих вызовов
for (b = 1; b < Nblocks; ++b) {
$C_{i,j} = C_{i,j} + A_{i,b-1}*B_{b-1,j}$ // счет, вызов локального dgemm
copy($A_{i,b}$) // подкачка во второй локальный буфер A
copy($B_{b,j}$) // подкачка во второй локальный буфер B
barrier()
switch() // смена используемых локальных буферов
}
switch()
$C_{i,j} = C_{i,j} + A_{i,Nblocks-1}*B_{Nblocks-1,j}$
barrier()
copy($C_{i,j}$) // запись результирующего блока в матрицу C
barrier()
}

Рисунок 5. Блочный алгоритм умножения матриц с подкачкой данных на фоне вычислений.

Таким образом, в этой мультитредовой программе уже применяется не гомогенная (однородная), а гетерогенная (неоднородная) тредовая модель. Треды ядра делятся на две части. Одна часть – вычислительная, другая – для работы с данными. Используется принцип DAE (decoupled access execute) - разделения работы по доступу к данным и вычислениям.

Видно, что предложенная локализация за счет подкачки блоков перед вычислением над ним похожа на приемы, которые применяют при эффективном использовании кэш-памяти в процессорах с традиционной архитектурой. Необходимо при этом отметить, что кэш-памяти традиционных коммерческих процессоров часто дополняются аппаратным механизмом предвыборки, который пытается распознать шаблон обращений и преднакачать данные в кэш. Для большей гибкости архитектуру многих процессоров дополнительно расширили введением команды prefetch, которая, тем не менее, выполняется в рамках общих ограничений одной программы (треда) вместе с вычислениями. Еще более мощное средство подкачки имеется в новом микропроцессоре Cell [6], в котором имеется программируемый MFC-

блок подкачки в локальную память посредством DMA-команд типа put/get по заданным адресам.

Более гибким и универсальным средством представляются программируемые аппаратные треды для подкачки данных МТП-суперкомпьютера и Cray XMT. Эти треды позволяют реализовать упомянутый принцип разделения работы по данным и вычислениям.

Организованная с применением мультитредовости подкачка данных все равно происходит через сеть, пропускная способность которой ограничена. Поскольку подкачка данных обладает спецификой, допускающей работу с блоками данных, а не с отдельными словами памяти, то для более эффективного использования пропускной способности сети во время процедуры копирования данных используются векторные операции загрузки из памяти. Дело в том, что при передаче данных по сети существуют накладные расходы, связанные с добавлением в сетевой пакет адресов данных, узла-получателя и другой служебной сетевой информации. Для более эффективного использования пропускной способности сети, которая, как показано было выше, является очень важным ресурсом, в систему команд процессоров J7/J10 были добавлены векторные команды считывания/записи, размер вектора – до восьми 64-разрядных слов. С другой стороны, важным параметром алгоритма является размер блока, при выборе размера которого важно соотношение количества обращений в глобальную память и операций с плавающей точкой. При размере блока NbxNb количество выполняемых операций равно $2*Nb^3+Nb^2$, а через сеть подкачивается Nb^2 элементов. При Nb равном 64 время подкачки следующего блока уже значительно уступает времени умножения блоков, поэтому сеть практически не влияет на время выполнения теста.

Для модернизированной программы результат получен очень высокий, сравнимый по эффективности с результатами на одном узле (см. таблицу 3). Между тем, есть и резервы, которые, возможно, будут учтены при доработке архитектуры МТП-суперкомпьютера. Суть здесь в следующем. Анализ динамических характеристик производительности во время выполнения программы на модели показывает, что асинхронная подкачка данных снижает эффективность использования арифметических функциональных устройств. Объяснение этому такое. Подкачка данных – это команды, которые нужно выдавать наряду с вычислительными командами. Чем больше выдано команд для подкачки, тем меньше выдается команд для вычислений. Это плата за универсальность программирования подкачек. Возможно, что простые случаи подкачек можно оптимизировать, введя аппарат DMA-команд и функциональность MFC-блока микропроцессора Cell на уровне устройств выполнения команд загрузки/записи (блок LSU) микропроцессоров J7/J10.

4. Сравнение результатов многопроцессорных систем на тестах умножения матриц и HPL

В таблице 3 представлено сравнение различных систем на различных вариантах умножения матриц и тесте HPL [7]. Данные по различным системам взяты из [8, 9].

Таблица 3. Сравнение различных систем на тесте умножения матриц и HPL.

Система	Тип проц.	Freq	NP	Тест	P	Эфф., %
HP Linux	Itanium2	1.5	1960	SRUMMA	10.3	88%
HP Linux	Itanium2	1.5	1960	pdgemm	8.6	74%
МТП	J7-2	0.5	256	MMult	0.8	86%
МТП	J7-2	0.5	2048	MMult	6.2	83%
МТП	J10-4	2.0	256	MMult	26.6	81%
МТП	J10-4	2.0	1024	MMult	103.9	79%
МТП	J10-4	2.0	4096	MMult	380.9	73%
HP Linux	Itanium2	1.5	1960	HPL	8.6	74%
Cray X1E	Cray X1E	1.3	248	HPL	3.4	76%
Cray XT3	Dual Core AMD	2.6	5202	HPL	43.5	80%
Blue Gene	PowerPC	0.7	1024	HPL	1.4	53%
Blue Gene	PowerPC	0.7	65536	HPL	259.2	71%

Обозначения: Freq – частота процессора (ГГц); P – производительность (ТФлопс); NP – количество процессоров.

На практике считается, что по пространственно-временной локализации обращений к памяти и соотношению вычислительных операций к операциям обращений к памяти тест умножения матриц можно рассматривать как аналог теста HPL списка top500 [10]. Примером подтверждения этому могут служить результаты, полученные на кластере HP Linux Cluster Platform 6000 rx2600 с сетью Quadrics. Эта система занимает 113 место в top500 на июнь 2007 года. На стандартном тесте PBLAS dgemm (pdgemm) и на HPL кластер показывает одинаковую производительность. На алгоритме SRUMMA наблюдается еще большая производительность, так как алгоритм использует вместо MPI оптимизированную асинхронную библиотеку коммуникаций, а также ряд приемов для лучшей производительности в сети. МТП-суперкомпьютер с процессором J7-2 показывает чуть меньшую абсолютную производительность и примерно такую же эффективность в сравнении с кластером HP Linux.

Принимая во внимание, что умножение матриц можно сопоставить с HPL, можно прогнозировать, что всего лишь на четырех

тысячах узлов МТП-суперкомпьютер с процессором J10-4 будет превосходить самый мощный суперкомпьютер в мире по top500 на июнь 2007 года Blue Gene, в котором используется значительно больше процессоров.

Таким образом, на примере задачи умножения плотнозаполненных матриц в данной работе показано, что МТП-суперкомпьютер, разрабатываемый для эффективного решения задач DIS-класса, может решать задачи CF-класса не менее эффективно, чем система с традиционной архитектурой.

Заключение

В данной работе приведены результаты исследования выполнения теста умножения плотнозаполненных матриц на перспективном МТП-суперкомпьютере с аппаратной поддержкой распределенной общей памяти. Рассмотрены два подхода к программированию такого класса задач с высокой степенью локализации данных.

Исследования такого рода на МТП-суперкомпьютере продолжаются, причем рассматриваются задачи и CF-класса, и DIS-класса.

Литература

1. L. Oliker et al., "The Potential of the Cell Processor for Scientific Computing". In Proceedings of the 3rd Conference on Computing Frontiers (Ischia, Italy, May 03 - 05, 2006).
2. C. Stork, "Exploring the Tera MTA by Example", 2000.
3. "Tera principles of operation", Tera Computer Company, 1998.
4. J. Feo, D. Harper, S. Kahan, P. Konecny, "Eldorado", Cray Inc, 2005.
5. M. Krishnan, J. Nieplocha, "SRUMMA: A Matrix Multiplication Algorithm Suitable for Clusters and Scalable Shared Memory Systems", IPDPS'04, 2004.
6. IBM DeveloperWorks. Cell Broadband Engine SDK programming handbook v 1.0, 2006.
7. High-Performance Linpack [HTML] (http://www.netlib.org/benchmark/hpl/).
8. HPC Challenge Benchmark [HTML] (http://icl.cs.utk.edu/hpcc/).
9. M. Krishnan, J. Nieplocha, "Optimizing Performance on Linux Clusters Using Advanced Communication Protocols: Achieving Over 10 Teraflops on a 8.6 Teraflops Linpack-Rated Linux Cluster", Proc. of International Conference on Linux clusters, 2005.
10. TOP500 Supercomputer Sites [HTML] (http://www.top500.org/).

Решетов Е.В.

Алгоритм нахождения величины компонент, на примере решения задачи определения минерального состава горных пород

В работе исследуется круг задач, связанных с определением компонентного состава некоторых объектов на основе измерений различных свойств данных объектов. Примером подобных задач являются определение компонентного состава воды, горных пород или газа, на основе измерения ряда физических величин: плотности, электрического сопротивления, радиоактивности и других параметров.

1. Общая постановка задачи

В общем виде задачу определения компонентного состава можно описать следующим образом.

Дан набор векторов измеренных параметров $y^k = (y_1^k, y_2^k, \ldots, y_N^k)$, $k = 1,2,\ldots,K$ (N – число измеренных параметров, K – число измерений). Необходимо найти набор векторов концентраций компонентов $x^k = (x_1^k, x_2^k, \ldots, x_M^k)$, $k = 1,2,\ldots,K$ (M - число компонент) исходя из уравнения

$$x^k * A = y^k \qquad k = 1,2,\ldots,K$$

или что тоже самое

$$XA = Y \tag{1}$$

где $A = \begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1N} \\ A_{21} & A_{22} & \ldots & A_{2N} \\ \ldots & \ldots & \ldots & \ldots \\ A_{M1} & A_{M2} & \ldots & A_{MN} \end{pmatrix}$ — матрица влияния компонентов на измеренные параметры, X – матрица составленная из векторов x^k, Y – матрица составленная из векторов y^k.

В случае если N = M (число измеренных параметров совпадает с числом компонент) и матрица A явно задана, то задача имеет единственное решение

$$x^k = y^k * A^{-1} \tag{2}$$

Но возможно и более сложные случаи, когда $N \neq M$, на x^k заданы ограничения x_{min} и x_{max}, $x_{min} \leq x_i^k \leq x_{max}$, $i = 1...M$, или матрица A задана не явно, а заданы лишь ограничения на её значения A_{min} и A_{max}, $A_{ijmin} \leq A_{ij} \leq A_{ijmax}$, $i = 1,2,\ldots,M$, $j = 1,2,\ldots,N$.

Таким образом, мы приходим к задаче математического программирования.

$$\min_{A \in G} F(A, Y, A_{\min}, A_{\max}, x_{\min}, x_{\max})$$

Найти А из области G, заданной A_{min} и A_{max}, минимизирующую функцию F (априорно полученная функция, оценивающая "качество" получаемого решения X с учётом ограничений наложенных на него).

2. Задача определения минерального состава горных пород

Рассмотрим задачу определения минерально-компонентного состава горной породы по комплексу спектрометрического гамма каротажа (концентрации калия, тория), нейтрон-нейтронного каротажа (водородосодержание), и гамма-гамма каротажа (плотность), с предварительной петрофизической настройкой по материалам исследования образцов керна.

Рассмотрим следующую модель горной породы. Пусть в состав горной породы (Гп) входят следующие основные составляющие (макрокомпоненты) – кварц (Кв), полевые шпаты (ПШ), флюид (вода, нефть, газ или их смесь) саполняющий поры пространства между макрокомпонентами, Гл1 и Гл2 – глинистые менералы различающиеся по ряду физических свойств.

Очевидно, что для таких измеренных величин как плотность, водородосодержание и т.д. измеренная величина в единице объёма будет равна сумме произведений соответствующих физических величин, свойственным каждой макрокомпоненте, умноженным на концентрации этих макрокомпонент. То есть строится система петрофизических уравнений:

$$\begin{cases} K_{Гл1}*\delta_{Гл1}*C_K^{Гл1} + K_{Гл2}*\delta_{Гл2}*C_K^{Гл2} + K_{ПШ}*\delta_{ПШ}*C_K^{ПШ} + K_{Кв}*\delta_{Кв}*C_K^{Кв} + K_{П}*\delta_{Фл}*C_K^{Фл} = C_K*\delta_{Гп} \\ K_{Гл1}*\delta_{Гл1}*C_{Th}^{Гл1} + K_{Гл2}*\delta_{Гл2}*C_{Th}^{Гл2} + K_{ПШ}*\delta_{ПШ}*C_{Th}^{ПШ} + K_{Кв}*\delta_{Кв}*C_{Th}^{Кв} + K_{П}*\delta_{Фл}*C_{Th}^{Фл} = C_{Th}*\delta_{Гп} \\ K_{Гл1}*\delta_{Гл1} + K_{Гл2}*\delta_{Гл2} + K_{ПШ}*\delta_{ПШ} + K_{Кв}*\delta_{Кв} + K_{П}*\delta_{Фл} = \delta_{Гп} \\ K_{Гл1}*\omega_{Гл1} + K_{Гл2}*\omega_{Гл2} + K_{ПШ}*\omega_{ПШ} + K_{Кв}*\omega_{Кв} + K_{П}*\omega_{Фл} = \omega_{Гп} \\ K_{Гл1} + K_{Гл2} + K_{ПШ} + K_{Кв} + K_{П} = 1 \end{cases} \quad (3)$$

где:

K_i - объемные доли i-го макрокомпонента в горной породе;

δi, $\delta_{Гп}$- плотности i-го макрокомпонента и горной породы;

C_K, C_{Th} - массовые доли концентрации калия и тория в породе;

$\omega_{фл}$ – водородный индекс порового вещества (водородный индекс – водородосодержание в долях относительно содержания водорода в воде принятого за единицу);

$\omega_{Гп}$ – условный водородный индекс горной породы, рассчитанный по зависимости, построенной для матрицы, состоящей из кальцита ($CaCO_3$) с пористостью, заполненной водой.

$\omega_{гл1}$, $\omega_{гл2}$, $\omega_{пш}$, $\omega_{кв}$ – удельные вклады водородных индексов макрокомпонент глина1, глина2, полевые шпаты, кварц в условный водородный индекс горной породы;

Для данного кванта глубины составляется система уравнений, соответствующих всем физическим величинам, определенным по керну и ГИС. Дополнительным уравнением, связывающим все компоненты, является уравнение математического баланса, т.к. сумма всех компонент, складывающих горную породу, равна единице.

Перейдём от системы (3) к общему виду (1):

$y^k = (C_K{}^k * \delta^k,\ C_{TH}{}^k * \delta^k,\ \omega^k,\ \delta^k)$ - вектор-строка, образованная данными результатов обработки индивидуальных методов каротажа, соответствующими одной глубине;

$x^k = (Гл1^k, Гл2^k, Кпш^k, Кв^k, Кп^k)$ – вектор-строка концентраций макрокомпонент.

$$A = \begin{pmatrix} \delta_{Гл1} * C_K^{Гл1} & \delta_{Гл2} * C_K^{Гл2} & \delta_{ПШ} * C_K^{ПШ} & \delta_{Кв} * C_K^{Кв} & \delta_{Фп} * C_K^{Фл} \\ \delta_{Гл1} * C_{Th}^{Гл1} & \delta_{Гл2} * C_{Th}^{Гл2} & \delta_{ПШ} * C_{Th}^{ПШ} & \delta_{Кв} * C_{Th}^{Кв} & \delta_{фл} * C_K^{Гл2} \\ \delta_{Гл1} & \delta_{Гл2} & \delta_{ПШ} & \delta_{Кв} & \delta_{Фл} \\ \omega_{Гл1} & \omega_{Гл2} & \omega_{ПШ} & \omega_{Кв} & \omega_{Фл} \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} —$$

матрица петрофизических характеристик каждой объёмной макрокомпоненты в единицах объёма.

В качестве критериев для поиска решения задачи даются значения петрофизических коэффициентов, полученные в результате исследований каменного материала:

1. минимальные значения петрофизических характеристик Amin;
2. максимальные значения петрофизических характеристик Amax;
3. ограничения по содержанию суммы глинистых минералов и пористости.

Так как матрица А явно не задана, то для решения задачи необходимо произвести её поиск.

Критерием выбора петрофизической матрицы для коллекторов является минимальное количество отрицательных значений содержаний макрокомпонент и непревышение суммы глинистых минералов заданной величины. Для глин также подбираются параметры, но при этом значение суммы глинистых минералов должно быть больше критического. Дополнительным параметром поиска является близость найденных параметров к математическому ожиданию значений петрофизических характеристик макрокомпонент, полученных на основе исследования кернового материала.

В качестве возможного подхода для поиска матицы A может выступать наложение равномерной сетки на N-мерный куб, ограниченный значениями A_{min} и A_{max}, которые задают область поиска G:

$$\{ A \in G; A = \begin{pmatrix} A_{11} & \dots & A_{15} \\ A_{21} & \dots & A_{25} \\ \dots & \dots & \dots \\ A_{51} & \dots & A_{55} \end{pmatrix}; \; A_{мин\ i\ j} \le A_{ij} \le A_{max\ ij}; \; i=1,2,\dots,5, \; j=1,2,\dots,5\} \quad (4)$$

и полный перебор по всем возможным узлам сетки S

$$\{ \forall A \in S : A \in G; A = \begin{pmatrix} A_{11} & \dots & A_{15} \\ A_{21} & \dots & A_{25} \\ \dots & \dots & \dots \\ A_{51} & \dots & A_{55} \end{pmatrix}; \; A_{ij} = A_{min\ ij} + l_{ij} * (A_{max\ ij} - A_{min\ ij}) / m_{ij}; \; l_{ij} = 1,2,\dots,m_{ij}; \; i=1,2,\dots,5, \; j=1,2,\dots,5\} \quad (5)$$

где m_{ij} – число шагов сетки в соответствующем измерении

На каждом кванте глубины производится поиск матрицы A и решение уравнения (3). Так как поиск матриц для всех квантов глубины является времяёмким процессом, то, для ускорения расчетов без существенной потери в точности, было выбрано следующее правило. Рассматриваются не все кванты глубины, соответствующие коллекторам, а только те, в которых коллектора являются наиболее чистыми, т.е. содержат минимальное количество глинистых минералов и, как следствие, тория. В глинах же выбираются кванты глубины с наибольшим содержанием глинистых минералов и, как следствие, максимальной концентрацией тория. Эмпирическим путём было получено, что для получения надёжных результатов при подборе матрицы A, достаточно использовать 60 квантов глубины каждого литотипа. Если в обрабатываемом разрезе меньше 60 квантов, то для выбора матрицы нужно использовать все кванты глубины, соответствующие литотипу.

Данный метод имеет следующие недостатки: большое количество вычислений $\prod k_{ij}$, и недостаточная точность при малом числе шагов сетки — решение в этом случае может находиться между узлами сетки.

3. Применение эволюционного алгоритма

В качестве более эффективного метода решения поставленной задачи был предложен эволюционный алгоритм, основанный на методе случайного поиска.

Метод случайного поиска обладает достаточно хорошей сходимостью, а эволюционный алгоритм был применен с целью избежать попадания в локальный минимум.

При решении задачи поиска матрицы A в качестве гена принималась матрица из области поиска G (4). Начальная популяция – из случайно расположенных генов в области поиска G. На каждом шаге выбирались 20 генов-победителей с наибольшим фитнесом, от каждого гена-победителя получалось 20 генов потомков. Итого 20*20 = 400 генов потомков. Число 20 было выбрано в ходе проведения экспериментальных исследований с различными количествами генов-победителей и генов-потомков.

В качестве фитнес-функции рассматривалась доля квантов глубины, в которых: 1) значения меньше нуля имеет концентрация хотя бы одной макрокомпоненты; 2) сумма глинистых минералов больше заранее заданных величин. В качестве дополнительно параметра фитнес-функции рассматривался модуль расстояния от гена до центра области G (математического ожидания, полученного на керне), но умноженный на 10^{-6}, то есть ему придаётся существенно меньшее значение, чем доля корректно рассчитанных квантов глубины.

Гены-потомки получаются от генов-победителей путём случайного выбора из N-мерной окрестности (20и-мерной для нашего случая) вокруг точки, заданной матрицей для данного гена:

$$\{ A \in G'; A = \begin{pmatrix} A_{11} & \dots & A_{15} \\ A_{21} & \dots & A_{25} \\ \dots & \dots & \dots \\ A_{51} & \dots & A_{55} \end{pmatrix}; A_{\text{род ij}} - d_{ij} < A_{ij} < A_{\text{род ij}} + d_{ij};\ i=1,2,...,5,$$

$$j=1,2,...,5\} \qquad (6)$$

где G' – область выбора генов-потомков;

$A_{род}$ – ген-родитель;

$$d = \begin{pmatrix} d_{11} & \dots & d_{15} \\ d_{21} & \dots & d_{25} \\ \dots & \dots & \dots \\ d_{51} & \dots & d_{55} \end{pmatrix}$$ – размер окрестности. $d_{ij}=(A_{max\ ij} - A_{min\ ij}) / 20;$

i=1,2,...5, j=1,2,...,5.

Размер окрестности изначально равен одной десятой размера исходной области, в которой производится поиск решения, но затем в процессе работы эволюционного алгоритма с каждым шагом пропорционально уменьшается, если ген остаётся "живым". При этом полученные гены-потомки выбираются по закону Гаусса с математическим ожиданием, равным гену-родителю, и размером окрестности, соответствующим величине дисперсии, умноженной на 3.

В качестве мутации на каждом шаге в популяцию добавляется 400 (число генов, получаемых в результате кроссовера) генов, расположенных произвольным образом в исходной области поиска. Таким образом, популяция состоит из 800 генов (400 — кроссовер и 400 — мутация).

Работа алгоритма останавливается после 50 итераций.

Так как в каждой итерации в популяции содержатся гены победители с предыдущей итерации, то решение, получаемое в последующих итерациях, будет заведомо не хуже. Случайный поиск вокруг генов-победителей позволяет более точно находить локальный минимум. А добавление случайных генов из всей области поиска даёт возможность попадать в различные локальные минимумы, которые могут оказаться искомым глобальным минимумом.

Заключение

Так как решения, отыскиваемые с помощью эволюционного алгоритма, не привязаны к каким-либо конкретным точкам, как в случае перебора по сетке, то решение может находиться более точно. Проведенные исследования показали, что, в среднем, доля корректно рассчитанных квантов глубины при поиске матрицы по равномерной сетке составляет 70 – 90%. А доля корректно рассчитанных квантов глубин с использованием эволюционного алгоритма 80 – 95% для тех же каротажных данных и том же времени вычислений.

На рисунках 1 и 2 показано сопоставление результатов, полученных в ходе решения задачи с помощью эволюционного алгоритма (кривые), с результатами полученными в результате анализа кернового материала (точки), для параметров "коэффициент пористости" и "сумма глинистых минералов". Как видно, прослеживается достаточно хорошая корреляция, что показывает применимость данного метода для определения минерально-компонентного состава горной породы.

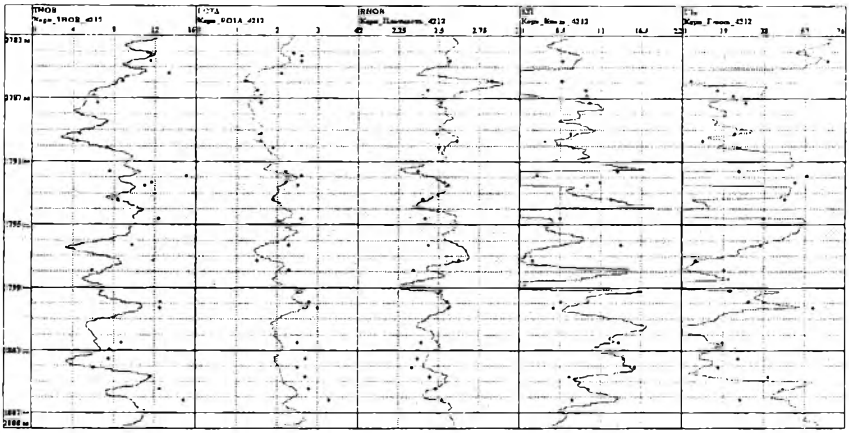

Рисунок 1. Сопоставление данных полученных в результате расчетов с данными исследований керна, для отложений ЮС1.

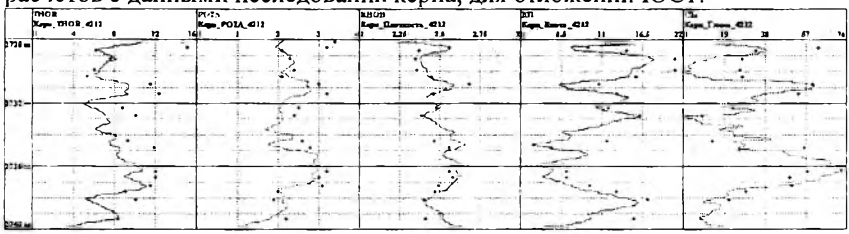

Рисунок 2. Сопоставление данных полученных в результате расчетов с данными исследований керна, для отложений ЮС2.

Литература

1. Калмыков Г.А, Коротков К.В., Теленков В.М., Ревва М.Ю. Применение комплекса радиоактивных методов исследований скважин для оценки емкостных свойств терригенных коллекторов Западной Сибири (на примере пласта ПК19) // Геология нефти и газа №1, 2005г. стр. 36-44.
2. Калмыков Г.А. Методика определения минерально-компонентного состава терригенных пород в разрезах нефтегазовых скважин по данным комплекса ГИС, включающего спектрометрический ГК Диссертация на соискание степени кандидата технических наук, М., ВНИИгеосистем, 2001.
3. Goldberg D.E. Genetic Algorithms in Search, Optimization and Machine Learning. - Addison Wesley Publishing Company, Inc., 1989.
4. John R. Koza Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems).
5. Денисов Д.В. Сходимость метода случайного поиска с постоянным шагом // “Вопросы оптимизации и управления”. Изд-во МГУ, 1978.

Погуляй Е.В.

Разработка алгоритма навигации биомолекулярного нанороботa для задач фармокологии

1. Введение

Работа посвящена моделированию лекарственных средств при помощи метода распознавания образов. Лекарства можно получить как из природных источников, так и при помощи создания искусственных молекул, ориентированных на достижение определенного эффекта. Это направление объединяет несколько областей научной деятельности: биоинформатику, физику и нанотехнологии.

Современная разработка лекарств часто включает отбор небольших молекул по способности взаимодействовать с заранее выбранным протеином. Таким образом, создается одна или несколько коллекций молекул, эффективно взаимодействующих с определенным целевым протеином. Также разработка лекарств включает поиск молекул по их способности регулировать биологические процессы в клетке или всем организме в целом без учета их взаимодействия с выбранным протеином. На третьем шаге проводится синтез конечного лекарства из найденных молекул, обладающего всеми необходимыми свойствами [1].

Рассмотрим более подробно методы, применяемые при моделировании лекарств. Одними из первых начали применяться различные модификации генетического алгоритма. Генетический алгоритм представляет собой метод, который оперирует частями информации словно природа генами в процессе эволюции [2].

Для любого приложения генетического алгоритма сначала определяется формальное представление «гена». В данном случае алгоритм оперирует строками чисел. В качестве чисел можно рассматривать трехмерные координаты каждого из атомов, составляющих молекулу, но в этом случае процедура мутации будет создавать поврежденные молекулы, в которых некоторые атомы будут лежать слишком далеко. Другое представление модели: через торсионные углы. Протеин представляется в виде множества торсионных углов при допущении стандартной геометрии связей. Длины связей (бондов) и углы между ними принимаются константами и не могут быть измены в процессе работы алгоритма. Это предположение отличается от реальности, но торсионные углы дают достаточно степеней свободы для представления любой незначительной деформации.

Аналогичное представление молекулы протеина в виде последовательности чисел применяется при других исследованиях, проводимых при помощи методов распознавания образов. При решении задачи при помощи нейросетей, последовательность чисел рассматривается как вектор в N мерном пространстве, где N – число элементов в последовательности, каждому вектору присваивается метка класса, разбивающая множество векторов на подходящие и неподходящие по некоторому критерию. Далее эти вектора используются для обучения классификатора, построенного на основе нейросети [3].

Данная работа посвящена решению задачи о навигации нанобота и его взаимодействии с целевым протеином. Переформулируем исходную постановку задачи, сведя ее к решению задачи распознавания образов. Классификацию будем проводить при помощи метода опорных векторов, т.к. он имеет необходимые нам преимущества при работе с пространствами признаков большой размерности.

Цель — построить метод, позволяющий решать одну из наиболее ресурсоемких задач области моделирования лекарств, при помощи методов распознавания образов, провести его численное исследование.

2. Постановка задачи

Первый шаг в разработке нового лекарства обычно заключается в определении и выделении целевого рецептора, с которым должно связаться потенциальное лекарство. Он состоит из тестирования большого числа небольших молекул на их способность связываться с целевым рецептором. Это приводит исследователей к задаче о разделении структур на активные (связывающиеся) и неактивные (не связывающиеся). Решение этой задачи может быть использовано в последующем создании новых структур, не только связывающихся с целевым рецептором, но и обладающих другими возможностями, необходимыми для успешного лекарства [1].

Рассмотрим эту задачу подробнее: у нас есть молекула, на которую мы хотим воздействовать и конкретная структура (будем называть ее нанороботом), действие которой мы хотим проверить. Робот будет взаимодействовать с молекулой при выполнении условия (1):

$$\sum_{i=1}^{n} R_i \le R_{пороговое} \tag{1}$$

$$R_i = \sum_{j=1}^{N} r_{ij}^2 \tag{2}$$

$$r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \quad (3)$$

где индексация i идет по атомам робота, а j по атомам протеина.

Даже после того, как исследователь предоставит целевой рецептор и молекулу лекарства, взаимодействие которых надо проверить, остается решить еще одну задачу, требующую большого объема вычислений. Из-за наличия теплового движения структура протеина постоянно изменяется, т.к. у части связей фиксированными являются только расстояния между атомами, а углы, под которыми они расположены, могут меняться. Поэтому лекарство должно одинаково хорошо взаимодействовать с различными вариантами одного и того же протеина. Это приводит к необходимости генерировать по предоставленному образцу целого множества молекул, и для каждой из них проверять способность лекарства взаимодействовать с ней. Генерация базы данных представляет достаточно сложную задачу и производится при помощи методов молекулярной динамики.

Необходимость рассматривать множество молекул влияет на постановку задачи (1)–(3) следующим образом: координаты атомов целевой молекулы становятся переменными, зависящими от способных изменяться углов между связями. Обозначим эти углы через γ_i: $x_j = f_j^{\,x}(\gamma_1,...,\gamma_K)$, $y_j = f_j^{\,y}(\gamma_1,...,\gamma_K)$, $z_j = f_j^{\,z}(\gamma_1,...,\gamma_K)$, т.е. условие (1) примет следующий вид:

$$\min_{x_i,y_i,z_i} \max_{X,Y,Z} \sum_{i=1}^{n} R_i \le R_{пороговое} \quad (4)$$

где $X = \{x_j\}$, $Y = \{y_j\}$, $Z = \{z_j\}$ допустимые множества координат атомов протеина.

Сформулируем исходную задачу, сведя ее к решению задачи распознавания образов. Распознавание будем проводить при помощи метода опорных векторов, т.к. он имеет необходимые нам преимущества при работе с пространствами признаков большой размерности.

Рассмотрим пространство признаков, вектора которого имеют следующий вид:

$$(h, \alpha_1,...,\alpha_{N-2}, \beta_1,...,\beta_{N-3}, y),$$

где $(h, \alpha_1,...,\alpha_{N-2}, \beta_1,...,\beta_{N-3})$ - координаты вектора в пространстве признаков, а $y \in \{\pm 1\}$ определяет принадлежность образа к одному из двух классов: происходит взаимодействие между молекулой протеина и роботом или нет.

Пространство признаков было выбрано следующим образом:

h — расстояние от робота до молекулы.

α_i — угол между связями (бондами) одного атома, т.е. векторами n_1 и n_2.

β_i — угол между плоскостями, в которых лежат соседние тройки атомов.

3. Описание алгоритма SHAKE

3.1. Молекулярная динамика со связями

Моделирование методом молекулярной динамики состоит в численном решении уравнений Ньютона

$$m_i \frac{d^2 r}{dt^2} = f_i,\ f_i = -\frac{\partial U}{\partial r_i} \tag{5}$$

Силы f_i, действующие на атом, вычисляются по заданной потенциальной энергии $U = U(r_1, r_2, ..., r_{N_p})$, которая зависит от полного набора $3N_p$ атомных координат. Вид этой функции определяется либо из опытных соображений, либо вычисляется методами квантовой механики. При моделировании конденсированных систем учитываются несферические жесткие связи, которые имеют вращательные степени свободы. Это требует добавления уравнений вращательного движения.

3.2. Численные алгоритмы молекулярной динамики со связями

Алгоритм SHAKE [4].

Рассмотрим наименьшую нетривиальную цепочку, состоящую из трех атомов, связанных твердыми невесомыми связями с длинами l_1 и l_2. Тогда уравнения связей будут иметь следующий вид:

$$\sigma_1(r_{12}) = |r_{12}|^2 - l_1^2 = 0 \tag{6}$$

$$\sigma_2(r_{23}) = |r_{23}|^2 - l_2^2 = 0 \tag{7}$$

где $r_{ij} = r_j - r_i$.

Уравнения жестких связей (6), (7) будем учитывать в уравнениях движения с помощью множителей Лагранжа.

Алгоритм:

1. Обозначим положения атомов в момент времени t^n через r_i^n. Интегрируем уравнения движения на один временной шаг, полагая $a_k = 0$, т.е. без учета связей. Обозначим полученные предварительные позиции (в момент времени t^{n+1}) r_i'. Эти положения не удовлетворяют уравнениям связей, и значения $\sigma_1(r_{12}')$, $\sigma_2(r_{23}')$ будут равны ненулевым величинам σ_1' и σ_2'.
2. Производим коррекцию координат следующим образом:

$$r_1^{n+1} = r_1' + \frac{a_1}{m_1} r_{12}^n$$

$$r_2^{n+1} = r_2' - \frac{a_1}{m_2} r_{12}^n + \frac{a_2}{m_2} r_{23}^n \qquad (8)$$

$$r_3^{n+1} = r_3' - \frac{a_2}{m_3} r_{23}^n$$

a_k получаем из уравнения (5), потребовав, чтобы скорректированные положения удовлетворяли уравнениям связи (6), (7). Пренебрежем квадратичными членами и далее будем решать линейную систему уравнений. Из полученной СЛАУ вычисляем a_1, a_2

3. Подставляем полученные a_1, a_2, переход на шаг 1.

Решение линеаризованных уравнений включает обращение матрицы. Чтобы этого избежать, введем еще одно упрощение: проходя по молекулярной цепочке с одного конца до другого, мы будем рассматривать только одну связь на атом:

- Сначала связь r_{12} исправляется перемещением 1 и 2 атомов.
- При исправлении следующей связи r_{23}, разрушается предыдущая, что будет исправлено дальнейшими итерациями.
- Несколько проходов по цепочке уменьшают ошибки, привнесенные пренебрежением квадратичными членами и учетом одной связи в один момент времени.

4. Описание метода опорных векторов

4.1. Метод опорных векторов

Рассмотрим случай двух линейно разделимых классов в N-мерном пространстве. Тогда разделяющая поверхность будет иметь вид

$$(w \cdot x) + b = 0 \quad w \in R^N, b \in R \tag{1.1}$$

а решающая функция вид

$$f(x) = \operatorname{sgn}((w \cdot x) + b), \tag{1.2}$$

Метод опорных векторов заключается в решении следующей задачи оптимизации:

$$\min_{W,b} imize \frac{1}{2}\|w\|^2$$

При условии $y_i \cdot ((w \cdot x_i) + b) \geq 1, i = 1,...,m.$ (1.3)

Способом разрешить (1.3) является решение двойного Лагранжиана:

$$\max_{\alpha \geq 0}(\min_{W,b} L(w,b,\alpha)), \tag{1.4}$$

где

$$L(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i (y_i \cdot ((x_i \cdot w) + b) - 1). \tag{1.5}$$

После дифференцирования $L(w,b,\alpha)$ по b и w и подстановки полученных результатов, задача (1.4) приобретает вид [5]:

$$\max_{\alpha \in R^m} imize \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{1.6}$$

При условии $\alpha_i \geq 0, i = 1,...,m,$ и $\sum_{i=1}^{m} \alpha_i y_i = 0.$ (1.7)

4.2. Структурный и эмпирический риск

Рассмотрим проблему распознавания образов на более формальном уровне. В задаче распознавания образов с двумя классами мы стремимся оценить функцию

$$f : X \to \{\pm 1\},$$

основываясь на входных обучающих данных. Мы предполагаем, что данные являются независимыми с некоторой неизвестной (но фиксированной) функцией распределения вероятностей $P(x, y)$. Наша

цель найти функцию, которая правильно классифицирует новые данные (x, y), т.е. мы хотим иметь функцию $f(x) = y$ для примеров (x, y), которые будут произведены $P(x, y)$.

Если мы не устанавливаем никакого ограничения на класс функций, из которых выбираем нашу оценку f, то даже функция, которая хорошо классифицирует тренировочные образы, например, удовлетворяя $f(x_i) = y_i$ для всех $i = 1,...,m$, может не делать хорошего вывода для новых данных. Чтобы увидеть это, заметим, что для каждой функции f и любого тренировочного множества $(\bar{x}_1, \bar{y}_1),...,(\bar{x}_{\bar{m}}, \bar{y}_{\bar{m}}) \in R^N \times \{\pm 1\}$, удовлетворяющего условию $\{\bar{x}_1,...,\bar{x}_{\bar{m}}\} \cap \{x_1,...,x_m\} = \{\}$, существует другая функция f^*, такая что $f^*(x_i) = f(x_i)$ для всех $i = 1,...,m$, но $f^*(\bar{x}_i) = f(\bar{x}_i)$ для всех $i = 1,...,\bar{m}$. Поскольку нам дают только обучающие данные, мы не имеем ни каких средств отбора для выбора из этих двух функций. Следовательно, только минимизация ошибки обучения (или эмпирического риска)

$$R_{emp}[f] = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left|f(x_i) - y_i\right|,$$

не подразумевает маленькую испытательную ошибку (называемую риском), усреднённую по всему множеству тестовых примеров.

$$R[f] = \int\frac{1}{2}\left|f(x) - y\right|dP(x, y)$$

Статистическая теория обучения, или VC (Vapnik-Chervonenkis) теория, показывает, что класс функций, из которого выбирается f (функция, обладающая подходящей разделительной способностью для обучающих данных), обязательно должен быть ограничен. VC теория обеспечивает существование границ для структурного риска. Минимизация этих границ, которые зависят от эмпирического риска и мощности класса разделяющих функций, приводит к принципу структурной минимизации риска. Самое известное понятие VC теории - это VC размерность h, определенное как наибольшее число точек, разделимых всеми возможными способами при помощи функций данного класса. [6] Пример VC границы: если $h < m$ это VC размерность класса функций, которые алгоритм обучения может сгенерировать, тогда для всех функций в этом классе, с вероятностью промаха $1 - \eta$, границей будет следующее неравенство:

$$R(f) \le R_{emp}(f) + \phi(\frac{h}{m}, \frac{\log(\eta)}{m}) \tag{2.1}$$

где ϕ определяется как

$$\phi(\frac{h}{m}, \frac{\log(\eta)}{m}) = \sqrt{\frac{h(\log\frac{2m}{h} + 1) - \log(\frac{\eta}{4})}{m}}. \tag{2.2}$$

Более сильные границы могут быть получены, используя концепцию *VC энтропии* или *функции роста* [5]. Они тяжелее для оценки, но при этом играют фундаментальную роль в концептуальной части VC теории.

3. ν-смягчение границы классификатора, основанного на опорных векторах

На практике разделяющая гиперплоскость может не существовать (например, сильный шум размывает границы между классами). Для разрешения этой проблемы введем дополнительные переменные:

$$\xi_i \ge 0, i = 1,...,m \tag{3.1}$$

нужные для ослабления условия:

$$y_i \cdot ((w \cdot x_i) + b) \ge 1 - \xi_i, i = 1,...,m. \tag{3.2}$$

А так же ν-параметризацию, $\nu \in [0,1]$. для смягчения границы. ν определяет нижнюю и верхнюю границу числа опорных векторов, лежащих с неверной стороны от разделяющей гиперплоскости.

Первичной задачей этого приближения будет следующее:

$$\min_{W \in H, \xi \in R^m, \rho, b \in R} imize(\tau(w, \xi, \rho)) = \frac{1}{2}\|w\|^2 - \nu\rho + \frac{1}{m}\sum_{i=1}^{m}\xi_i \tag{3.3}$$

При условиях:

$$y_i \cdot ((w \cdot x_i) + b) \ge 1 - \xi_i, i = 1,...,m \tag{3.4}$$

и $\xi_i \ge 0, \rho \ge 0$ (3.5)

Для объяснения смысла ν введем понятие "граничная ошибка" — так мы обозначим тренировочные точки с $\xi_i > 0$. Эти точки либо ошибки, либо лежат в пределах граничной области. Формально, отношение граничных ошибок к общему числу точек имеет вид:

$$R_{emp}^{\rho}[g] = \frac{1}{m}\left|\{i | y_i g(x_i) < \rho\}\right| \tag{3.6}$$

Теперь установим двойственную задачу $\nu - SV$ классификатора. Рассмотрим Лагранжиан, использующий множители $\alpha_i, \beta_i, \delta \geq 0$:

$$L(w,\xi,b,\rho,\alpha,\beta,\delta) = \frac{1}{2}\|w\|^2 - \nu\rho + \frac{1}{m}\sum_{i=1}^{m}\xi_i$$
$$-\sum_{i=1}^{m}(\alpha_i(y_i((x_i,w)+b)-\rho+\xi_i)+\beta_i\xi_i+\delta\rho) \tag{3.7}$$

Эта функция должна быть минимизирована по первичным переменным w,ξ,β,ρ и максимизирована по вторичным α_i,β_i,δ. Вычислив соответствующие частные производные и приравняв их к нулю, получаем следующие условия:

$$w = \sum_{i=1}^{m}\alpha_i y_i x_i. \tag{3.8}$$

$$\alpha_i + \beta_i = \frac{1}{m} \tag{3.9}$$

$$\sum_{i=1}^{m}\alpha_i y_i = 0 \tag{3.10}$$

$$\sum_{i=1}^{m}\alpha_i - \delta = \nu \tag{3.11}$$

Подставляя (3.8), (3.9) в L, используя $\alpha_i, \beta_i, \delta \geq 0$ и применяя ядерную функцию, получаем следующую квадратичную оптимизационную задачу [7]:

$$\max_{\alpha\in R^m} imize\, W(\alpha) = -\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j k(x_i,x_j) \tag{3.12}$$

При условиях:

$$0 \leq \alpha_i \leq \frac{1}{m} \tag{3.13}$$

$$\sum_{i=1}^{m}\alpha_i y_i = 0 \tag{3.14}$$

$$\sum_{i=1}^{m}\alpha_i \geq \nu \tag{3.15}$$

В результате решающая функция примет форму:

$$f(x) = \mathrm{sgn}(\sum_{i=1}^{m} \alpha_i y_i \cdot k(x, x_i) + b). \qquad (3.16)$$

5. Результаты вычислений

1. Определение зависимость риска от параметра ядерной функции.

Исследование проводилось с целью определения оптимального значения параметра ядерной функции для данной задачи. Обучение классификатора проводилось на базе данных, содержащей N=300 векторов. Из рисунка 1 видно, что оптимальное значение параметра $\gamma = 0.05$.

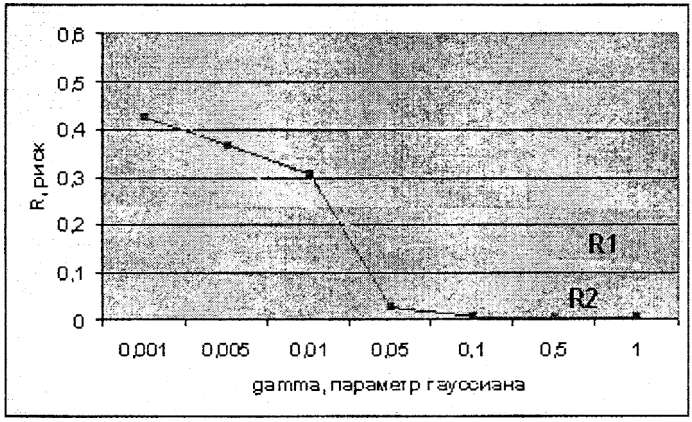

Рисунок 1. Зависимость риска классификации (R1) и ошибки обучения (R2) от параметра ядерной функции.

2. Определение зависимости риска от размера обучающей базы данных.

При построении классификатора в качестве ядерной функции использовался гауссиан $k(x_1, x_2) = e^{-\gamma(x_1, x_2)}$ с параметром $\gamma = 0.05$.

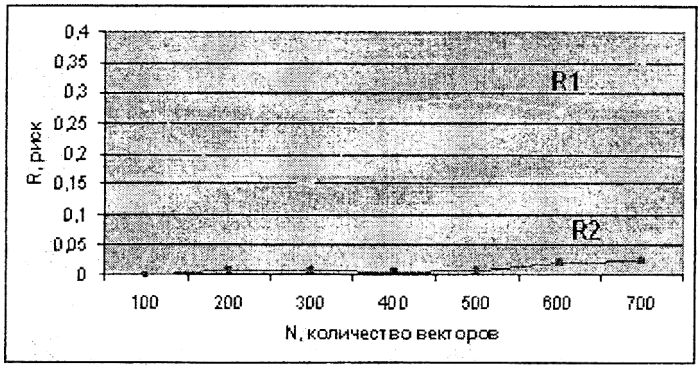

Рисунок 2. Зависимость риска классификации (R1) и ошибки обучения (R2) от размера обучающей базы данных.

Из рисунка 2 можно сделать следующие выводы:

1. При малом размере обучающей базы данных ошибка обучения мала, т.к. разделяющая поверхность хорошо аппроксимирует немногочисленные данные, но риск классификации высок, т.к. обучающих данных недостаточно для локализации кластеров точек.
2. При среднем размере базы данных риск классификации достигает минимального значения, что объясняется достаточной представительностью выборки для определения кластеров.
3. При дальнейшем увеличении базы данных происходит переобучение классификатора, отражающееся на росте риска классификации.

Полученные результаты согласуются со статистической теорией Вапника [8].

6. Заключение

В ходе работы исходная оптимизационная задача была сведена к задаче распознавания образов. На основе метода опорных векторов был построен метод, позволяющий решать одну из наиболее ресурсоемких задач области моделирования лекарств, а именно поиска небольших молекул, способных взаимодействовать с выбранным целевым протеином.

Была написана программа, реализующая построенный метод и проведено численное исследование на модельных данных. Результаты исследования согласуются со статистической теорией Вапника [8].

В дальнейшем планируется исследовать построенный метод на протеинах, состоящих из сотен атомов, т.е. потребуется расширить размерность пространства признаков. На этом этапе будет использовано

главное преимущество метода опорных векторов, позволяющего работать с пространствами признаков большой размерности.

Литература

1. Industrial Pharmaceutical Biotechnology. H. Klefenz, Wiley-VCH Verlag GmbH, 2002.
2. Adaptation in Natural and Artificial Systems, 2nd Ed. J. H. Holland, MIT Press, 1992.
3. Neural Networks as Data Mining Tools in Drug Design. J. Gasteiger, A. Teckentrup, L. Terfloth, S. Spycher J. Phys. Org. Chem., 16, pp. 232-245 (2003).
4. Numerical integration of the cartesian equations of motion of a system with constraints: Molecular dynamics of n-alkanes. J.-P. Ryckaert, G. Ciccotti, H.J.C. Berendsen, J. Comput. Phys. 23 (1977) pp. 327-341.
5. Training -support vector classifiers: Theory and algorithms. C.-C. Chang, C.-J. Lin, Neural Computation, 2001.
6. VC-dimension for characterizing classifiers. Andrew W. Moore, 2001.
7. A tutorial on v-Support Vector Machines. Pai-Hsuen Chen, Chih-Jen Lin, Bernhard Scholkopt, 2001.
8. The Nature of Statistical Learning Theory. V. Vapnik, Springer-Verlag, 1995.

Раздел III
Имитационное моделирование

Волканов Д.Ю., Черей М.В.

Исследование применимости алгоритмов нечёткого поиска для анализа результатов имитационного моделирования ВС РВ

Введение

Имитационное моделирование применяется при проектировании и исследовании различных технических систем. При имитационном моделировании исследование изучаемого объекта проводится на имитационной модели (имитаторе объекта). Частным случаем имитационного моделирования является *дискретно-событийное имитационное моделирование* - моделирование системы в дискретные моменты времени, когда происходят события, отражающие последовательность изменения состояний системы во времени.

Одной из задач, решаемой при помощи дискретно-событийного имитационного моделирования, является задача разработки вычислительных систем реального времени (ВСРВ). Имитационная модель ВСРВ в зависимости от уровня детальности может использоваться для выбора структуры разрабатываемой системы, проверки количественных характеристик выполнения программ, проверки правильности работы циклограммы обмена и отладки программного обеспечения. При дискретно-событийном моделировании в качестве результата моделирования обычно выступает *трасса выполнения имитационной модели*. В трассу записываются события, отражающие последовательность изменения состояний системы во времени. Для проведения исследования моделей необходимо уметь анализировать полученную трассу. Одним из способов анализа трассы является поиск её фрагментов, отражающих поведение компонента близкое к эталонному поведению. В данной статье исследуется возможность поиска таких фрагментов трасс имитационных моделей при помощи алгоритмов нечёткого поиска. Под *нечётким поиском* понимают поиск объектов, похожих в некотором смысле на заданный эталон.

1. Трассы имитационных моделей и их представление

Рассмотрим трассы конкретной среды моделирования, на примере которых и будем проводить дальнейшие исследования.

Стенд Математического Моделирования Комплекса Бортового Оборудования [1] — это стенд моделирования, позволяющий производить эксперименты с моделями БСРВ в режиме полунатурного моделирования. *Полунатурное моделирование* — это способ моделирования, при котором часть компонентов системы представлена имитационными моделями, а часть — натурными компонентами. В результате эксперимента собирается трасса, содержащая информацию о событиях, произошедших в различных компонентах во время имитационного эксперимента. Внутри каждого компонента последовательность событий упорядочена по времени. Главной особенностью трасс моделей БСРВ является их большой размер. Например, для одной из моделей при моделировании 10 минут работы БСРВ собиралась трасса размером почти 2 гигабайта.

Событием трассы называется любое изменение состояния имитационной модели, существенное с точки зрения исследуемых свойств БСРВ, которое может произойти во время работы имитационной модели и которое отражается в трассе имитационной модели. Например, обмен данными по каналу МКИО представляется в трассе последовательностью событий передачи и ожидания командных и ответных слов и слов данных. Событие характеризуется набором атрибутов:

- типом — Т;
- подтипом — П;
- временем возникновения — В;
- набором дополнительных атрибутов — А.

Тип однозначно определяет принадлежность события тому или иному классу событий. Например, изменение параметров в модели задаётся типом Update. Подтип определяет группы событий внутри одного типа событий. Например, для типа Update различают два подтипа - обновление скалярного параметра и обновление параметра-массива. Существует *конечное множество* возможных типов и подтипов событий. Время возникновения - это модельное время возникновения события. Дополнительные атрибуты - некоторые дополнительные параметры события, которые необходимы для его описания. Дополнительные атрибуты могут отсутствовать. Таким образом, событие представляется четвёркой $C_i = \{T_i, П_i, B_i, A_i\}$.

Учитывая особенности некоторых атрибутов, рассмотрим способ представления событий. Каждому конкретному событию, основываясь на паре атрибутов тип и подтип, сопоставим символ некоторого алфавита. Так как возможные значения атрибутов «тип» и «подтип» — конечные множества, то такое отображение можно построить. Рассмотрим пример. Пусть имеется алфавит символов a-z и пусть есть события $C_1 = \{T_1, П_1, B_1, A_1\}$, $C_2 = \{T_2, П_2, B_2, A_2\}$, $C_3 = \{T_3, П_3, B_3, A_3\}$, отличающихся по атрибутам тип и подтип. Рассмотрим

события в виде пар $C_i = \{T_i, П_i\}$. Построим взаимнооднозначное отображение каждой пары $\{T_i, П_i\}$ в некоторый символ алфавита, то есть $\{T_1, П_1\} = a$, $\{T_2, П_2\} = b$, $\{T_3, П_3\} = c$. Тогда последовательность событий $C_1=\{T_1, П_1\}$, $C_2 = \{T_2, П_2\}$, $C_3 = \{T_3, П_3\}$, $C_1=\{T_1, П_1\}$, $C_1=\{T_1, П_1\}$ преобразуется в последовательность следующих символов алфавита: abcaa.

Таким образом, трасса может быть однозначно закодирована символами некоторого конечного алфавита. Стоит заметить, что конечность алфавита даёт возможность сравнения любых двух закодированных событий на равенство. Абстрагируясь от ряда атрибутов события, значительно уменьшается размер исследуемой трассы. Однако при этом набор проверяемых свойств ВСРВ ограничивается. Используя предложенный способ кодирования, можно проверять только те свойства ВСРВ, для которых важна лишь упорядоченность событий в одном из компонентов. Примером такого свойства может служить соответствие циклограммы обмена по аппаратному каналу, полученной экспериментально, — эталонной циклограмме обмена.

2. Задачи анализа трасс имитационных моделей

В настоящее время поиск в трассах осуществляется при помощи использования поиска по регулярным выражениям [2]. Данный механизм не всегда полностью удовлетворяет потребностям исследователя. К примеру, когда исследователь знает лишь некоторое число ключевых событий, и не знает порядок их следования, количество других событий между ними, то использование регулярных выражений может породить результаты, которые в дальнейшем потребуют дополнительной ручной обработки. Рассмотрим нечёткий поиск как способ анализа трасс имитационных моделей.

При помощи нечёткого поиска, можно решать следующие задачи:

- поиск последовательностей событий в трассе;
- сравнение двух фрагментов трасс.

Ответы, получаемые при решении данных задач, позволяют проводить анализ трассы имитационной модели.

Конкретизируем перечисленные выше задачи.

Задача 1. Поиск последовательности событий в трассе.

Объектом исследования является закодированная в некотором алфавитом трасса. Искомым объектом (шаблоном) является закодированная тем же алфавитом последовательность событий. Целью поиска является проверка наличия искомой последовательности или последовательности, похожей на искомую, в трассе. Здесь мера схожести – параметр, который задается исследователем. Практическим примером может служить проверка на правильность некоторой

последовательности событий, отвечающих определенному «составному» действию, например, обмену данными между компонентами модели ВСРВ по каналу МКИО.

Задача 2. Сравнение двух фрагментов трасс.

Здесь объектом исследования и шаблоном являются трассы одного размера, закодированные одним алфавитом. Целью является получение численного значения коэффициента схожести двух трасс. Примером может служить сравнение трассы работы модели с трассой эталонного поведения.

Рассмотрим задачи исследования трассы, для которых, используя данный способ кодирования, можно получить адекватный результат. Это множество тех задач, где важна последовательность событий, но не важно время их возникновения. Например, это уже упоминавшаяся задача анализа обменов данных по каналу МКИО. При анализе обменов по МКИО важно распознать какой из форматов обмена использовался. Формат эталонного обмена однозначно задаётся последовательностью событий трассы. В трассе эксперимента, как показывает практика, несмотря на требования стандарта по обменам данными по каналу МКИО, для некоторых из форматов обмена внутри последовательности событий обмена допустимы иные события. Поэтому для нахождения обмена по МКИО необходимо задать эталон обмена в виде строки, кодирующей эталонную последовательность событий и меру близости между строками, кодирующими обмены. Фрагменты реальной трассы, содержащей обмены по МКИО, сравниваются с эталонным фрагментом и вычисляется их мера близости к эталонному обмену.

С учётом особенностей трасс и поставленных задач, перейдём к вопросу выбора алгоритмов. Так как способ кодирования трасс, предложенный в пункте 2, фактически строит отображение трассы в строку символов некоторого алфавита, то рассмотрим существующие алгоритмы нечёткого поиска в текстовых строках.

3. Алгоритмы нечёткого поиска

Рассмотрим основные понятия связанные со строками. *Строка* — это одномерная последовательность символов некоторого конечного алфавита [3, 4]. *Участок строки — подпоследовательность* — символы исходной строки, стоящих на произвольных местах, но в том же порядке. Далее возникает вопрос в выборе функции расстояния. В литературе в основном используются следующие функции расстояния: расстояние Хэмминга, расстояние Левенштейна и расстояние редактирования [3, 4].

Не ограничивая общности, в реализациях алгоритмов используется расстояние Левенштейна. Можно использовать любую из трёх вышеперечисленных функций расстояния, стоит только учесть, что

полученные ответы будут требовать разной интерпретации. *Расстояние Левенштейна* — это минимальное по цене (сумма весов всех операций) преобразование первой строки во вторую в случае, когда разрешены операции вставки, удаления и замены с единичными весами. Таким образом, не делается различий по важности между удалением, вставкой или заменой события в трассе. Перейдем теперь к рассмотрению самих алгоритмов.

Существуют различные классификации алгоритмов нечёткого поиска[3, 4]. Можно разделить алгоритмы по применяемому в них методу, по временным затратам или затратам по памяти, по областям применения и так далее.

Предлагается классифицировать алгоритмы по выдаваемому результату. Тогда алгоритмы, относящиеся к разным классам, будут давать на выходе разные данные о трассах, что будет способствовать более детальному их анализу. Выделим следующие группы алгоритмов нечёткого поиска по выдаваемому результату:

- алгоритмы, дающие на выходе численное значение расстояния между строками
- алгоритмы, дающие на выходе длину наибольшей общей подпоследовательности (далее НОП) двух строк
- алгоритмы, дающие на выходе вектор позиций, начиная с которых участки строки и заданный участок имеют не более k различий.

Для проведения экспериментов были реализованы алгоритмы из каждой группы. Рассмотрим более подробно, какие задачи решают алгоритмы, относящиеся к каждой из этих групп.

Алгоритмы первой группы не дают никакой информации о расположении одной строки относительно другой, не позволяют определить общие участки двух строк. У них на выходе число — расстояние между двумя входными строками в рамках той метрики, которая была заложена в алгоритм. Эти алгоритмы применимы для сравнения входных данных одного размера (применительно к текстовым строкам, размер — длина строки), иначе интерпретация полученного результата становится затруднительной. Таким образом, они могут быть использованы для решения задачи 2, сформулированной в пункте 2.

Алгоритмы, относящиеся ко второй группе, позволяют найти как расстояние между эталоном и трассой, так и, непосредственно, расположение их наибольшей общей подпоследовательности. Здесь уже допустимы входные данные разных размеров, то есть эти алгоритмы применимы для решения задач 1 и 2 из пункта 2.

Алгоритмы из третьей группы позволяют получить вектор позиций исходной строки, начиная с которых, встречается искомая последовательность символов, имеющий не более чем k различий с

соответствующей частью исходной строки. Эти алгоритмы возможно применять для решения задачи 2, сформулированной в пункте 2.

Из набора алгоритмов нечёткого поиска в строках выберем те, которые наиболее подходят для решения задач анализа трасс имитационных моделей. Для этого, из множества характеристик, присущих этим алгоритмам (простота реализации, затраты по памяти и времени и так далее), выделим наиболее значимую. Такой характеристикой, в нашем случае, будет количество затрачиваемой памяти. Это связано в первую очередь с тем, что трассы большие по размеру. Вторым по значимости является критерий затрат по времени, так как результат поиска часто требуется получить в наиболее короткие сроки.

Руководствуясь указанному выше критерию и иными отмеченными ниже соображениями, для исследования было выбрано пять алгоритмов нечёткого поиска. Несколько слов о выборе каждого:

Алгоритм Вагнера-Фишера [5]: этот классический алгоритм не удовлетворяет выбранному выше критерию, но на его идеи опираются многие другие алгоритмы, дающие существенные улучшения по времени и памяти. Также он удобен для сравнения скорости работы и затрат на память.

Алгоритм Хиршберга [6]: этот алгоритм представляет собой оптимизированный по памяти алгоритм Вагнера-Фишера.

Алгоритм Накатсу [7]: алгоритм с пространственной сложностью O(nD), где D - это разность между длиной строки и длиной НОП двух строк, n - длина второй строки. Имеет хорошие показатели по времени работы и затратам по памяти.

Алгоритм Укконена [8]: улучшение алгоритма Вагнера-Фишера как по времени, так и по памяти.

Алгоритм Ландау-Вишкина [9]: этот алгоритм позволяет решить задачу поиска всех позиций в строке, начиная с которых искомый участок отличается от участков исходной строки не более чем на заданное значение. Имеет неплохие показатели по времени работы.

4. Результаты экспериментов

Для проверки работоспособности алгоритмов нечёткого поиска для решения задачи поиска в трассах было разработано программное средство с графическим интерфейсом. Программа написана на C++ с использованием библиотеки Qt. Все реализации алгоритмов описаны в программе в качестве методов соответствующего класса, что позволяет легко добавлять реализации новых алгоритмов.

Целью экспериментов является сравнение быстродействия выбранных алгоритмов при анализе трасс имитационного моделирования. Сравнение по объему занимаемой памяти можно провести, используя теоретические оценки затрат по памяти,

приведенные в описании алгоритмов. Эти оценки остаются справедливыми, так как задача поиска в трассах имитационного моделирования сводится к задаче поиска в текстовых строках, для которых получены соответствующие оценки.

Трассы для экспериментов были построены с использованием модели, описанной средствами СММ КБО.

Далее будем использовать следующие сокращения: В-Ф — алгоритм Вагнера-Фишера, Х — алгоритм Хиршберга, Н — алгоритм Накатсу, Л-В х – алгоритм Ландау-Вишкин(с параметром отклонения х), У – алгоритм Укконен. В описанных ниже сериях экспериментов применялись все алгоритмы, кроме серии экспериментов №2.

Ниже приведём описание серий проводимых экспериментов и их результаты. Необходимо отметить, что найденные фрагменты трассы совпадают с шаблоном с некоторым коэффициентом схожести. Порог для каждого эксперимента, начиная с которого найденный фрагмент считается искомым, определяется экспертом.

Серия экспериментов №1.

Часто целью поиска является не одиночное событие, а некоторая их последовательность. К примеру, исследователя может интересовать какой-то определённый режим работы модели или обмен двух моделей. В таких случаях следует искать упорядоченный набор событий. Целью экспериментов этой серии была проверка наличия последовательных событий (могут разделяться другими событиями), в трассе. В качестве входных данных, выбирались трасса с чередующимися определёнными событиями и искомый участок трассы (шаблон), состоящий из 4 событий, причём только два события шаблона совпадали с событиями трассы. Но в шаблоне они шли подряд, а в трассе между ними были другие события.

Как показали приведённые эксперименты, на основе результатов, выдаваемых алгоритмом Л-В, трудно сделать какие-либо выводы. В нашем примере, зная структуру трассы, это можно объяснить следующим образом: четыре события в точной последовательности, требуемой шаблоном, не встречаются в исследуемой трассе. Помсняв параметр отклонения от шаблона на 1 или 2 в Л-В, мы можем получить некоторые участки трассы, которые отличаются от заданных на 1 или 2 события, но подтвердить, что это искомые последовательности, может только эксперт.

Наилучшим образом в этих экспериментах показал себя алгоритм Хиршберга.

Серия экспериментов №2.

Целью экспериментов из этой серии также был поиск всех вхождений последовательности событий (могут разделяться другими событиями), в трассе. Но в отличие от предыдущих экспериментов, в тестовых трассах присутствуют 3 независимо повторяющихся события,

которые есть в определённом порядке в шаблоне. Для поставленной задачи из реализованных алгоритмов целесообразно применять алгоритм Л-В, так как он позволяет находить множественные вхождения шаблона с заданным значением числа отличий.

Данная серия экспериментов показала, что данный алгоритм чувствителен к параметру отклонения. При его применении требуется правильная подборка параметра и правильно составленный шаблон. В данных экспериментах шаблон сознательно составлялся так, что он отличался от подряд идущих событий в трассах для поиска. Если бы шаблон был точной частью трассы, которая последовательно повторяется, то нужный результат был бы получен при значении параметра отклонения 0. Тем не менее, правильная настройка алгоритма, позволяет найти нужные последовательности событий в трассах, хотя и с некоторой погрешностью.

Серия экспериментов №3.

Рассмотрим теперь случай, когда исследователь не знает структуру трассы, количество и порядок входящих в неё событий, но знает возможные типы событий. Возникает задача анализа этой трассы при помощи уже имеющихся данных, таких как ранее составленные шаблоны. К примеру, исследователь может пытаться выяснить, были ли в этой трассе интересующие его события и если были, то отличаются ли они как-нибудь от тех, что встречаются в трассе. Целью экспериментов этой серии было выявление как можно большей информации о вхождении шаблона в заданную трассу (позиция в трассе, степень отличия). Данная информация получалась путём экспертного анализа ответов одного из алгоритмов дающих длину НОП на выходе и алгоритма Л-В с различными параметрами отклонения.

Для экспериментов данной группы в качестве тестовых трасс для поиска выбирались трассы прогона одной из моделей, шаблонами – трассы из четырёх событий.

Заметим, что наиболее эффективным в данной ситуации оказалась пара алгоритмов Н и Л-В.

Серия экспериментов №4.

Целью экспериментов являлось сравнение двух трасс одинаковой длины. Эта задача актуальна при оценке правильности работы модели. Имея трассу правильного поведения, мы можем сказать, насколько любая другая трасса, описывающая работу модели, является близкой к эталону.

Как показали эксперименты, для решения данной задачи возможно применять любой из рассмотренных алгоритмов. Все они дают численный ответ, который определят схожесть трасс.

Общие выводы:

Проведённые эксперименты показали, что нечёткий поиск применим для решения задач, сформулированных в разделе 2.

Таблица 1 показывает алгоритмы, наиболее хорошо показавшие себя в определённых сериях экспериментов. На графиках 1 и 2 показано время работы использованных алгоритмов на разных наборах данных с одинаковой точностью ответов. Точки оси X показывают количество событий в исследуемой трассе и шаблоне, точки оси Y — время работы в миллисекундах.

По результатам экспериментов можно сделать следующие выводы:

- при применении алгоритма Ландау-Вишкина для нечёткого поиска в трассах важно подобрать правильное значение параметра отклонения, при котором будет достигаться наибольшая результативность по количеству и качеству найденных совпадений
- используя комбинации алгоритмов, возможен анализ трасс с более высокой точностью
- для получения меры близости двух трасс возможно применять любой из предложенных алгоритмов. Наиболее оптимальным оказались алгоритмы Накатсу и Укконена.

Таблица 1. Применимость алгоритмов для анализа трасс.

Алгоритм	В-Ф	Х	Н	Л-В0	Л-В1	Л-В2	У
Серия экспериментов №1	-	+	+	+	+	+	-
Серия экспериментов №2	+	+	+	+	-	-	+
Серия экспериментов №3	-	-	+	+	+	+	-
Серия экспериментов №4	+	+	+	+	-	-	+

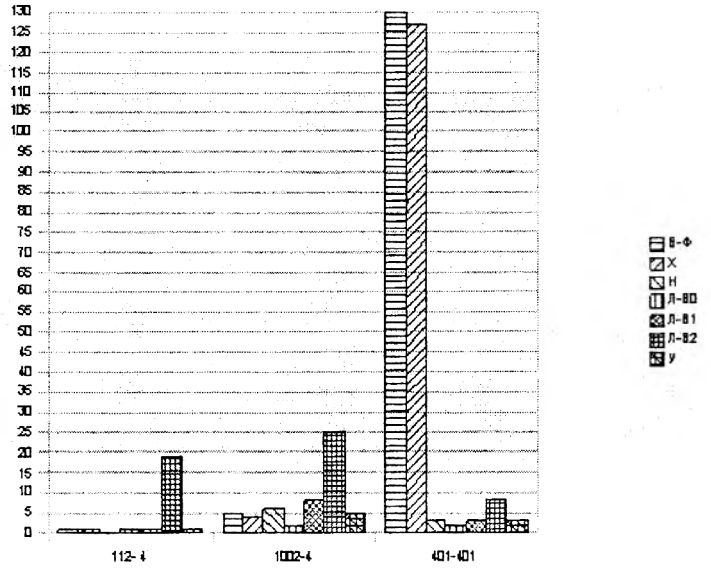

Рисунок 1. Время работы алгоритмов на малых объемах данных.

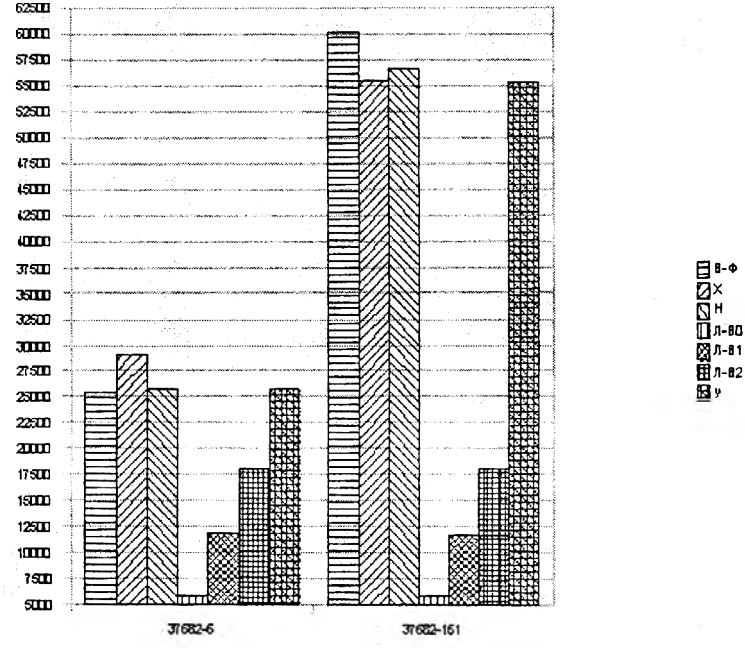

Рисунок 2. Время работы алгоритмов на больших объемах данных.

Заключение

Используемый способ представления трасс имитационных моделей позволяет использовать алгоритмы нечёткого поиска в текстовых строках для решения задач анализа трасс. Были рассмотрены существующие алгоритмы нечёткого поиска в текстовых строках. В соответствии с сформулированными критериями, выбрана часть алгоритмов, на основе которых реализовано программное средство анализа трасс имитационных моделей. С целью качественной и количественной оценки реализованных алгоритмов были проведены эксперименты.

Стоит заметить, что при осуществлении анализа трасс, не принимался во внимание такой важный атрибут событий как время, а также ряд дополнительных атрибутов, характерных для каждого из типов событий. В дальнейшем планируется ввести предварительную обработку трассы с учетом временных меток событий, основываясь на алгоритмах работы с временными рядами, а также расширить список кодируемых атрибутов.

Литература

1. Грибов Д.И., Смелянский Р.Л. Комплексное моделирование бортового оборудования летательного аппарата // Методы и средства обработки информации. Труды второй Всероссийской научной конференции. - М.: Издательский отдел ф-та ВМиК МГУ им. М.В. Ломоносова, 2005. - С.59-74.
2. Бочков С.О. Смелянский Р.Л. Отладка программ в распределённых вычислительных системах //Ж. Программирование 1988. №4.
3. Graham A.S. String Search // School of Electronic Engineering Science University College of North Wales, 1992.
4. Смит Б. Методы и алгоритмы вычислений на строках // Пер. с англ. М.: Издательский дом «Вильямс», 2006.
5. Wagner, R.A., Fischer, M.J. The String-to-string correction problem // Jour. ACM 21, 168-173, 1974.
6. Hierschberg, D.S. A linear space algorithm for computing maximal common subsequences // Commun. ACM 18, 133-152, 1975.
7. Nakatsu, N., Y. Kambayashi, S.Yajima A longest common subsequence algorithm suitable for similar text strings // Acta Informatika 18, 171-179, 1982.
8. Ukkonen, E. Finding approximate patterns in strings // Jour. Algorithms 6, 132-137, 1985.
9. Landau, G.M., U. Vishkin Fast string matching with k differences // Jour. Comp. and Susyem Sci. 37, 63-78, 1988.

Балаханов В.А., Костенко В.А.

Способы сведения задачи построения статико-динамического однопроцессорного расписания для систем реального времени к задаче нахождения на графе маршрута

Введение

Алгоритмы оптимизации, имитирующие кооперативное поведение муравьев в колонии, впервые были предложены в 1992 году [1] для решения задачи коммивояжера. Впоследствии муравьиные алгоритмы были успешно использованы для решения таких комбинаторных задач, как квадратичная задача о назначениях [2], задача упаковки в контейнеры [3], задачи построения расписаний [4,5,6].

Идея муравьиных алгоритмов основана на моделировании поведения муравьев при нахождении кратчайшего пути от муравейника к источнику пищи. Муравьи при перемещении оставляют особое вещество феромон, который используется в дальнейшем другими муравьями при выборе маршрута. Чем выше концентрация феромона на том или ином маршруте, тем выше вероятность того, что муравьи будут выбирать для движения именно этот маршрут.

Общую схему работы муравьиных алгоритмов можно представить в следующем виде:

1. Задание начального количества феромона на ребрах графа.
2. Определение количества и начального положения муравьев.
3. Поиск муравьями решений в соответствии с заданным алгоритмом построения маршрута.
4. Обновление количества феромона на ребрах в зависимости от качества полученных решений.
5. Если условие останова не выполнено, то переход к п.2.

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

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

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

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

В данной работе рассматриваются два способа сведения задачи построения статико-динамического однопроцессорного расписания для систем реального времени к задаче нахождения на графе маршрута.

1. Задача построения статико-динамического однопроцессорного расписания для систем реального времени

Статико-динамическое расписание представляет собой набор окон, каждое из которых характеризуется временем открытия, временем закрытия и списком работ, выполняющихся внутри окна. При этом порядок выполнения работ внутри окна определяется динамически и заранее неизвестен. Длина окна должна быть не меньше, чем суммарное время выполнения работ внутри него. Наряду с заданным списком прикладных задач в вычислительной системе могут выполняться системные задачи, список которых заранее неизвестен. На выполнение системных задач в каждом окне должен быть отведен определенный резерв времени.

Использование таких расписаний при работе систем реального времени позволяет уменьшить время реакции на прерывания и переключение режимов работы. Однако системы реального времени накладывают дополнительные ограничения на расписание. Кроме времени выполнения, каждая работа характеризуется также директивным интервалом, в пределах которого возможно ее выполнение. При этом временной интервал окна должен лежать внутри каждого из директивных интервалов работ, выполняющихся в данном окне, чтобы гарантировать, что директивные интервалы работ не будут нарушены.

Примером систем реального времени, использующих статико-динамические расписания, являются системы, работающие по стандарту ARINC-653 [7]. Одним из основных понятий стандарта является понятие раздела: в системе имеется определенный набор разделов, и

каждая работа характеризуется номером раздела. Работы из одного раздела могут выполняться непосредственно одна за другой, без задержек; для перехода из одного раздела в другой необходимо переключение контекста, которое требует определенного времени. Все работы внутри окна должны принадлежать одному разделу, таким образом, работы внутри окна могут выполняться непосредственно друг за другом, без временных затрат на переключение контекста, которое может производиться только между закрытием одного окна и открытием следующего. Таким образом, для каждой работы задано время выполнения, номер раздела и директивный временной интервал выполнения.

В данной работе будут рассматриваться системы, работающие в соответствии со стандартом ARINC-653. Приведем формальную постановку задачи построения расписания для таких систем:

Пусть задано множество работ:

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

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

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

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

Требуется построить расписание, представляющее собой набор окон:

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

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

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

При этом должны выполняться следующие ограничения:

1. $\forall(i,j)\in[1..m],i\neq j:SW_i\cap SW_j=\varnothing$ — множества работ, размещенных внутри окон, не пересекаются;
2. $\forall i\in[1..n],\forall j\in[1..m],a_i\in SW_j:s_i\le S_j<F_j\le f_i$ — временной интервал окна лежит внутри директивных интервалов работ, выполняющихся в окне;
3. $\forall(i,j)\in[1..n],\forall k\in[1..m],a_i,a_j\in SW_k:p_i=p_j$ — разделы работ, размещенных внутри одного окна, совпадают;
4. $\forall(i,j)\in[1..m],i<j:S_j\le F_i+\Delta 1$ — окна не пересекаются и между любыми двумя соседними окнами есть промежуток не короче времени, необходимого на переключение контекста;

5. $\forall j \in [1..m]: \sum_{a_i \in SW_j} t_i + \Delta 2 \le F_j - S_j$ — суммарное время выполнения всех работ из одного окна с учетом резерва времени не больше, чем длина окна.

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

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

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

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

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

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

1. Инициализация расписания: $Time=0$ – текущая длина расписания; $k=1$ – номер места размещаемой работы в SL; $SW^0=\varnothing$ – список работ в добавляемом окне;
2. Установка начальных параметров окна: $S=\max(Time,s_{ik})$, $F=f_{ik}$ – границы добавляемого окна; $T=\Delta 2$ – минимальная необходимая длина окна с учетом добавленных работ; $R=r_{ik}$ – раздел для работ в окне;
3. Обновление значений параметров окна с учетом новой добавляемой работы: $S'=\max(S,s_{ik})$, $F'=\min(F,f_{ik})$, $T'=T+t_{ik}$;
4. Добавление работы в текущее окно: если $r_{ik}=R$, $T' \le F'-S'$ (условия корректности не нарушаются), то: $S=S'$, $F=F'$, $T=T'$, $SW^0=SW^0 \cup \{a_{ik}\}$, $k=k+1$, если $k \le n$ (список SL еще не пройден), переход к п.3;
5. Проверка возможности добавления работы в новое окно: если $f_{ik}-F-\Delta 1<t_{ik}$, то $k=k+1$, если $k \le n$ переход к п.3 – данная работа не может быть размещена ни в текущее окно, ни в новое окно (дальше в

списке еще могут быть работы, которые можно разместить в текущее окно);

6. Закрытие окна: $F=S+T$ – устанавливаем время закрытия окна минимально возможным;
7. Добавление данного окна в расписание: $SP=SP\cup\{<S,F,SW^0>\}$;
8. Пересчет длины расписания: $Time=F+\Delta 1$, $k=k+1$;
9. Если $k \le n$ (список еще не пройден), переход к п.2.

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

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

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

Недостатком данного подхода является то, что множество корректных расписаний содержит больше элементов, чем множество последовательностей работ, т.е. существуют расписания, для которых не существует соответствующих им цепочек работ. Причина заключается в том, что последовательность работ не задает однозначно распределение работ по окнам. Более того, можно привести пример задач, для которых оптимальное расписание не будет принадлежать к пространству поиска муравьиного алгоритма. Пример исходных данных такой задачи: $SW=\{a_1=<0,11,4,1>,a_2=<4,11,2,1>,a_3=<4,11,2,1>\}$, $\Delta 1=\Delta 2=1$ (см. рисунок 1). Прямоугольниками показаны директивные интервалы, заштрихованными областями – время выполнения работ. Раздел у всех работ одинаковый.

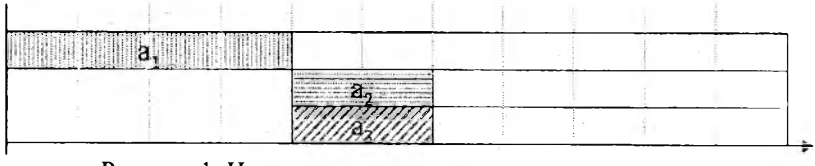

Рисунок 1. Исходные данные.

На рисунке 2 показаны все возможные расписания, которые могут быть построены приведенным алгоритмом по различным маршрутам в графе. Все эти расписания не являются оптимальными.

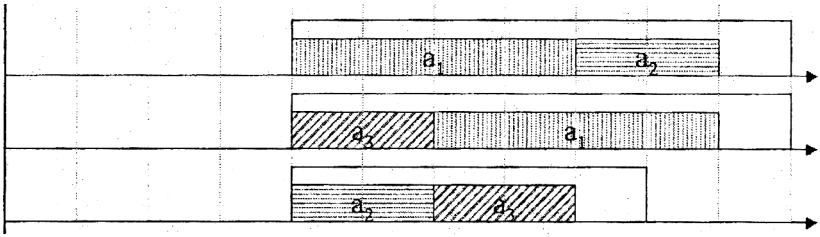

Рисунок 2. Построенные расписания.

При этом расписание, в котором все работы размещены, существует (рисунок 3).

Рисунок 3. Оптимальное расписание.

Можно выдвинуть гипотезу, что не существует полиномиального алгоритма A построения расписания по маршруту в графе G такого, что для $\forall(SW,\Delta 1,\Delta 2):\exists SL:A(SL)=SP^0$, где SP^0 – оптимальное расписание. Т.е. не существует полиномиального алгоритма, который для любой частной задачи мог бы построить оптимальное расписание по одному из возможных маршрутов. По крайне мере, пока такой алгоритм найти не удалось, однако нет и доказательства, что такого алгоритма не существует.

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

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

Построим граф $G'=<N',A'>$, где

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

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

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

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

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

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

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

двух). При этом в последовательности SL присутствует только одна вершина, соответствующая каждой работе.

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

1. *Базис индукции*: Т.к. все вершины, соответствующие работам из первого окна, стоят в последовательности вершин друг за другом, не прерываются вершиной с закрытием окна, то они будут размещаться в одно и то же окно. Т.к. исходное расписание является корректным, это возможно без нарушения условий корректности 1-5. Следовательно, списки работ в первом окне в SP и SP^* будут совпадать (поскольку список вершин, соответствующих работам из первого окна, заканчивается вершиной с закрытием окна, в данное окно не попадут «лишние» работы). Время открытия и закрытия первого окна могут различаться в SP^* и SP, но при этом из свойств (1) и (2) следует, что $S^*_1 \le S_1$, $F^*_1 \le F_1$.
2. *Шаг индукции*: $F^*_{i-1} \le F_{i-1}$, и, следовательно (из свойства (2)), $S^*_i \le S_i$, $F^*_i \le F_i$; далее повторяя предыдущие рассуждения, получаем, что корректность i-го окна в SP гарантирует корректность соответствующего окна в SP^* и совпадение списка работ, размещенных в них.

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

Теорема доказана.

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

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

Заключение

Способ сведения задачи построения статико-динамических расписаний к задаче нахождения маршрута на графе, в котором каждой работе соответствуют две вершины, увеличивает пространство поиска

муравьиного алгоритма. Однако, в отличие от способа, в котором каждой работе соответствует одна вершина графа, гарантирует принадлежность оптимального расписания к пространству поиска муравьиного алгоритма.

Литература

1. Dorigo M. Optimization, Learning and Natural Algorithms. // PhD Thesis. Dipartimento di Elettronica, Politechnico Di Milano, Milano. 1992.
2. Stuzle T., Dorigo M. ACO Algorithms for the Quadratic Assignment Problem. // New Ideas in Optimization, McGraw-Hills, 1999. P. 33-50.
3. Levine J., Ducatelle F. Ant Colony Optimization and Local Search for Bin Packing and Cutting Stock Problems. // Journal of the Operational Research Society, 2003.
4. Ritchie G. Static Multi-processor Scheduling with Ant Colony Optimization and Local Search. // Master's Thesis. University of Edinburgh, Edinburgh. 2003.
5. Blum C., Sampels M. Ant Colony Optimization for FOP Shop Scheduling: A case study on different pheromone representation // In Proceedings of the 2002 Congress on Evolutionary Computations, Honolulu. 2002.
6. Гафаров Е.Р. Гибридный алгоритм решения задачи минимизации суммарного запаздывания для одного прибора // Москва, ВЦ РАН. 2006.
7. Arinc Specification 653. Airlines Electronic Engineering Committee. [PDF] (http://www.arinc.com).

Бычков И.А., Костенко В.А.

Возможность использования арбитражного кольца Fibre Channel в вычислительных системах реального времени

Введение

Многие разработчики вычислительных систем реального времени рассматривают стандарт Fibre Channel как основной для построения среды обмена данными в вычислительных системах реального времени [1]. Стандарт Fibre Channel позволяет:

- обеспечить высокую пропускную способность (от 133 Мбит/с до 4 Гбит/с);
- обеспечить временные задержки при передаче сообщения не более 10 мкс;
- поддерживать широковещательные и групповые передачи сообщений;
- обеспечить высокую надежность (вероятность ошибки при передаче одного бита не превышает 10^{-12});
- обеспечить гарантированную доставку сообщения за гарантированное время.

Одной из поддерживаемых стандартом Fibre Channel топологий является арбитражное кольцо. В арбитражном кольце информация между устройствами передается по однонаправленному замкнутому контуру, и планирование обменов осуществляется динамически.

В вычислительных системах реального времени на обмены данными накладываются временные ограничения, нарушение которых приводит к потере работоспособности системы. При использовании арбитражного кольца в качестве среды обменов данными в вычислительных системах реального необходимо гарантированно обеспечить выполнение временных ограничений.

В данной работе рассматривается один из возможных подходов к построению среды обменов вычислительных систем реального времени на основе арбитражного кольца Fibre Channel, гарантированно обеспечивающий корректность выполнения обменов данными в режиме реального времени.

1. Краткое описание арбитражного кольца Fibre Channel

Арбитражное кольцо – это топология, в которой информация между устройствами передается по однонаправленному замкнутому контуру [2]. Стандарт Fibre Channel поддерживает различные физические среды передачи данных [3], в том числе коаксиальный

кабель, витую пару, оптоволоконный кабель. В данной работе предполагается, что физическая среда передачи данных – оптоволоконный кабель.

Для подключения к арбитражному кольцу устройство должно содержать в своем составе FC-AL адаптер (порт) [4], который содержит передатчик и приемник, т.е. каждый порт имеет один входной канал связи и один выходной канал. Каналы в арбитражном кольце организованы в однонаправленный замкнутый контур.

Каждое сообщение, которое должно быть передано от источника приемнику, передается внутри кадра данных. Кадр – это единица передачи данных, содержащая помимо передаваемых слов данных, также информацию об устройстве-приемнике данного кадра, информацию, позволяющую осуществлять контроль ошибок, и другую служебную информацию. Информация передается блоками по 4 байта, каждый такой блок называется словом [5], каждое слово кодируется 40 битами.

В стандарте Fibre Channel предусмотрено несколько классов обслуживания. Топология арбитражного кольца поддерживает [3, 6, 7] только следующие классы обслуживания:

- класс 1 – выделенное соединение с подтверждениями;
- класс 2 – передача без установки соединения с подтверждениями;
- класс 3 – передача без установки соединения без подтверждений.

Класс 3 предоставляет возможность передачи данных сразу нескольким выбранным приемникам или сразу всем устройствам в кольце. В дальнейшем в данной работе предполагается, что обмены данными осуществляются в 3-м классе обслуживания.

Перед началом работы арбитражное кольцо проходит инициализацию, в процессе которой происходит настройка параметров кольца, получение портами адресов и др. Несмотря на то, что процесс инициализации занимает значительное время, в данной работе предполагается, что к моменту начала планируемого информационного обмена процесс инициализации уже был успешно завершен.

Работа порта арбитражного кольца происходит следующим образом. Поступающие во входной канал слова попадают во входной буфер. Слова, передаваемые в текущий момент времени в выходной канал, находятся в выходном буфере. Размер выходного буфера стандартом не определен, поэтому в данной работе будем считать, его равным 40 бит. Это означает, что при поступлении слова в выходной буфер, первый бит сразу же, без задержки начинает передаваться в выходной канал. После того как слово из выходного буфера было передано в выходной канал, порт декодирует очередной набор из 40 бит из входного буфера [8]. Результатом декодирования может быть либо одно из управляющих слов, либо четырехбайтовое число.

В каждый момент времени каждый порт в арбитражном кольце находится в одном из нескольких возможных состояний. Состояние порта при передаче очередного слова определяется результатом декодирования предыдущего принятого слова и состоянием порта в момент его декодирования.

Опишем основные состояния порта.

MONITORING. В этом состоянии все принятые портом слова ретранслируются далее, т.е. порт передает в выходной канал принятый набор из 40 бит. Порт переходит в это состояние, когда он не имеет сообщений для передачи.

ARBITRATING. Порт переходит в это состояние, когда ему необходимо получить доступ к кольцу для передачи информации другому (или другим) портам, т.е. в этом состоянии порт пытается получить подтверждение от всех других портов, что они не претендуют на доступ к каналу и готовы ретранслировать по кольцу слова, передаваемые портом-источником. В этом состоянии порт ретранслирует кадры данных (если они передаются) без изменений, но при этом между кадрами он отправляет специальное служебное слово ARB, с указанием своего сетевого адреса. Если другие порты с меньшим адресом не претендуют на доступ к кольцу, они ретранслируют это служебное слово дальше до тех пор, пока оно не вернется к исходному порту. При этом порт перейдет в состояние ARBITRATION WON.

ARBITRATION WON. Состояние, в котором порт считается выигравшим арбитраж. В этом состоянии порт должен открыть один или несколько других портов для передачи им данных с помощью служебного слова OPN. При этом порт переходит в состояние OPEN.

OPENED. Порт-приемник переходит в это состояние, когда он получает слово OPN с указанием своего адреса, т.е. когда порт-источник пытается открыть порт-приемник.

OPEN. Состояние, в которое переходит порт-источник после открытия порта-приемника (после отправки слова OPN). В этом состоянии порт начинает передавать кадр с данными. Однако перед и после отправки кадра порт должен передать не менее шести служебных слов-заполнителей. При этом количество слов данных в кадре не должно превышать 528.

XMITTED CLOSE. Порт переходит в это состояние, когда у него больше нет данных для передачи, и для закрытия портов-приемников он передал служебное слово CLS. В этом состоянии порт ожидает полного обхода словом кольца (при этом он передает слова-заполнители), после чего он переходит в состояние MONITORING.

RECEIVED CLOSE. Порт-приемник переходит в это состояние, когда он получает служебное слово CLS. В этом состоянии порт

считается закрытым. В этом состоянии порт ретранслирует слово CLS и переходит в состояние MONITORING.

В нашей модели будем считать, что в начале каждого такта каждый порт имеет во входном буфере полностью принятое 40-битное слово. В течении текущего такта порт декодирует полученное слово в набор байтов (или служебное слово), обрабатывает его и помещает в свой выходной буфер слово, предназначенное для передачи в выходной канал. В течение текущего такта слово из выходного буфера передается в выходной канал.

Пусть арбитражное кольцо состоит из P устройств. Каждое устройство в кольце имеет свой уникальный адрес. Будем считать, что для арбитражного кольца из P устройств адрес каждого из устройств лежит в интервале [1, P]. Каждое устройство подключено к одному порту. Номера портов арбитражного кольца также будем нумеровать от 1 до P.

В арбитражном кольце устройства могут располагаться в разном порядке. Топологией арбитражного кольца будем называть последовательность адресов устройств в данном кольце (начиная от первого порта в направлении передачи данных). Топология задает отображение (определяет привязку) устройства к порту арбитражного кольца. Топологию кольца можно задать как перестановку q длины P. i-й элемент перестановки q (будем обозначать символом $q(i)$ или q_i) равен адресу устройства, подключенного к i-му порту. Таким образом, последовательность номеров портов в кольце в порядке следования один за другим всегда будет иметь вид 1, 2, 3, ... , P. Последовательность же адресов устройств, соответствующая этим портам, будет иметь вид q_1, q_2, ..., q_P. То есть за устройством с адресом q_1 следует устройство с адресом q_2, и т.д., а за устройством с адресом q_P следует устройство с адресом q_1.

Топологию арбитражного кольца можно задать и другим способом. Рассмотрим последовательность p_1, p_2, ..., p_P номеров портов, подсоединенных к устройствам с адресами 1, 2, ..., P. Здесь элемент p_k (другое обозначение $port(k)$) равен номеру порта k-го устройства. Такая последовательность задает отображение адресов устройств на номера портов, подсоединенных к этим устройствам. Очевидно, что $port(q_i) = i$ и $q(port(k)) = k$. Последовательность p тоже является перестановкой, причем обратной к перестановке q, т.е. $p = q^{-1}$.

Количество тактов, необходимых для передачи слова от i-го порта следующему за ним порту, будем считать известным и обозначать символом L_i. Величина L_i складывается из времени, в течении которого сигнал распространяется в физической среде передачи данных, и времени, в течении которого сигнал находится во входном буфере порта.

Будем называть расстоянием $L(i,j)$ от i-го до j-го порта количество тактов, необходимых для передачи слова от i-го порта до j-го порта. Аналогично, расстоянием от k-го устройства до n-го устройства будем называть количество тактов, необходимых для передачи слова от k-го устройства до n-го. Очевидно, это расстояние равно $L(port(k), port(n))$. Величину $L(i,j)$ можно вычислить по формуле

$$L(i,j) = \begin{cases} \sum_{k=i}^{j-1} L_k & , если\ j > i \\ Z - \sum_{k=j}^{i-1} L_k & , если\ j < i \\ Z & , если\ j = i \end{cases}$$

Здесь $Z = \sum_{i=1}^{P} L_i$ – время обхода (количество тактов, необходимых для обхода) всего кольца.

Период времени, за который устройство – источник сообщения отправляет в канал информацию, необходимую для передачи i-го сообщения можно разделить на три этапа, начинающихся последовательно друг за другом:

Первый этап – передача устройством-источником в канал информации, которая может быть передана еще до того, как известны слова данных подлежащего передаче сообщения. К первому этапу относится арбитраж (Z тактов), установка соединения с приемниками (OPN) и передача шести слов-заполнителей. Общее время, затрачиваемое источником на выполнение первого этапа при передаче i-го сообщения составляет $v_i^1 = Z + K + 6$ тактов. Здесь K – число приемников сообщения. При этом, если используется широковещательная связь, т.е. если приемниками сообщения являются сразу все устройства кольца, то K полагается равным 1, поскольку нет

необходимости открывать каждое из устройств в отдельности. Можно открыть сразу все устройства кольца.

Второй этап – передача устройством-источником кадра данных, содержащего сообщение. Ко второму этапу относится: передача семи служебных слов, передача слов данных от источника к приемнику, передача двух служебных слов. Общее время, затрачиваемое источником на выполнение второго этапа при передаче i-го сообщения составляет $v_i^2 = v_i + 9$ тактов (здесь v_i – количество слов данных).

Третий этап – передача слов, необходимых для корректного завершения сеанса связи. К третьему этапу относятся: передача шести слов-заполнителей, отправка служебного слова CLS и ожидание полного обхода его по кольцу. Общее время, затрачиваемое источником на выполнение третьего этапа при передаче i-го сообщения составляет $v_i^3 = Z + 6$ тактов.

Введем также величину Δ_i (время передачи), которая равна количеству тактов, необходимых для того, чтобы слово от источника достигло приемника:

$$\Delta_i = \max_{d \in DST_i} \left(L(port(src_i), port(d)) \right) - 1$$

где DST_i – множество приемников сообщения. Слова, передаваемые источником на втором этапе, будут полностью приняты приемником (или всеми приемниками, если их несколько) сообщения через Δ_i тактов после окончания второго этапа.

Необходимо подчеркнуть, что третий этап передачи сообщения может пересекаться во времени с началом передачи другого сообщения, точнее с его первым этапом. Это возможно за счет того, что если на третьем этапе передачи сообщения некоторое промежуточное устройство передало служебное слово CLS, то это устройство уже становится свободным, поскольку оно больше не задействовано в передачи данного сообщения, и поэтому может уже начинать процедуру арбитража с целью передачи очередного сообщения. При этом устройство-передатчик ожидает полного обхода всего кольца служебного слова CLS до окончания третьего этапа.

2. Задача построения расписания обменов в арбитражном кольце Fibre Channel

Каждое сообщение m_i, подлежащее передаче по арбитражному кольцу, характеризуется следующими величинами:

- v_i – количество слов данных в сообщении. Полагаем, что для корректных сообщений $v_i \in [1; 528]$, т.е. v_i не превышает максимального количества слов, которое может быть передано в одном кадре данных.
- src_i – адрес устройства, передающего сообщение.
- DST_i – множество адресов устройств, принимающих сообщение.
- $[s_i, f_i]$ – директивный интервал, где s_i – это момент времени, в который содержание сообщения стало известно устройству источнику, f_i – время до которого приемник должен завершить прием кадра данных, содержащего сообщение.

Расписанием будем называть множество сообщений m_i, которым помимо исходных величин, описанных выше, приписываются также следующие величины: s_i', v_i^1, v_i^2, Δ_i, v_i^3.

Величина s_i' — время старта сообщения (определяет момент начала первого этапа передачи сообщения m_i), величины v_i^1, v_i^2 и v_i^3 — длительности выполнения первого, второго и третьего этапов передачи соответственно, а величина Δ_i — время передачи.

Таким образом, для каждого сообщения интервалы времени, в течение которых продолжаются соответствующие этапы сообщения будут следующими:

- Первый этап продолжается в течение времени $[s_i'; s_i' + v_i^1)$.
- Второй этап продолжается в течение времени $[s_i' + v_i^1; s_i' + v_i^1 + v_i^2)$.
- Информация, переданная передатчиком на втором этапе полностью принята приемником на f_i^*-м такте, $f_i^* = s_i' + v_i^1 + v_i^2 + \Delta_i$.
- Третий этап продолжается в течение времени $[s_i' + v_i^1 + v_i^2; s_i' + v_i^1 + v_i^2 + v_i^3)$.

Общая продолжительность V_i передачи сообщения m_i составляет $v_i^1 + v_i^2 + v_i^3$ такта.

Интервал времени $[s_i^* = s_i' + v_i^1, f_i^*]$ будем называть *интервалом размещения* сообщения m_i. В течение этого интервала времени осуществляется передача слов данных сообщения.

Определим предикат $part(k,T,i)$, который для каждого устройства с адресом k, каждого такта шины T и для каждого сообщения m_i показывает, участвует ли k-е устройство на такте T в передаче сообщения m_i. k-е устройство участвует в передаче сообщения m_i на такте T, если оно обязано на такте T ретранслировать в выходной канал слово из входного канала (либо если k-е устройство является передатчиком – осуществлять передачу очередного слова m_i-го сообщения).

Предикат $part(k,T,i)$ можно вычислить по следующей формуле:

$$part(k,T,i) = true \Leftrightarrow$$

$$\begin{cases} T \in [s_i'; s_i' + V_i) & \text{, если } k = src_i \\ T \in [s_i'; s_i' + V_i) \wedge T - s_i' - (V_i - Z) \leq \\ \qquad \leq L(port(src), port(k)) \leq T - s_i' & \text{, если } k \neq src_i \end{cases}$$

На итоговое (построенное) расписание накладываются следующие ограничения корректности:

- интервал размещения каждого сообщения m_i должен находиться в пределах соответствующего директивного интервала, т.е. должно выполняться следующее условие:
$$(s_i \leq s_i^*) \wedge (f_i^* \leq f_i);$$
- в каждый момент времени T для каждого устройства k должно существовать не более одного сообщения m_i такого, что $part(k,T,i) = true$.

При таких условиях корректности гарантированно выполняются все временные ограничения на передачу сообщений и корректно выполняются все обмены.

Задача построения расписания обменов в арбитражном кольце ставится следующим образом. Исходными данными являются:

- P – число устройств в арбитражном кольце;
- $\{L_i\}$ – длины каналов между портами;
- топология арбитражного кольца q;

- $M = \{ m_i \},\ i = \overline{1,N}$ – набор исходных сообщений.

Требуется найти корректное расписание, содержащее наибольшее число сообщений.

3. Алгоритм построения расписания обменов

Предлагаемый алгоритм построения расписаний относится к классу гибридных алгоритмов сочетающих жадные стратегии и стратегии ограниченного перебора. При описании алгоритма, под выполнением работы будем понимать передачу сообщения.

Введем следующие величины:

- $MaxRbkLev \in [0, N-1]$ – максимальная глубина перебора (является параметром алгоритма). Если $MaxRbkLev = m$, то ограниченным перебором будут проверяться $(m+1)!$ вариантов;
- $UList$ – список неразмещенных работ, в начале работы алгоритма содержит в произвольном порядке все работы, подлежащие выполнению;
- $SList$ – список размещенных работ (в начале пустой);
- $RbkLev$ – текущая глубина перебора (в начале равна нулю). Откат (удаление работы из списка $SList$) соответствует увеличению текущей глубины перебора $RbkLev$ на единицу. Размещение работы в список $SList$ соответствует уменьшению текущей глубины перебора, но не более чем до нуля;
- $NextTask$ – работа-кандидат на размещение в расписание.

Каждую работу также будем характеризовать величиной $fail_i$ — количеством случаев, в которых данная работа не поместилось в свой директивный интервал. В начале работы алгоритма эти величины равны нулю.

Отметим, что, не смотря на то, что величина s_i' характеризует размещенную работу, будем приписывать эту величину также и неразмещенным работам из списка $UList$. В этом случае величина s_i' будет означать время, в которое начнется работа m_i, если её разместить в расписание (т.е. переместить в список $SList$).

Введем основные операции, используемые в алгоритме.

- Операция $Calc(SList, UList)$ – пересчитывает величины s_i' для всех работ из списка $UList$ при заданном списке $SList$ размещенных работ.

- Операция $Check(UList)$ – проверяет, помещается ли каждая работа из списка $UList$ в свой директивный интервал. Для выполнения этой проверки достаточно проверить, что для всех работ из $UList$ выполнено неравенство $f_i^* \le f_i$. Если все работы помещаются в свои директивные интервалы, то операция возвращает $true$, в противном случае – $false$. В процессе работы операция увеличивает на единицу величину $fail_i$ для каждого сообщения, которое не помещается в свой директивный интервал.
- Операция выбора работы ограниченным перебором $Rollback(SList, UList, RbkLev, NextTask)$. Осуществляет откат к предыдущему состоянию и выбор работы $NextTask$ таким образом, что размещение этой работы в расписание (если оно возможно) будет очередным вариантом ограниченного перебора. Операция возвращает $true$, если откат успешно осуществлен и выбрана работа $NextTask$. Операция возвращает $false$, если все варианты ограниченного перебора были исчерпаны и очередная работа $NextTask$ не выбрана.
- Операция $Del(UList)$ удаления работы из списка $UList$ – удаляет из списка $UList$ работу с наибольшей величиной $fail_i$ (если таких работ несколько, то среди них выбирается работа с наименьшей величиной $\left(f_i - s_i - v_i^2\right)$).
- Операция $First(UList, NextTask)$ – выбирает из списка $UList$ работу $NextTask$ с наименьшей величиной f_i^*. Если таких работ несколько, то среди них выбирается работа с наименьшим номером.
- Операция $Next(UList, LastTask, NextTask)$ – выбирает из списка $UList$ работу $NextTask$, следующую по величине f_i^* за работой $LastTask$ (переданной в качестве параметра операции). Если такая работа существует, то операция возвращает $true$, в противном случае операция возвращает $false$.

Ниже приведено описание предложенного алгоритма построения расписания обменов:

1. Вычислить величины v_i^1, v_i^2, Δ_i, v_i^3 для каждой работы из списка неразмещенных работ $UList$.
2. Если список неразмещенных работ $UList$ пуст, то останов алгоритма. Список $SList$ содержит построенное расписание работ.

3. Выполнить операцию $Calc(SList, UList)$ пересчета величин s_i' работ из списка $UList$.

   Если операция $Calc$ вернула $true$, то выполнить операцию $First(UList, NextTask)$ выбора из списка $UList$ работы $NextTask$ с наименьшей величиной f_i^*.

   Если же операция $Calc$ вернула $false$, то:

   a. Выполнить операцию $Rollback(SList, UList, RbkLev, NextTask)$ выбора из списка $UList$ работы $NextTask$ ограниченным перебором.

   b. Если операция $Rollback$ вернула $false$, то выполнить операцию $Del(UList)$ удаления работы из списка $UList$ и перейти к п.2.

4. Разместить работу $NextTask$ в $SList$. Удалить работу $NextTask$ из $UList$.
5. Если $RbkLev > 0$, то уменьшить $RbkLev$ на единицу.
6. Перейти к п.2.

Операция $Rollback(SList, UList, RbkLev, NextTask)$ выбора работы ограниченным перебором:

1. Если список $SList$ пустой, то окончить выполнение текущей операции $Rollback$ с результатом $false$.
2. Переместить последнюю работу $LastTask$ из конца списка $SList$ в начало списка $UList$ и увеличить $RbkLev$ на единицу.
3. Выполнить операцию $Calc(SList, UList)$ пересчета величин s_i' работ из списка $UList$.
4. Выполнить операцию $Next(UList, LastTask, NextTask)$ выбора из списка $UList$ работы $NextTask$, следующей по величине f_i^* за работой $LastTask$.
5. Если операция $Next$ вернула $true$, то окончить выполнение текущей операции $Rollback$ с результатом $true$.

   Если же операция $Rollback$ вернула $false$, то:

   a. Если $RbkLev \geq MaxRbkLev$ (превышена глубина перебора), то окончить выполнение текущей операции $Rollback$ с результатом $false$.

b. Если же $RbkLev < MaxRbkLev$ (глубина перебора не превышена), то рекурсивно вызвать операцию выбора работы ограниченным перебором $Rollback(SList, UList, RbkLev, NextTask)$ и окончить выполнение текущей операции $Rollback$ с результатом рекурсивного вызова.

При выполнении операции $Calc(SList, UList)$, т.е. при пересчете величины s_i' для всех работ из списка $UList$ эти величины должны быть выбраны таким образом, чтобы размещение работы m_i в расписание, начиная с такта s_i', не нарушало бы корректности расписания. Этому условию соответствует выбор величины s_i' по формуле: $s_i' = \max(GetFirstTime(m_i, SList), s_i - v_i^1)$. Согласно этой формуле за время старта s_i' в зависимости от того, какая величина больше, выбирается либо величина $(s_i - v_i^1)$, либо величина $GetFirstTime(m_i, SList)$.

Величина $(s_i - v_i^1)$ будет выбрана в случае, когда время старта s_i' ограничивается снизу временем s_i. Поскольку первый этап передачи сообщения не обязан помещаться в директивный интервал, то работа m_i в таком случае может начаться за v_i^1 тактов до времени s_i.

Величина $GetFirstTime(m_i, SList)$ будет выбрана в случае, когда время старта s_i' ограничивается снизу последним сообщением из $SList$. Если порт-источник последнего сообщения m_{last} из $SList$ совпадает с портом-источником очередного сообщения m_i, то это означает, что первый такт шины, на котором такой порт может начать передачу очередного сообщения равен $s_{last}' + V_{last}$, т.к. порт-отправитель должен дождаться полного обхода по кольцу служебного слова CLS. Если же порт-источник последнего сообщения не совпадает с портом-источником очередного сообщения, то передача очередного сообщения может начаться раньше — спустя $(L(port(src_{last}), port(src_i)) + 1)$ такт после передачи портом-источником служебного слова CLS, которое передается на $(s_{last}' + V_{last} - Z)$-м такте:

$$GetFirstTime(m_i, SList) =$$

$$= \begin{cases} 0 & , \text{если } |SList| = 0 \\ s'_i + V_i & , \text{если } src_{last} = src_i \\ \left(s'_{last} + V_{last} - Z + 1 + \right. & \\ \left. + L(port(src_{last}), port(src_i)) \right) & , \text{в остальных случаях} \end{cases}$$

Алгоритм, описанный выше, корректно строит искомое расписание. Однако время работы такого алгоритма можно существенно уменьшить, если в процессе работы отсекать заведомо бесперспективные ветви поиска. Для этого операции $Calc$, $Check$, $First$ и $Next$ должны обрабатывать не все работы из списка $UList$, а только те, которые при размещении в расписание в дальнейшем не приведут к заведомому откату. Один из возможных вариантов реализации этих операций заключается в следующем.

Выполнение каждой из указанных выше операций начинается с того, что величина $MinFin$ выбирается равной f_{first} (здесь m_{first} – первая работа списка $UList$). Работы обрабатываются в порядке их расположения в списке. Перед обработкой очередной работы m_i проверяется, меньше ли f_i чем $MinFin$, и если меньше, то величина $MinFin$ выбирается равной f_i. Далее, работа m_i обрабатывается только, если величина s_i окажется меньше, чем $MinFin$. В противном случае такая работа не обрабатывается, т.к. в списке неразмещенных работ заведомо существует работа, которая должна располагаться в расписании раньше работы m_i.

Здесь под обработкой понимается действие, выполняемое той или иной операцией (для операции $Calc$ – это расчет величины s'_i, для операции $Check$ – проверка возможности размещения работы m_i в расписание (с возможным увеличением величины $fail_i$), для операции $First$ – нахождение минимума величины f_i^* и текущего минимума, для операции $Next$ – нахождение минимума величины f_i^* и текущего минимума, но только для работ m_i, для которых $(f_i^* > f^*_{LastTask}) \vee (f_i^* = f^*_{LastTask} \wedge i > LastTask)$).

Указанный выше способ сокращения времени выполнения алгоритма работает корректно, т.к. выбран один и тот же порядок обработки работ, поэтому те работы, которые не были пересчитаны операцией $Calc$, заведомо не будут рассматриваться и другими операциями. Эффективность указанного выше способа очень велика — время работы алгоритма уменьшается на несколько порядков.

Получим верхнюю оценку вычислительной сложности алгоритма. Пусть число сообщений в исходном наборе равно N, $MaxRbkLev = m$, $m \in [0, N-1]$ (при $m = N-1$ алгоритм будет осуществлять полный перебор). Вычислительная сложность операции $Rollback$ равна $O(N * m)$ (ввиду наличия в ней рекурсии с глубиной не большей, чем m). Вычислительные сложности всех остальных операций, определенных выше, равны $O(N)$. При размещении или удалении очередной работы алгоритм осуществляет ограниченный перебор не более, чем $(m+1)!$ вариантов размещения работ. В худшем случае выбор очередного варианта осуществляется с помощью выполнение операции $Rollback$, которая выполняется не более, чем за $O(m * N)$ операций. После перебора $(m+1)!$ вариантов в худшем случае осуществляется удаление (за $O(N)$ операций) одной работы из списка $UList$. Число шагов алгоритма равно $N-1$. Общее число операций алгоритма $O(m * N^2 * (m+1)!)$.

Приведем результаты численных исследований алгоритма на модельных данных. Случайным образом многократно строился заведомо совместимый набор входных данных с заданными характеристиками, после чего запускался алгоритм построения расписания. При этом фиксировалось качество построенного расписания (A – число работ, размещенных в расписание) и RBK — число запусков операции $Rollback$ выбора работы ограниченным перебором (именно эта операция оказывает основное влияние на время выполнения алгоритма).

Характеристиками входного набора данных являются:

- Эффективность использования канала (Eff) – отношение суммы общих продолжительностей (включая первый и третий этапы) всех работ, размещенных в расписание, к продолжительности всего расписания. Эффективность использования канала может превышать 100%, поскольку информационные обмены могут накладываться друг на друга по времени.
- Дисперсия ($Disp$) – среднее отношение продолжительности директивного интервала к интервалу размещения сообщения.

- Количество устройств в кольце было взято равным 20.
- Количество сообщений в исходном наборе было взято равным 1000.
- Максимальная глубина перебора была взята равной 2.

В Таблице 1 приведена статистическая информация по 100 запускам алгоритма.

Таблица 1. Статистическая информация по 100 запускам алгоритма.

Eff	*Disp*	A min	A avg	A max	RBK min	RBK avg	RBK max
120 %	10,0	999	999,97	1000	1	10,58	34
120 %	7,0	999	999,90	1000	6	23,67	62
120 %	5,0	996	998,98	1000	18	48,27	107
120 %	3,0	992	997,91	1000	15	37,56	73
120 %	1,1	1000	1000,0	1000	0	0	0
100 %	10,0	1000	1000,0	1000	0	0,52	4
100 %	7,0	1000	1000,0	1000	0	1,74	9
100 %	5,0	1000	1000,0	1000	0	3,34	11
100 %	3,0	999	999,96	1000	0	2,27	7
100 %	1,1	1000	1000,0	1000	0	0	0
75 %	10,0	1000	1000,0	1000	0	0,01	1
75 %	7,0	1000	1000,0	1000	0	0,07	1
75 %	5,0	1000	1000,0	1000	0	0,29	4
75 %	3,0	1000	1000,0	1000	0	0,29	3
75 %	1,1	1000	1000,0	1000	0	0	0

Как видно из таблицы, продолжительность работы алгоритма несколько возрастает при превышении эффективности использования канала 100%. При $Disp \approx 5$ продолжительность работы алгоритма имеет максимальное значение. Это объясняется тем, что при величине $Disp$ близкой к единице интервалы размещения работ заданы очень жестко – они почти совпадают с директивными интервалами. При очень большой величине $Disp$ работы, имеют много возможных мест размещения – их директивные интервалы во много раз превышают интервалы размещения. Наибольшую трудность представляет как раз промежуточный случай.

Отметим, что на персональном компьютере AMD Athlon XP, 2,30 ГГц время построения расписания для наиболее сложного случая (107 операций выбора работы ограниченным перебором) составляет менее 0,6 сек.

Заключение

В данной работе был предложен способ организации обменов в арбитражном кольце Fibre Channel, гарантированно обеспечивающий корректность выполнения всех обменов и выполнение всех временных ограничений, накладываемых на информационные обмены.

В работе рассмотрен вариант алгоритма построения расписаний обменов, в которых допустим только 3-й класс обслуживания. Однако, предложенный алгоритм допускает настройку на построение расписаний обменов, в которых также допустимы 1-й и 2-й классы обслуживания.

Литература

1. Павлов А.М. Коммерческая сетевая информационная технология для применения в военных проектах // Мир компьютерной автоматизации. 2000. № 4. (http://www.mka.ru/?p=40031).
2. Fibre Channel Arbitlated Loop (FC-AL) // working draft proposal American National Standard for Information Technology, 1995. 98p. (http://www.t11.org/ftp/t11/member/fc/al/fcal45rl.pdf).
3. Fibre Channel Physical and Signalling Interface - 2 (FC-PH-2) // working draft proposed American National Standard for Information Systems, 1996. 161p. (http://www.t11.org/ftp/t11/member/fc/ph-2/fc-ph-2_74.pdf).
4. Михайлов А., Коротченко С., Пащенков П. Топология и оборудование // Журнал iXBT.com. 1999. (http://www.ixbt.com/storage/fibre_3.html).
5. 8b/10b encoding// WikiPedia. The Free Encyclopedia. (http://en.wikipedia.org/wiki/8B10B).
6. Fibre Channel Physical and Signalling Interface - 3 (FC-PH-3) // working draft proposed American National Standard for Information Systems, 1997. 112 p. (http://www.t11.org/ftp/t11/member/fc/ph-3/fcph3_94.pdf).
7. Fibre Channel Arbitrated Loop - 2 (FC-AL-2) // NCITS working draft proposal American National Standard for Information Technology, 2001. 140 p. (http://www.t11.org/ftp/t11/pub/fc/fs/02-038v0.pdf).
8. Fibre Channel Physical and Signaling Interface (FC-PH) // working draft proposed American National Standard for Information Systems, 1994. 388 p. (http://www.t11.org/ftp/t11/member/fc/ph/fcph_43.pdf).

Раздел IV
Инструментальные средства и сообщения

Королев Л.Н, Попов А.М., Попова Н.Н., Зленко П.А., Мещеряков Д.К., Певцов С.Е., Позднеев А.В., Попова Е.А., Сальников А.Н., Федулова И.А.

Развитие информационной системы, направленное на задачи идентификации биомолекулярных структур с использованием нейросетей и генетических алгоритмов[1]

Одной из важных фундаментальных задач современной прикладной математики и информатики является задача анализа плохо структурированных экспериментальных данных, полученных по разным методикам, с целью извлечения из них новых знаний, выявления новых законов, характеризующих реальные явления и процессы.

При обработке таких данных, как правило, очень больших объемов, сильно зашумленных, с потерей в них значительной части информации, не всегда удается применить мощные средства современной классической математики.

Исследование и разработка новых методов анализа таких данных в разных предметных областях, создание инструментальных средств, позволяющих автоматизировать вычислительные эксперименты по проверке работоспособности предложенных методов и алгоритмов, и упрощающих их использование при решении конкретных задач, являются важными шагами в направлении решения этой общей фундаментальной задачи.

На факультете вычислительной математики и кибернетики МГУ при поддержке РФФИ с 2002 года ведутся интенсивные исследовательские и прикладные работы в направлении разработки информационной системы обработки подобного рода экспериментальных данных [1]. Особое внимание при этом было уделено исследованиям вновь появившихся эвристических методов типа эволюционных и генетических алгоритмов, генетическому программированию, связанным с общей идеей «Машинного Обучения (Machine Learning)».

[1] Работа выполнена при поддержке грантов РФФИ 99-07-90229, 02-07-90130, 05-07-90238.

Разработка проблемно-ориентированных информационных систем, построенных с использованием распределенных вычислительных ресурсов, является актуальным направлением развития современного программного обеспечения. Целью создания таких систем является предоставление пользователю – эксперту в конкретных научных областях, широкого спектра вычислительных услуг по интеллектуальной обработке данных, скрывая от пользователя проблемы, связанные с организацией сбора, хранения и обновления данных, особенностей построения современных вычислительных систем.

Такая система была разработана и ее опытная эксплуатация началась с 2000 г.

Система представляет собой распределенный программный комплекс, предоставляющий пользователям унифицированный способ запуска на различных вычислительных машинах встроенных методов обработки данных, реализованных в виде специально оформленных вычислительных модулей. Особое внимание было уделено организации нейросетевой обработки данных, использованию параллельных вычислений, возможности подключения к Системе различных свободно распространяемых пакетов программ, в том числе, пакетов визуализации и графических систем.

На рисунке 1 представлена архитектура Системы на уровне основных функциональных компонент.

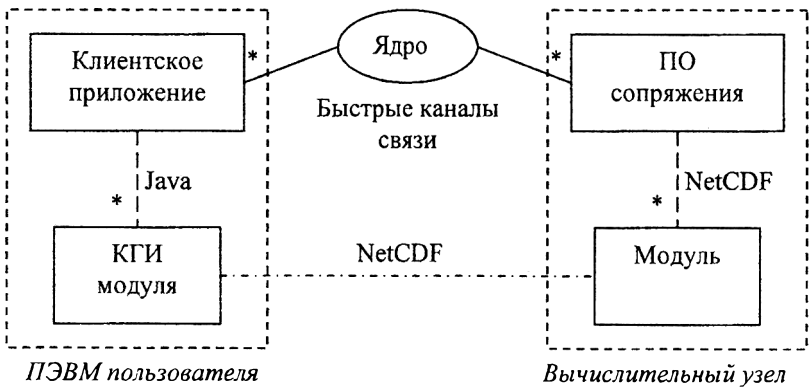

Рисунок 1. Архитектура информационной системы обработки экспериментальных данных.

Здесь * — множество однотипных объектов, пунктир — связь между компонентами приложения внутри машины, штрих-пунктир — опосредованная связь, КГИ — компоненты графического интерфейса.

Важной проблемой, решаемой в рамках гранта на основе применения построенной информационной системы, является проблема автоматизированного анализа и идентификации сложных биомолекулярных соединений на основе данных масс-спектроскопии и электроэнцефалографии. Разработанная информационная система использовалась для проведения исследований и решения ряда актуальных задач биоинформатики и биомедицины.

В ходе исследований по этому направлению были разработаны новые методы и алгоритмы для идентификации структурных элементов сложных биомолекулярных соединений по экспериментальным данным с учетом нечеткости, зашумленности и частичной потери в экспериментальных измерениях. Алгоритмы опробованы на решении задачи идентификации протеинов по аминокислотной последовательности, восстановленной из масс-спектра. Разработанный метод протестирован на большом числе реальных масс-спектров. Результаты тестирования по количеству правильно идентифицированных последовательностей превысили известные алгоритмы (SPIDER, CIDentify) [2-4]. Был разработан новый трехмерный численный код частиц в ячейке (Particle-in-Cell Code) для моделирования поведения ионных облаков в ловушке масс-спектрометра в реальных условиях эксперимента. Это позволило эффективно распараллелить предложенный метод обработки, который был реализован на вычислительной системе IBM 690 pSeries, установленной на факультете ВМК МГУ, и на супер-ЭВМ SOLO - кластере из 40 систем IBM 690 pSeries, установленной в институте молекулярной физики AMOLF в Нидерландах.

Проведены исследования по разработке новых подходов и алгоритмов для интеллектуального анализа экспериментальных данных, связанных с измерением электрической активности головного мозга. Исследован и реализован метод решения обратной задачи ЭЭГ, позволяющий определять изменяющиеся во времени зоны активности нейронных источников. Метод основан на Генетическом Алгоритме. Показано, что в отличие от ранее предложенных в мире методов данный метод позволяет определять как несколько активных зон, так и появление новых зон электрической активности в процессе временного развития. Показано, что в отличие от имеющихся алгоритмов предложенный метод позволяет локализовать глубинные источники электрической активности мозга по ЭЭГ сигналу. Разработанный метод использован для решения практической задачи о выделении признаков специфической ментальной активности -- запоминания информации [5]. Метод успешно внедрен в НИИ нормальной физиологии им. В.К. Анохина. Разработан оригинальный алгоритм анализа сигналов ЭЭГ на основе метода случайных деревьев (Random Forest метод),

превзошедший по своей эффективности наиболее развитые зарубежные методы [6].

В рамках общей проблемы поиска закономерностей в символьных последовательностях исследована задача определения структуры символьной последовательности, кодирующей ДНК человека с точки зрения временной сложности. В ходе выполнения проекта был разработан метод построения приближённого решения задачи поиска структуры последовательности как набора зависимостей. Важной особенностью предлагаемого метода является независимость временной сложности предлагаемого метода от длины анализируемой последовательности, что делает его более эффективным для анализа последовательностей большой длины, чем существующие аналогичные алгоритмы. Разработанный метод был применены к экспериментальному исследованию последовательности, представляющей собой кодировку Y-хромосомы человека. Проведенные исследования алгоритма подтвердили правильность применяемого подхода. Результаты его работы представлены к дальнейшему изучению специалистами в области биоинженерии.

В рамках исследований, проводимых совместно с Институтом физико-химической биологии им. А.Н. Белозерского МГУ и Людвиговского института раковых исследований "Ludwig Institute for Cancer Research", были разработаны алгоритмы определения повторов – слабо отличающихся последовательностей нуклеотидов в человеческом геноме. Была реализована параллельная реализация алгоритма MUSCLE. В многопроцессорном варианте на двенадцати процессорах задача решается в 2,4 раза быстрее, чем последовательный вариант [7].

Результаты работ, выполненных по гранту, неоднократно докладывались на конференциях, в том числе и международных. Исполнителями гранта за последние 3 года опубликовано более 30 работ. В работах по гранту принимают участие студенты и аспиранты. В 2007 году исполнителями гранта защищено 3 кандидатские диссертации, 3 дипломные работы.

Основные публикации

1. В.Ю.Воронов, Горицкая В.Ю., Зленко П,А., Истомин Т.Е., Колпаков Р.В., Королев Л.Н., Мещеряков Д.К., Попова Н.Н., Сальников А.Н. Специализированная распределенная система обработки экспериментальных данных. //Методы и средства обработки информации. Труды второй Всероссийской научной конференции./ Под ред.Л.Н.Королева - М.: Издательский отдел факультета вычислительной математики и кибернетики МГУ, 2005, с.213-219.

2. Pevtsov S., Fedulova I., Mirzaei H., Buck C., Zhang X. Performance evaluation of existing de novo sequencing algorithms // J.Proteome Research –2006- Vol.5.-N11.-P.3018-3028.
3. Певцов С.Е. Исследование эффективности существующих алгоритмов идентификации пептидов //Масс-спектрометрия.- 2006. – Т.3. - №4.- С.95-105.
4. Федулова И.А. Робастный алгоритм идентификации протеинов в базе данных // Вестник Московского Университета, серия 15 "Вычислительная математика и кибернетика" /Издательство МГУ, 2007.
5. Попов А.М. Решение обратной задачи электроэнцефалографии с помощью стохастических методов распознавания образов// Вестник Московского университета. Серия 15. Вычислительная математика и кибернетика. №3б, 2006, стр. 91-100.
6. Попова Е.А. Метод ансамблей деревьев решений для анализа электрической активности мозга. // Вестник Московского Университета, серия 15 "Вычислительная математика и кибернетика" / Издательство МГУ, 2007.
7. Salnikov A.N. PARUS: A Parallel Programming Framework for Heterogeneous Multiprocessor Systems //Lecture Notes in Computer Science. Recent Advantages in Parallel Virtual Machine and Message Passing Interface / Springer-Verlag, 2006, c. 408-409.

Королев Л. Н.

О некоторых проблемах компьютерной реализации алгоритмов кластеризации и классификации[1]

Введение

Широкий круг задач, которые решаются с применением компьютеров, сводятся к классификации и кластеризации [1] элементов некоторых множеств, в которых тем или иным способом закодированы свойства реальных объектов и особенности предметной области исследований.

Существенно то, что в качестве исходных данных для решения задач классификации выступают выборки ограниченного объема кодов элементов, представляющих реально существующие классы объектов. Выборки, принадлежащие тому или иному классу, как правило, формируются экспертами на основе опыта и интуиции, и могут содержать неточности и ошибки, возникающие из принятой методики представления данных и способов кодирования.

При обработке на компьютерах данных любого происхождения мы имеем дело с множествами, состоящими из элементов, представляющих слова над конечным алфавитом. Эти слова можно интерпретировать как числа, графы, текст и т. п. В дальнейшем изложении, говоря о множествах, мы чаще всего будем подразумевать множества двоичных слов — в конечном итоге компьютер может обрабатывать элементы только таких множеств.

В задачах кластеризации требуется группировать схожие между собой элементы множества в условиях, когда формально не определена метрика, оценивающая взаимные отношения между реальными объектами, поэтому в определении степени похожести часто приходится основываться на интуиции и проверке гипотез, соответствующих здравому смыслу.

Формализация понятия «здравого смысла» сводится к выработке ряда необходимых условий, которым должны удовлетворять результаты, и к доказательствам, что полученные решения дают требуемые результаты.

Основная проблема при решении задач классификации и кластеризации состоит в создании алгоритмов, удовлетворяющих этим плохо формализуемым условиям.

---

[1] Работа выполнена при поддержке гранта РФФИ № 05-07-90238.

Две постановки задачи кластеризации

Возможны две постановки задачи кластеризации.

Первая связана с определением ландшафта плотности распределения объектов в некотором пространстве.

Это означает, что дано пространство, в котором расположены интересующие нас объекты с заданными характеристиками, и перед нами стоит задача, во-первых, обнаружить их, во-вторых, определить области скоплений таких объектов.

Примерами первой постановки задачи кластеризации могут служить задачи обнаружения звездных скоплений, выделения областей повышенного и пониженного давления, задачи из области биологии, химии, физики, экономики, социологии, где требуется определить области сосредоточения объектов, обладающих заданными свойствами.

Вторая постановка состоит в том, что в множестве, состоящем только из объектов и не содержащем никаких других элементов, следует отыскать группы «схожих» объектов.

Вторая постановка сводится к анализу заданного множества объектов с целью определения структуры этого множества, например, определения частоты встречаемости «похожих» элементов, разделения объектов на группы «похожих», определения рельефа упорядоченной последовательности объектов.

Примером постановки задачи кластеризации второго типа может служить кластеризация текстов для информационного поиска материалов, объединенных в соответствии с запросами.

Учитывая возможности и ограничения современных компьютеров, на которых нам придется выполнять алгоритмы классификации и кластеризации, мы будем пытаться строить формальные математические постановки задач классификации и кластеризации.

Начнём с формальной постановки задачи классификации.

Пусть некоторое множество объектов M разбито на N непересекающихся подмножеств $K_1, K_2, K_3 \ldots K_N$, которые мы назовем классами.

Пусть явно заданы методом перечисления (списками) непересекающиеся конечные подмножества $E_i \subset K_i \subset M$ $(i=1,N)$, которые мы назовем выборками представителей классов $K_1, K_2, K_3 \ldots K_N$. При этом:

$$E_i \subset K_i \subset M \ (i=1,N) \ u \ K_i \cup K_j = \varnothing \ (i \neq j). \qquad (*)$$

Будем считать, что каждый объект обладает набором присущих ему характеристик и, что все семантические характеристики рассматриваемых объектов любой предметной области заданы двоичными наборами ограниченной длины, и число различных характеристик, приписываемых объектам, также конечно.

Иными словами, мы будем иметь дело с конечными множествами, элементы которых суть вектора со значениями координат, представимых в машинном (двоичном) коде. Эти вектора мы будем называть векторами характеристик соответствующих им объектов.

Заметим следующее, каждая координата вектора характеристик, обладает предписанной ей семантикой. Диапазон возможных значений, представляющих эту семантику, у каждой координаты свой.

Таким образом, с формальной точки зрения мы имеем дело с множеством векторов, часть из которых является признаками некоторых реальных объектов предметной области исследований, а часть этого общего пространства может не принадлежать к векторам-характеристикам. Однако, если не оговорено особо, мы будем считать, что все эти вектора являются векторами-характеристиками объектов. Обозначим это конечное множество той же буквой V.

Требуется найти такие алгоритмы (функции), которые определяли бы для любого элемента e множества V принадлежность этого элемента одному и только одному классу. Идентификатор класса, которому принадлежит элемент множества, будем считать целым числом. Задача сводится к построению функции, которая отображает множество V на множество целых чисел - номеров классов, на которые разбиты элементы исходного множества. При этом должно соблюдаться условие, что все элементы выборок должны оставаться в своих классах.

Если не налагать никаких других условий на отношения элементов, принадлежащих множествам классов K_1, K_2, K_3, ... K_N, кроме условий

$$E_i \subset K_i \subset V \ (i=1,N) \text{ и } K_i \cup K_j = \varnothing \ (i \neq j) \qquad (*)$$

и условий сохранения выборок в своих классах, то такая постановка не имеет практического смысла, т.к. существует множество алгоритмов, решающих такую задачу.

Беря за основу любое произвольное деление множества на классы, мы можем построить алгоритм, который решает поставленную задачу. Алгоритм определения принадлежности произвольного вектора v к одному из классов K_i, заданных своей выборкой E_i, прост и состоит в следующем:

- Проверим, не совпадает ли наш вектор v с одним из векторов, принадлежащих выборкам E_i.
- Если наш вектор совпал с одним из векторов, принадлежащих выборке E_i, мы полагаем, что этот вектор принадлежит классу E_i.
- Если вектор не совпал ни с одним из векторов выборок, то отнесем волевым порядком наш вектор к одному из классов.

Формально при таком алгоритме будут соблюдены требования (*) исходной задачи. В то же время, очевидно, что такой алгоритм,

основанный на произвольном выборе принадлежности к классу-сборщику «неопознанных» объектов, практически не пригоден, хотя формально он решает поставленную задачу.

При решении практических задач классификации и кластеризации явно или неявно предполагается, что существует некоторый критерий «похожести» друг на друга элементов, принадлежащих одному классу. В метрических пространствах в качестве такого критерия можно считать расстояние. Но во многих практически важных задачах приходится иметь дело с пространствами, в которых метрика не введена, или даже с пространствами, в которых не возможно ввести приемлемую метрику, отвечающую интуитивному понятию «похожести».

Задача классификации приобретает смысл, если найдено подобие метрики, которое все вектора, принадлежащие одному классу, делает близкими по «похожести», делает классы в некотором смысле «компактными».

Заметим, что любой конструктивный алгоритм, умеющий распределять элементы множества на непересекающиеся классы, можно считать способом формального введения понятия «похожести», или «подобия», или «близости» элементов. Так понимаемая «похожесть», не удовлетворяет классическим аксиомам расстояния, например, неравенство треугольника в этом случае, как правило, не работает.

В конкретных задачах множество V представляет собой векторы признаков некоторых реальных объектов, которые мы хотим классифицировать, опираясь на интуитивные или экспертные представления о близости одних объектов к другим.

В задачах автоматической классификации и кластеризации считается, что алгоритмы формирования выборок нам неизвестны. Именно в таких условиях необходимо решать задачи. В качестве начальных данных мы можем использовать только сами выборки, на основе анализа которых должны конструироваться гипотезы структуры классов и алгоритмы их построения.

Формальный анализ выборок

Рассмотрим простой пример, демонстрирующий возможность построения алгоритма классификации на основе анализа выборок.

Пусть исходное множество V представляет собой множество двоичных слов (векторов) заданной длины n =6. Пусть нам заданы две выборки слов, принадлежащих двум классам A и B:

$$E_A \subset A \subset V$$

$$010111, 111001, 101101, 010001 \text{ и}$$

$$E_B \subset B \subset V.$$

$$111100, 100010, 010110, 101010.$$

Анализ содержимого этих двух выборок показывает, что все элементы выборки A являются нечетными двоичными числами, а элементы выборки B — четными. Это, в свою очередь, позволяет нам высказать интуитивно обоснованное предположение, что весь класс A состоит только из нечетных чисел, а весь класс B — только из четных. Алгоритм искомого отображения, если верно наше предположение, очевиден: проверяется младший разряд слова, если он равен 1, то слово относится к классу A, в противном случае — к классу B. Это позволяет построить булеву функцию, разделяющую классы. Она совершенно проста $F(x_1, x_2, x_3, x_4, x_5, x_6) = x_6$.

Степень похожести двух векторов (слов) также определяется по их младшим разрядам: если они одинаковы, то элементы похожи, если различны, то непохожи. Это мало соответствует тому, что мы понимаем под метрикой в обычном классическом ее понимании, но, тем не менее, любым двум векторам может быть поставлено в соответствие число 0 или 1, определяющее их близость. Формально нашу догадку, о том простом алгоритме, который позволяет классифицировать все элементы данного множества на два класса, можно интерпретировать как некоторый метод анализа выборок для конструктивного построения алгоритма.

Рассмотрим другой пример выборок, принадлежащих двум классам множества 6-разрядных двоичных слов:

$$E_A \subset A \subset V$$

$$010101, 111011, 101110, 010101 \text{ и}$$

$$E_B \subset B \subset V.$$

$$111100, 100010, 010010, 101001.$$

Эти выборки составлены таким образом, что в выборке A в трех младших разрядах всегда только по две двоичных 1, а в выборке B — в трех младших разрядах всегда по одной двоичной 1.

Это позволяет нам регулярным образом строить логические функции, различающие классы. В этом примере они таковы:

$$\varphi_1(x_1, x_2, x_3, x_4, x_5, x_6) = \overline{x}_4 \overline{x}_5 x_6 \vee \overline{x}_4 x_5 \overline{x}_6 \vee x_4 \overline{x}_5 \overline{x}_6,$$

$$\varphi_2(x_1, x_2, x_3, x_4, x_5, x_6) = \overline{x}_4 x_5 x_6 \vee x_4 \overline{x}_5 x_6 \vee x_4 x_5 \overline{x}_6.$$

В этом примере переменными, по которым определяется принадлежность к классам, оказались три младших разряда. Поиск сочетания значимых переменных, по которым можно построить функции, определяющие принадлежность к классам, можно осуществлять направленным перебором. Расчеты показывают, что прямой перебор для реальных задач неприемлем. Отыскание таких функций — одна из задач дискретной математики.

Анализ содержания выборок, относящихся к различным классам, состоит в оценке статистики, в анализе частоты встречаемости значений компонент (координат) векторов, включенных в выборки.

В частности, в нашем первом простом примере мы обнаружили, что частота встречаемости 1 в последнем разряде выборки E_A равна 1, а частота встречаемости 1 в последнем разряде выборки E_B равна 0. Тем самым были обнаружены два взаимоисключающих признака, однозначно характеризующих эти классы, и тем самым алгоритм классификации был однозначно определен.

Эти простые примеры подсказывают один из возможных путей поиска определяющих признаков принадлежности объекта к одному из классов, на основе формального анализа состава выборок. Эту задачу мы будем рассматривать применительно к исходному множеству, состоящему из двоичных векторов.

Пусть задано множество V двоичных слов одинаковой длины n. Очевидно, что такое множество слов конечно и его мощность равна 2^n. Рассмотрим некоторое подмножество $K \subset V$ — класс K.

Пусть $E_K \subset K$ — некоторая выборка элементов, принадлежащих классу K.

Введем понятие *матрицы* Q_{EK} размерности $2 \times n$ *частотных характеристик выборки* E_K, которая представляет собой частоты появления букв двоичного алфавита 0 и 1 в каждой j-той позиции n-разрядного слова.

$$Q_{EK} = \begin{pmatrix} a_{01} a_{11} \\ a_{02} a_{12} \\ \cdot \\ a_{0n} a_{1n} \end{pmatrix}$$

Элемент a_{ij} обозначает частоту встречаемости i-той буквы нашего алфавита на j-той позиции слова, принадлежащего выборке E_K.

Любое слово из V длины n с помощью такой матрицы можно отобразить в некоторое векторное пространство V_{EK} размерности n, состоящее из векторов размерности n, координаты которых состоят из чисел, обозначающих частоты появления соответствующих букв в столбцах матрицы Q_{EK}. Множество V_{EK} векторов, полученное таким образом, мы назовем множеством, индуцированным выборкой E_{EK}.

Можно взять другую выборку $E_B \subset E$ из того же класса $E \subset V$, и индуцировать с помощью соответствующей матрицы Q_{EB} множество V_{EB}. Вообще говоря, множества, индуцируемые разными выборками из одного и того же класса, будут различны.

Однако если класс построен по какой-то логике, т. е. существует функции или алгоритмы, генерирующие рассматриваемый класс, то правомочна гипотеза, утверждающая что V_{EK} и V_{EB} совпадают. Косвенным признаком существования функции, различающей класс, может служить близость индуцируемых выборками характеристических матриц или «компактность» объединения множеств $V_{EA} \cup V_{EB}$. Здесь слово *компактность* взято в кавычки, т. к. определение понятие компактности может зависеть от предметной области.

Отображение $E_K \rightarrow V_{EK}$ не является взаимно однозначным. Если, например, все a_{ij} одинаковы, то все слова множества V отобразятся только на один вектор пространства образов V_{EK}.

Если частота появления какой-либо буквы на какой-либо позиции слова равна 1, то данная буква может служить характеристикой, определяющей класс K. Правомочна гипотеза, утверждающая, что наличие данной буквы в данной позиции любого слова однозначно определяет принадлежность его к классу K. Вероятность того, что такая ситуация в выборке E_K возникла чисто случайно, даже при небольших объемах выборки, очень мала.

Анализ частоты встречаемости букв алфавита на различных позициях слова согласуется с идеей поиска логических функций, разделяющих на классы.

В каждом столбце можно найти букву алфавита, частота появления которой максимальна, можно так же вычислить среднюю частоту встречаемости буквы на заданной позиции. По означенным частотам можно построить слова, которые мы назовем типичными характеристическими словами класса K или центром класса K по заданной выборке E_K.

Очевидно следующее *утверждение*. Если в каждом из столбцов все частоты a_{ij} различны, то отображение, полученное с использованием такой матрицы, взаимнооднозначно.

Переход от букв к векторам позволяет нам вводить понятие расстояния между двумя словами $\rho(s_1, s_2)$, пользуясь аппаратом классической векторной алгебры в евклидовом пространстве, например, так:

$$\rho(s_1,s_2) = \rho(v_1,v_2) = \sqrt{\sum_{1}^{n} (x_{i1} - x_{i2})^2},$$

где x_{i1}, x_{i2} — соответствующие элементы векторов v_1, v_2.

Можно также ввести в рассмотрение понятие расстояния слова от подмножества K, как минимальное расстояние до центра подмножества.

Анализ выборок в задачах классификации, связанный с поиском логической функции, разделяющей исходное множество на

классы, требует выполнения огромного перебора с применением методов минимизации булевых функций.

Задачи кластеризации, связанные с поиском в исходном множестве групп «похожих», близких друг к другу элементов, также требует огромного перебора вариантов группирования, соответствующих интуитивному понятию близости.

Анализ одного простого алгоритма классификации

Пусть на отрезке $[a, b] \subset X$ заданы два конечных непересекающихся множества точек $x_{1i} \in [a, b]$, $i = 1..к_1$ и $x_{2j} \in [a, b]$. $j = 1..к_2$.

Пусть точки x_{1i} принадлежат некоторому подмножеству [a, b], которое мы назовем классом $К_1$, а точки x_{2j} принадлежат другому подмножеству $К_2 \subset [a, b]$. При этом $К_1 \cup К_2 = \varnothing$ и $К_1 \cup К_2 = [a, b]$. Без дополнительных условий, как было отмечено, постановка задачи конструирования функции F(x), разделяющей эти подмножества-классы и определенной на всем отрезке [a, b], не имеет смысла, т.к. всегда можно построить алгоритм, причисляющий к одному классу только точки x_{1i}, а ко второму классу все другие точки, включая точки x_{2j}, или, наоборот, отнести все точки x_{2j} к одному классу $К_2$, а все остальные, включая x_{1i}, к классу $К_1$.

Дополнительное условие, которому должна удовлетворять функция F(x), разделяющая множество [a, b] на два класса, интуитивно состоит в том, что точки более близко расположенные к точкам соответствующего класса должны причисляться именно к этому классу. Это интуитивно понятное условие требует формализации.

Рассмотрим один из простейших алгоритмов классификации, удовлетворяющий условию, в котором «близость» точки *x* к тому или иному классу определяются однозначно. Этот простейший алгоритм построения F(x) состоит в том, что будем искать разделяющую функцию, обладающую обязательным свойством в точках $x_{1i} \in К_1$ принимать отрицательные значения ($F(x_{1i}) < 0$), в точках $x_{2j} \in К_2$ — положительные значения ($F(x_{2j}) \geq 0$), а во всех других точках F(x) должна принимать значения, определяемые близостью к точкам выборки из соответствующего класса. Рассмотрим некоторый алгоритм конструктивного построения целого класса таких функций, удовлетворяющих перечисленным выше требованиям. Он достаточно прост и состоит в следующем. Будем искать требуемую функцию в виде полинома, степень которого определится в ходе ее конструирования.

Отрезок определения нашей функции [a, b] мы разметим следующим образом: в каждом интервале, на границах которого точки выборок принадлежат разным классам, произвольно выберем единственную точку, в которой искомая функция-полином будет

принимать значение 0. соответственно степень полинома определится числом чередований точек, принадлежащих разным выборкам.

Всегда можно будет построить полином соответствующей размерности, одинарными корнями которого будут именно эти точки перемен знака искомого полинома.

Пусть ω_k — точки из интервалов, на границах которых расположены точки выборки, принадлежащие разным классам. Тогда функция

$$F(x) = \prod_{k=1}^{m} (x - \omega_k)$$

будет полиномом, отвечающим нашим требованиям. Во всех точках первого класса она будет положительной, во всех точках второго класса она будет отрицательной, и в любой другой точке x интервала ее знак будет определяться, в зависимости от ее расположения относительно нуля функции на рассматриваемом интервале.

В зависимости от того, как выбраны точки ω_k нулей полинома на интервалах перемены знака функции, мы будем получать функции с различными характеристиками.

Можно, например, потребовать от полинома, чтобы он был максимально «гладкий» — добиваться минимизации константы Липшица. Можно потребовать, чтобы достигался максимум в определенном интервале, например, в интервале наибольшей плотности точек одного из классов. Возможно и появление самых экзотических требований к полиному, зависящих от предметной области исследований.

Замечание 1. Вышеприведенный алгоритм позволяет определять кластеры в выборках представителей классов. Мы назовем точки ω_k точками перехода. Если общее число точек перехода меньше суммы точек выборок из классов K_1 и K_2, то это означает наличие кластеров с числом элементов в них не менее 2. Центры этих кластеров можно определить, вычислив точки, в которых первая производная нашего полинома обращается в 0, вычисляя корни уравнения

$$\sum_{i=1}^{m} \frac{\prod_{k=1}^{m} (x - \omega_k)}{(x - \omega_i)} = 0.$$

На рисунке 1 приведен вид полинома: крестиками помечены точки выборки, принадлежащие K_1, кружками — точки K_2.

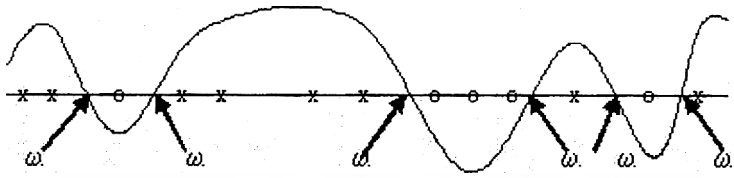

Рисунок 1.

Замечание 2. В приведенном выше алгоритме построения функции, разделяющей классы, мы рассматривали одномерное пространство, в некоторых точках которого располагались элементы выборок. При этом негласно предполагалось, что метрика в нем Евклидова. Способ построения полинома по нулям в заданных точках естественным образом может быть перенесен на дискретный случай, когда выборки заданы в точках на одномерной решетке.

При обработке на компьютерах пространства, с которыми в большинстве случаев мы имеем дело, всегда дискретны и конечны. Любое многомерное дискретное конечное пространство можно свести к одномерному, выбрав подходящим образом правило нумерации его точек.

Всегда можно подобрать такой способ нумерации, что в одномерном представлении элементы выборки одного класса окажутся собранными в одну компактную область, и тем самым в одномерном представлении будет только одна точка, разделяющая классы. Не обязательно при этом, что такое разделение будет иметь какой либо смысл, отвечающий постановке конкретной задачи. При этом следует учитывать, что такие пространства, отражающие реалии с должной точностью, могут иметь многие миллионы или миллиарды точек.

Переход к многомерным решеткам, которые чаще всего используются в конкретных задачах, может быть осуществлен методом, который демонстрируется для двумерной решетки.

Для простоты рассмотрения мы будем считать, что наша решетка состоит из N столбцов и N строк и выборок, принадлежащих разным классам K_1 и K_2.

Каждый столбец и каждую строку можно интерпретировать как одномерные решетки, для которых выполнить процедуры поиска разделяющих функций $F_y(x)$ и $F_x(y)$.

Вычислительная сложность данного процесса определяется размерами выборок и числом точек переключения ω_{xi} и ω_{yj} по соответствующим строкам и столбцам.

Важной проблемой классификации по приведенной методике в многомерном случае является проблема согласования присваивания значения класса узлу решетки с координатами (x, y), не принадлежащему ни одной из выборок.

Она состоит в том, что по строкам этот узел может быть квалифицирован как принадлежащий классу K_1, а по столбцам он может оказаться, принадлежащим классу K_2.

Эта проблема технически решаема на основе соответствующих соглашений, учитывающих семантику решаемой задачи.

Еще раз подчеркнем, что и в многомерном случае мы полагаемся на метрику Евклида, что не всегда отвечает существу конкретной задачи.

В одномерном и в многомерном случае мы ищем параметризованную функцию $F(X, \omega_1, \omega_2, \omega_3, \ldots, \omega_m)$, где параметры в наших примерах суть нули полинома степени m.Используя идеи эволюционных вычислений можно попытаться сконструировать генетический алгоритм [4], решающий поставленную задачу.

В качестве фитнес-функции [5, 6] будет выступать число неправильных классификаций представителей выборок в областях покрытия рассматриваемого пространства расположения объектов зонами отрицательных и положительных значений функции $F(X, \omega_1, \omega_2, \omega_3,\ldots, \omega_k)$. Эти зоны индуцируются взаимным расположением точек ω.

Задача сведется к минимизации числа противоречий в определении принадлежностей к своему классу выборок, попавших в соответствующие зоны. Заметим, что эту задачу можно трактовать как задачу о минимальных покрытиях.

Для поиска глобального экстремума так заданной функции мы используем традиционную схему генетического алгоритма.

Возьмем наугад несколько значений ω_i, принадлежащих области определения функции $F(X, \omega_1, \omega_2, \omega_3,\ldots, \omega_k)$.

Строка этих значений $\omega_1, \omega_2, \omega_3,\ldots, \omega_k$ будет нашим геномом. В качестве популяции будет выступать несколько таких строк.

В отличие от обычного генетического алгоритма, длина геномов в одной популяции может быть различной. Соответственно, операции кроссовера и мутации должны быть иными. При равенстве значений фитнесс-функции более высокое качество придается геному меньшей длины.

Мы не будем далее углубляться в детали реализации такого алгоритма. Как и во всяком эволюционном алгоритме, основанном на методе «проб и ошибок», в ходе итерационного процесса здесь , по-видимому, будут более сложные критерии оценки достоверности решений.

Вернемся к постановке задачи кластеризации.

В случае постановки задачи, когда под кластеризацией понимается поиск *границ* областей *сосредоточения* объектов, *обладающих заданными свойствами,* нам следует точно определить понятия: *сосредоточения* объектов, понятие *обладания заданными свойствам,* понятие поиска *границ* областей.

Будем считать, что все значения признаков представляют собой двоичные наборы, которые можно интерпретировать как представление чисел в двоичной системе.

Это позволяет нам рассматривать вектор характеристик объекта как точку в многомерном векторном пространстве.

Понятие *обладания заданными свойствам* означает существование вычислимой функции от вектора характеристик, принимающей значения, идентифицирующие степень наличия заданного свойства у объекта, описываемого допустимым вектором характеристик.

Свойство может быть также задано как логическая функция от координат вектора, которая фиксирует наличие или отсутствие заданного свойства у вектора. Коль скоро нам заданы такие функции, то поставленная задача сводится к построению плотностей распределения объектов в рассматриваемом пространстве.

Рассмотрим более интересный случай автоматической кластеризации. Эта задача формулируется так.

Дано: неупорядоченное множество объектов, заданных векторами своих числовых характеристик, для простоты рассмотрения - наборами двоичных слов одинаковой длины.

Требуется выделить группы схожих элементов, т. е. ничего заранее не зная, найти функцию, которая при подстановки в нее слова определяла бы номер класса схожих ему элементов. Без дополнительных условий и гипотез такую задачу нельзя решать.

Задача может считаться поставленной, когда формально определены понятия *схожести* или введена метрика в пространстве объектов (двоичных векторов).

Схожесть может быть определена перечислением представителей схожих объектов, и тогда эта задача становится задачей классификации по прецедентам и может быть решена нейросетевыми [7] методами.

Нейросеть, решающая такую задачу, по-существу, является способом введения подобия меры.

Классические методы математической статистики, позволяющие оценивать частоты встречаемости слов и частей слов, широко применяемы в обработке данных. Методы, так называемого, интеллектуального анализа данных, главным образом основанные на статистике, дают возможность выявлять группы элементов, которые

можно характеризовать как схожие и тем самым автоматически находить прецеденты, точнее, формировать обучающие выборки для дальнейшего конструирования классифицирующих функций методами нейросетевых или эволюционных вычислений.

Литература

1. Ту Дж., Гонсалес Р. Принципы распознавания образов: Пер. с англ. // М.: Мир, 1978.
2. Розенблатт Ф. Принципы нейродинамики. // «Наука», М., 1965.
3. С. Осовский. Нейронные сети для обработки информации. // М.: «Финансы и статистика», 2002.
4. Holland J. Adaptation in natural and artificial systems. // MIT press, Cambridge MA, 1992.
5. Srinivas M, Patnaik L. Genetic Algorithms, a survey. //J. Computer, v.27, № 6, 28-43, IEEE press. June 1994.
6. Forrest S. Genetic Algorithm: Principles of Natural Selection Applied to Computation // J. Sciense. V.261, 872-878, August 1993.
7. Рудаков К.В., Чехович Ю.В. О проблеме синтеза обучающих алгоритмов выделения трендов (алгебраический подход) // Прикладная математика и информатика. # 8. М.: Изд. отдел ф-та ВМиК МГУ, 2001. С. 97-113.

Аннотации

Рябов Г.Г., Серов В.А. Компьютерные комбинаторно-топологические построения и их преобразования // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

Рассматриваются симплициально-решеточные модели на множествах целых точек **Zn** и примитивных векторов в евклидовых пространствах **Rn** с позиций эффективной компьютерной реализации. Инструментальная система на базе этих моделей представлена как вариант прототипа комбинаторного топологического процессора, который в качестве сопроцессора суперкомпьютера значительно расширяет вычислительные возможности.

Ил.: 8. Библ.: 20 назв.

Брусенцов Н.П., Владимирова Ю.С. Конструктная компьютеризация аристотелевой силлогистики // Программные системы и инструменты. Тематический сборник № 8. М.: Изд-во факультета ВМиК МГУ, 2007.

Осуществлено исчерпывающая алгебраизация и конструктная эффективная компьютеризация аристотелевой силлогистики на основе гераклитова диалектического принципа сосуществования противоположностей.

Доложено на Ломоносовских чтениях 2007 г. на факультете ВМиК МГУ.

Ил.: 1. Библ.: 7назв.

Леонов М.В., Иванов А.В., Клеменков П.А., Пейсахов И.Б. О мобильных практикумах и информационных системах. // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

Сообщается об опыте подготовки и использования мобильных практикумов по программированию и мобильных информационных систем для специалистов по таксономической ботанике. Подготовленные программные комплексы загружаются с флэш-брелков, не требуют инсталляции на жесткий диск, имеют версии для среды Windows и Linux.

Ил.: 3. Библ.: 4 назв.

Махнычев В.С., Ильичев А.Б. Распараллеливание задачи о составлении вузовского расписания с использованием распределенного обхода дерева перебора // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

Решение задачи о составлении расписания затруднено огромным количеством различных вариантов, из которых предстоит

сделать выбор. Поэтому для решения задачи применяют в основном эвристические алгоритмы с небольшими элементами перебора. В данной работе предпринята попытка улучшить качество результатов, увеличив возможности перебора за счет использования распределенной системы. Примененная схема параллельного обхода дерева перебора была ранее испытана на программе игры в шахматы.

Для демонстрации эффективности описанных методов использовались различные тестовые наборы, приближенные к реальной задаче для отдельно взятого факультета.

Ил.: 0. Библ.: 2 назв.

Попова Е.А. Анализ масштабируемости и производительности параллельного алгоритма построения ансамблей деревьев решений для задачи локализации нейронных источников // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

В работе описан метод параллельного ансамбля деревьев решений применительно к задаче локализации нейронных источников внутри мозга. На основе анализа сигналов электроэнцефалографии (ЭЭГ) идея предложенного параллельного численного метода — рассматривать параметры источников как атрибуты деревьев решений, которые строятся параллельно. Метод основан на построении тренировочного набора данных по экспериментальному сигналу и дальнейшего построения классификатора на основании значения ошибки по потенциалу — разницы экспериментальных измерений и модельного потенциала. Рассматривается эффективность распараллеливания задачи локализации - распределения данных между процессорами и распределенного обучения ансамблей деревьев решений. Представлен анализ масштабируемости с числом процессоров задачи построения ансамблей деревьев решений при решении задачи локализации нейронных источников на многопроцессорных вычислительных комплексах. Параллельный алгоритм локализации источников разработан для архитектур с общей и распределенной памятью. Реализация выполнена с помощью MPI, в работе также обсуждается гибридная модель параллельных вычислений на MPI и OpenMP.

Ил: 8. Библ.: 17 назв.

Иванов С.А. Методы поиска закономерностей в символьных последовательностях // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

Существует три наиболее распространённых трактовки понятия “структура”: структура как набор зависимостей между элементами, таксономическая структура, иерархическая структура. В работе

рассматривается решение задачи определения структуры символьной последовательности как набора зависимостей между её элементами (отдельными символами алфавита), отстоящими друг от друга не более чем на некоторое заданное число символов. В работе предлагается метод поиска шаблонных зависимостей в последовательностях, основанный на кластеризации подслов анализируемой последовательности, чья сложность не зависит от длины анализируемой последовательности. Рассматриваются два различных метода построения кластеров подслов: метод с использованием генетического алгоритма и вариант метода K-средних.

Ил.:0. Библ.: 7 назв.

Нестеров С.В. Исследование ошибок измерения в задаче усвоения метеоданных со спутников–скаттерометров // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

В настоящее время сложилась ситуация, когда отсутствуют какие-либо исследования допустимости использования данных со спутников-скаттерометров для задачи усвоения данных с использованием фильтра Калмана. Для этого необходимо сконструировать вычислительное средство для изучения закона распределения ошибок. В данной дипломной работе на основе исследований закона распределения ошибок измерений спутников – скаттерометров для задачи усвоения метеоданных был предложен алгоритм получения эмпирического распределения спутниковых ошибок с помощью аналогичных данных с кораблей и буев. Для вычисления исследуемых распределений создана программная реализация, ориентированная на специальные форматы метеоданных. На основе аппарата характеристических функций случайных величин было получено распределение ошибок спутниковых измерений. Также на основе исследований было сделано несколько полезных выводов о характере распределения ошибок.

Ил.: 2. Библ.: 6 назв.

Позднеев А.В. Параллельный код частиц в ячейке для моделирования процессов в масс-спектрометре // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

В работе описан параллельный трехмерный код моделирования процессов в масс-спектрометре ионного циклотронного резонанса с преобразованием Фурье (FTICR). Масс-спектрометры являются основными инструментами, используемыми в протеомике для измерения масс биомолекул. Приводится математическая постановка задачи. Для ее решения предложено использовать метод «частиц в

ячейке» (PIC). Чтобы точно воспроизводить эффект ионного циклотронного резонанса, в схеме интегрирования уравнений движения используется поправочный множитель. Реализация метода выполнена на языке Fortran 90, код распараллелен с использованием директив OpenMP. При численном решении уравнения Пуассона используется библиотека FFTW. Проведен анализ зависимости ускорения отдельных шагов PIC-алгоритма от числа процессоров. Рассматриваются вопросы масштабируемости кода при увеличении размера задачи. Все расчеты проведены на SMP-системе факультета ВМК МГУ IBM pSeries 690 Regatta.

Ил: 8. Библ.: 15 назв.

Семенов А.С., Эйсымонт Л.К. Параллельное умножение матриц на суперкомпьютере с мультитредово-потоковой архитектурой // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

В статье на примере эффективно решаемой на суперкомпьютерах с традиционной архитектурой задачи умножения плотнозаполненных матриц показана возможность не менее эффективного решения таких задач с высокой пространственно-временной локализацией на разрабатываемом суперкомпьютере с перспективной мультитредово-потоковой архитектурой. Особенность полученного результата состоит в том, что разрабатываемый суперкомпьютер ориентирован прежде всего на эффективное решение другого класса задач с плохой пространственно-временной локализацией, на которых традиционные суперкомпьютеры показывают крайне низкие результаты по эффективности.

Ил.7. Библ. 10 назв.

Решетов Е.В. Алгоритм нахождения величины компонент на примере решения задачи определения минерального состава горных пород // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

В работе исследуется круг задач, связанных с определением компонентного состава некоторых объектов на основе измерений различных свойств данных объектов. Примером подобных задач являются определение компонентного состава воды, горных пород или газа, на основе измерения ряда физических величин: плотности, электрического сопротивления, радиоактивности и других параметров.

Ил.: 2. Библ.: 5 назв.

Погуляй Е.В. Разработка алгоритма навигации биомолекулярного нанороботa для задач фармокологии // Программные

системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

В работе рассматривается одна из наиболее ресурсоемких задач области моделирования лекарств, а именно задача поиска небольших молекул, способных взаимодействовать с выбранным целевым протеином. Таким образом создается одна или несколько коллекций молекул, эффективно взаимодействующих с определенным целевым протеином. Далее эти молекулы используются в качестве основы для синтеза конечного лекарства, обладающего всеми необходимыми свойствами. Для ее решения предложено использовать метод опорных векторов (SVM). Реализация метода выполнена на языке C++, визуализация результатов проводиться средствами библиотеки OpenGL. Исходные данные взяты из библиотеки NIST. Проведено численное исследование построенного метода.

Ил: 2. Библ.: 8 назв.

Волканов Д.Ю., Черей М.В. Исследование применимости алгоритмов нечёткого поиска для анализа результатов имитационного моделирования ВСРВ // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

В статье исследуется применимость алгоритмов нечёткого поиска для анализа результатов имитационного моделирования ВСРВ, приводятся примеры практических задач для которых эффективно применение алгоритмов нечёткого поиска.

Ил.: 2. Библ.: 9 назв.

Балаханов В.А., Костенко В.А. Способы сведения задачи построения статико-динамического однопроцессорного расписания для систем реального времени к задаче нахождения на графе маршрута // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

Для того чтобы использовать муравьиный алгоритм для решения конкретной задачи оптимизации, необходимо свести эту задачу к задаче нахождения на графе маршрута, обладающего определенными свойствами. В данной работе предложены два способа сведения задачи построения статико-динамического однопроцессорного расписания для систем реального времени к задаче нахождения на графе маршрута и выделены преимущества и недостатки каждого из способов.

Ил.: 3. Библ.: 6 назв.

Бычков И.А., Костенко В.А. Возможность использования арбитражного кольца Fibre Channel в вычислительных системах

реального времени // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

В данной работе рассматривается один из возможных подходов к построению среды обменов вычислительных систем реального времени на основе арбитражного кольца Fibre Channel, гарантированно обеспечивающий корректность выполнения обменов данными в режиме реального времени, а также алгоритм построения расписания обменов в арбитражном кольце.

Ил.: 0. Библ.: 8 назв.

Королев Л.Н, Попов А.М., Попова Н.Н., Зленко П.А., Мещеряков Д.К., Певцов С.Е., Позднеев А.В., Попова Е.А., Сальников А.Н., Федулова И.А. Развитие информационной системы, направленное на задачи идентификации биомолекулярных структур с использованием нейросетей и генетических алгоритмов // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

Сообщается о развитии информационной системы, нацеленной на обработку распределенных экспериментальных данных большого объема с использованием нейросетей, эвалюционных алгоритмов и других эффективных методик. Приведены некоторые результаты по решению актуальных задач.

Ил.: 1. Библ.: 7 назв.

Королев Л. Н. О некоторых проблемах компьютерной реализации алгоритмов кластеризации и классификации // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.

В статье рассматриваются схемы простых алгоритмов классификации и кластеризации с целью выделения проблем, возникающих при формализации и компьютерной реализации таких алгоритмов.

Ил.: 1. Библ.: 7 назв.