УДК 681.3

Е.А. Яценко

О ПРИМЕНЕНИИ МАШИННОГО ОБУЧЕНИЯ ДЛЯ ПРОЕКТИРОВАНИЯ АДАПТИВНЫХ ПРОГРАММ СОРТИРОВКИ В АЛГЕБРЕ АЛГОРИТМОВ

Проведен эксперимент по разработке адаптивной программы сортировки на основе использования метода выбора алгоритмов, системы машинного обучения и алгеброалгоритмического подхода. Средства машинного обучения позволяют автоматизировать построение адаптивного алгоритма на основе анализа экспериментальных данных, связанных с выполнением исходных алгоритмов, имеющихся в распоряжении разработчика. Для проектирования алгоритмов использован язык, основывающийся на системах алгоритмических алгебр.

Введение

В работах [1–3] создана инструментальная система программирования, названная Интегрированным инструментарием Проектирования и Синтеза программ (ИПС), основывающаяся на системах алгоритмических алгебр (САА) В.М. Глушкова [1]. В системе совместно используются три формы представления знаний: аналитическая (формула в алгебре алгоритмов), естественно-лингвистическая (текстовая) и графовая (граф-схемы). Особенность инструментария состоит также в использовании метода диалогового конструирования синтаксически правильных программ, ориентированном на исключение возникновения синтаксических ошибок в процессе проектирования. В упомянутых работах система была применена для синтеза многопоточных, кластерных, Грид-программ по схемам асинхронных алгоритмов. При этом исследовалось также совместное использование ИПС и системы переписывания правил TermWare [4] для автоматизации трансформации программ.

В последнее время все больше возрастает необходимость в разработке программных приложений, которые способны не только выполнять однажды заданную в алгоритме последовательность действий над предварительно определенными данными, но и способны сами анализировать поступающую информацию, находить в ней закономерности, выполнять прогнозирование и т. д. Создание подобных программ связано с необходимостью накопления и анализа большого количества экспериментальных данных (входных, промежуточных и результирующих). При этом возникает проблема автоматизации извлечения полезной информации из большого объема данных. Решение данной задачи осуществляется в рамках технологии интеллектуального анализа данных (ИАД) (Data Mining) [5] – одной из активно развивающихся областей информационных технологий, предназначенной для выявления полезных знаний из баз данных различной природы.

Учитывая вышесказанное в данной работе проведен эксперимент по разработке адаптивной программы сортировки, совмещающий проектирование в алгебре алгоритмов и интеллектуальный анализ данных. Под адаптивным понимается алгоритм, учитывающий характеристики входных данных и выбирающий наилучшую стратегию их обработки. При этом использована концепция задачи *выбора алгоритма* [6]. Цель упомянутой задачи – это ответ на вопрос “Какой из имеющихся алгоритмов выполнится наилучшим образом при решении заданной вычислительной проблемы?”. Исследованию данной задачи посвящены работы в различных предметных областях: прогнозирование временных рядов [7]; задачи удовлетворения ограничений (constraint satisfaction problem) [8]; оптимизация [9]; сортировка [10, 11]. В упомянутых работах автоматизация выбора алгоритмов выполняется с использованием ИАД и методов машинного обучения.

ISSN 1727-4907. Проблеми програмування. 2011. № 2

Суть предложенного в данной работе подхода к разработке адаптивной программы сортировки состоит в следующем. Исходные алгоритмы сортировки проектируются с использованием алгебраических средств в системе ИПС и затем выполняются на тестовой совокупности числовых массивов. Далее полученные экспериментальные данные анализируются в системе интеллектуального анализа данных Weka [12] с применением машинного обучения. Полученная в результате анализа обучающая модель (дерево решений) преобразуется в алгебраическую спецификацию (САА-схему) адаптивного алгоритма, содержащую информацию о выборе наилучшей стратегии сортировки в зависимости от характеристик входных данных. Затем схема преобразуется в код на целевом языке программирования (C++). В работе проведено сравнение производительности исходных алгоритмов и разработанной адаптивной программы.

Материал работы состоит из разделов. В разделе 1 рассматривается концепция задачи выбора алгоритма и методы машинного обучения, связанные с ее решением. Раздел 2 посвящен результатам проведенного эксперимента по разработке программы выбора алгоритма сортировки.

1. Проблема выбора алгоритма и машинное обучение

Цель задачи выбора алгоритма – это выбор наилучшего алгоритма для заданной вычислительной проблемы в соответствии с некоторыми ее характеристиками. Формальная модель проблемы выбора алгоритма была разработана Д. Райсом [6]. Упомянутая модель может быть использована для ответа на вопрос: “Какой из имеющихся у разработчика алгоритмов наилучшим образом подходит для решения задачи и определенных входных данных?”. Основными компонентами модели являются следующие:

- пространство задач P представляет множество экземпляров класса задач;
- пространство свойств F содержит измеримые характеристики (метрики) задач P;
- пространство алгоритмов A является множеством всех рассматриваемых алгоритмов, предназначенных для решения задачи;
- пространство производительности Y представляет отображение каждого алгоритма во множество показателей производительности.

Далее проблема выбора алгоритма может быть сформулирована следующим образом.

Для заданного экземпляра задачи $x \in P$, со свойствами $f(x) \in F$, необходимо найти отображение $S(f(x))$ в пространство алгоритмов A, такое, что выбранный алгоритм $\alpha \in A$ максимизирует отображение производительности $y(\alpha(x)) \in Y$.

Выбор свойств F в значительной степени зависит от предметной области и выбранных алгоритмов. Например, при рассмотрении алгоритмов сортировки последовательностей чисел, свойством является степень отсортированности входной последовательности. Показателем производительности может служить время выполнения алгоритма в секундах.

С использованием правил “если-то” решение задачи выбора алгоритма может быть, например, сформулировано так:

если входящие данные имеют характеристики C_1, C_2, ..., C_n, *то использовать алгоритм* A_1 *вместо алгоритма* A_2.

Выбор алгоритма может осуществляться статически или динамически. Статическая программа выбора алгоритма делает выбор до начала выполнения алгоритма и затем передает управление выбранному алгоритму. В отличие от статической, динамическая система выбора алгоритма может изменять его выбор динамически, контролируя выполнение алгоритма.

В реальных ситуациях, выбор алгоритма выполняется вручную экспертами, которые имеют хорошее теоретическое понимание вычислительной сложности различных алгоритмов и хорошо знакомы с их свойствами во время выполнения. Одним из способов автоматизации выбора

алгоритма является применение машинного обучения [6].

Проблеме автоматизации выбора алгоритмов для задач сортировки посвящены работы [10, 11]. В работе [10] при выборе алгоритма сортировки в качестве единственного свойства задачи рассматривается длина последовательности, подлежащей обработке. В процессе выбора алгоритма использовано динамическое программирование. Более подробное исследование проведено в [11] с использованием машинного обучения. В упомянутой работе помимо размера массива учитывается степень его отсортированности. Исследованы три метрики отсортированности: количество перестановок (inversions, INV); количество возрастающих подпоследовательностей (RUN); длина наибольшего подмассива (LAS). Показано, что наиболее эффективным является использование единственной метрики – RUN.

Данная работа также посвящена решению задачи выбора алгоритмов сортировки; при этом использован подход, разработанный в [11]. Основным отличием данной работы является использование алгеброалгоритмического подхода при разработке алгоритмов, позволяющего использовать высокоуровневые спецификации, не зависящие от реализации на конкретном языке программирования.

В работе [11] проблема выбора алгоритма сформулирована как задача классификации, являющаяся одним из методов машинного обучения.

Определение. Пусть имеется набор объектов, каждый из которых принадлежит одному из m классов. Задачей *классификации* является составление правила, по которому для любого объекта можно с большой степенью достоверности определить класс, которому данный объект принадлежит [5].

Пусть $X = \{x_1, ..., x_k\}$ – множество атрибутов объекта, $Y = \{1, \ldots, m\}$ – множество номеров (или наименований) классов. В результате классификации должна быть получена *целевая функция* $f: X \to Y$. Целевая функция $f(x_1, ..., x_k)$ сопоставляет объекту с атрибутами $x_1, ..., x_k$ соответствующий номер (метку) класса, к которому этот объект принадлежит.

Например, в случае задачи выбора алгоритма сортировки объектами задачи классификации являются обрабатываемые массивы [11]. Атрибутами объектов являются:

- x_1 – длина массива;
- x_2 – степень отсортированности.

Метками классов являются названия различных алгоритмов сортировки (сортировка вставками, быстрая и т. д.). Функция $f(x_1, x_2)$ сопоставляет каждому входному массиву название наилучшего (по времени выполнения) алгоритма, которым его необходимо сортировать. Более подробно пример описания данных для данной задачи рассматривается в разделе 2.

В распоряжении у исследователя обычно имеется некоторый набор объектов, у которых метка класса уже известна. Эти объекты могут быть использованы для обучения модели, т. е. подбора параметров модели классификации и для тестирования построенной модели классификации.

Классификация с обучением подразумевает следующие действия:

1) подготовка данных. Имеющийся набор объектов с известными метками классов разбивается на две части: обучающую выборку и тестовую выборку. Желательно, чтобы это разбиение было произведено случайным образом. Чаще всего обучающая выборка имеет размер больше, чем тестовая;

2) обучение модели. Параметры модели классификации подбираются на основе обучающей выборки таким образом, чтобы добиться наилучшего соответствия между предсказанными и фактическими метками классов;

3) тестирование модели. Полученная в результате обучения модель проверяется на достоверность. Для этого вычисляется процент неверных результатов классификации объектов из тестовой выборки.

Классификация с обучением имеет множество приложений, например, в таких областях, как кредитование, медицинская

диагностика, предсказание доходов, маркетинг.

К известным методам классификации с обучением относятся деревья решений, нейронные сети, сети Байеса [5]. В работе [11] показано, что наилучшим в случае задачи выбора алгоритмов сортировки является использование деревьев решений.

Дерево решений – это дерево, в котором каждой внутренней вершине поставлен в соответствие некоторый атрибут, каждая ветвь, выходящая из данной вершины, соответствует одному из возможных значений атрибута, а каждому листу дерева сопоставлен конкретный класс или набор вероятностей классов [5, 13]. На рис. 1 в качестве примера представлено упрощенное дерево решений для выбора одного из алгоритмов сортировки в зависимости от размера массива. Дерево построено на основе экспериментальных данных, приведенных в работе [10]. В разделе 2 рассматривается построенное дерево решений, дополнительно учитывающее степень отсортированности обрабатываемых массивов.

Для того чтобы классифицировать новый объект, необходимо двигаться по дереву сверху вниз, начиная с корня. При этом на каждом внутреннем узле дерева выбирается та ветвь, которая соответствует фактическому значению соответствующего атрибута. Добравшись до листа дерева, получаем тот класс, которому принадлежит объект согласно классифицирующему правилу. Существуют различные алгоритмы построения деревьев решений: ID3, C4.5, NewId, ITrule, CN2 и др. [5].

В данной работе, как и в [11], для проведения экспериментов была выбрана система машинного обучения Weka (Waikato Environment for Knowledge Analysis) [12]. Weka представляет собой набор средств визуализации и библиотеку алгоритмов машинного обучения для решения задач интеллектуального анализа данных и прогнозирования, с графической пользовательской оболочкой для доступа к ним. Система является свободным программным обеспечением, разработанным на языке Java в университете Вайкато (Новая Зеландия). На сегодняшний день она является лучшей библиотекой с открытым кодом для анализа данных. Weka позволяет непосредственно применять алгоритмы к выборкам данных, а также вызывать алгоритмы из программ на Java. Для использования Weka из систем, реализованных на других платформах, возможен вызов алгоритмов через интерфейс командной строки. Система позволяет выполнять такие задачи анализа данных, как:

- подготовка данных – предварительная обработка (preprocessing);
- отбор признаков (feature selection);
- кластеризация;

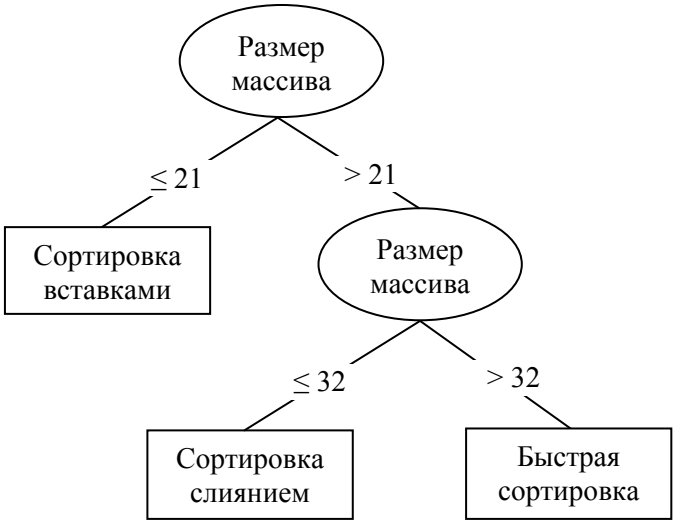

Рис. 1. Пример дерева решений

• классификация, в частности, деревья решений;

• поиск ассоциативных правил;

• регрессионный анализ;

• визуализация результатов.

Система Weka также подходит для разработки новых подходов в машинном обучении. Пользователями Weka являются исследователи в области машинного обучения и прикладных наук, она также широко используется в учебных целях.

Исходные данные в упомянутой системе представлены в виде матрицы признаковых описаний объектов, пример такой таблицы рассматривается в подразделе 2.2.

2. Применение метода выбора алгоритма для сортировок

Цель проведенного в данной работе эксперимента состояла в построении адаптивного алгоритма сортировки массивов на основе имеющихся, с использованием метода выбора алгоритма и машинного обучения, рассмотренных в разделе 1. Были использованы 5 алгоритмов: сортировка вставками (insertion sort), сортировка Шелла (shell sort), пирамидальная сортировка (heap sort), сортировка слиянием (merge sort), быстрая сортировка (quick sort) [14]. В адаптивной сортировке учитываются свойства обрабатываемого массива – длина и степень неотсортированности. Для проектирования алгоритмов использованы средства САА (см. подраздел 2.1); затем алгоритмы реализованы на языке C++.

Эксперимент состоял из следующих этапов:

1) подготовка обучающих данных (обучающей выборки). Произведена генерация входных массивов с различными характеристиками и затем каждый из алгоритмов сортировки по очереди был выполнен на этих данных. Алгоритм, выполнившийся за наименьшее время, был обозначен как *наилучший*. Собственно обучающие данные являются таблицей с информацией о размере массива, степени его неотсортированности и наилучшем алгоритме;

2) выполнение обучающего алгоритма, а именно, обучения с использованием дерева решений (см. раздел 1) на обучающей выборке в системе Weka;

3) преобразование дерева решений в САА-схему;

4) генерация по САА-схеме кода на целевом языке программирования;

5) сравнение времени выполнения адаптивного алгоритма и исходных алгоритмов на тестовом наборе массивов.

Перечисленные этапы более подробно рассматриваются в подразделе 2.2.

2.1. Формализованное проектирование программ в системах алгоритмических алгебр. В основу проектирования программ сортировки в рамках данной работы положен аппарат САА (системы алгоритмических алгебр) В.М. Глушкова [1]. САА являются двухосновной алгеброй $< U, B; \Omega >$, где U – множество операторов, B – множество логических условий. Сигнатура операций $\Omega = \Omega_1 \cup \Omega_2$ состоит из системы Ω_1 логических операций и операторных Ω_2 операций. На САА базируется алгоритмический язык САА/1, используемый для проектирования программ в ИПС. Данный язык предназначен для многоуровневого структурного проектирования и документирования последовательных и параллельных алгоритмов и программ. Преимуществом его использования является возможность описания алгоритмов в естественно-лингвистической форме (САА-схемы), удобной для человека, что облегчает достижение требуемого качества программ. Перечислим основные операции, используемые для разработки схем алгоритмов.

Составные условия строятся из базисных с помощью логических операций САА:

• дизъюнкция: ‘условие1’ ИЛИ ‘условие2’;

• конъюнкция: ‘условие1’ И ‘условие2’;

• отрицание: НЕ ‘условие’.

Составные операторы строятся из элементарных посредством операций:

• “оператор1” ЗАТЕМ “оператор2” – последовательное выполнение операторов;

• ЕСЛИ ‘условие’ ТО “оператор1”

ИНАЧЕ “оператор2” КОНЕЦ ЕСЛИ – оператор ветвления;

• ПОКА НЕ ‘условие_окончания_цикла’ ЦИКЛ “оператор” КОНЕЦ ЦИКЛА – оператор цикла.

В работах [1, 3] аппарат САА-М применен при решении задач символьной мультиобработки: последовательных и параллельных сортировок, поиска, языкового процессирования и др.

В качестве примера приведем САА-схему сортировки вставками входного массива A длины n:

```
СХЕМА insertionSort(A, n)
==== ДЛЯ '(i) от (1) до (n)'
     ЦИКЛ
       "(temp := a[i])"
       ЗАТЕМ
       "(j := i - 1)"
       ПОКА НЕ 'temp < a[j]'
               И
               'j >= 0'
       ЦИКЛ
          "(a[j + 1] := a[j])"
          ЗАТЕМ
          "(j := j - 1)"
       КОНЕЦ ЦИКЛА
       "(a[j + 1] := temp)"
     КОНЕЦ ЦИКЛА
КОНЕЦ СХЕМЫ insertionSort(A, n)
```

2.2. Результаты эксперимента. Для проведения эксперимента были подготовлены 5 алгоритмов сортировки, перечисленные в начале раздела 2, а также разработана программа (C++), позволяющая вызывать данные алгоритмы и сравнивать время их выполнения на различных входных массивах. Впоследствии данная программа также была дополнена адаптивным алгоритмом, выбирающим наилучший алгоритм сортировки в зависимости от длины и степени неотсортированности входного массива.

В данной работе степень неотсортированности массива A длины n вычисляется по формуле [11]

$$runs'(A) = \frac{runs(A)}{n},$$

где $runs(A)$ – количество возрастающих (отсортированных) подмассивов (англ. “runs up”) в сортируемом массиве. Метрика $runs'(A)$ принимает значения в диапазоне (0 … 1].

Например, для массива

$$A = < |10| \ 4 \ 5 \ 7| \ 1 \ 3| \ 2 \ 6 \ 9| \ 8| >,$$

$$runs(A) = 5;$$
$$runs' = 0.5.$$

Для отсортированного массива длины n степень неотсортированности $runs'(A) = 1 / n$; для массива чисел, расположенных в обратном порядке $runs'(A) = n / n = 1$.

В работе [11] показано, что для рассматриваемой задачи использование приведенной метрики является наиболее эффективным.

Для сравнения производительности алгоритмов сортировки на различных входных массивах была разработана программа генерации числовых массивов с различной степенью неотсортированности. С помощью данной программы подготовлен набор из 800 числовых массивов размером от 10 до 100 элементов. Упомянутый набор включал следующие типы массивов:

• 500 массивов со случайно перемешанными элементами, для генерации которых использован алгоритм “Algorithm 235” (Random Permutation) [15];

• 150 почти отсортированных массивов (nearly-sorted arrays [11, 16]). Такие массивы сгенерированы путем занесения в каждый массив последовательности чисел 1, 2, ..., n. Затем выполнена перестановка случайных пар элементов, количество которых составляет от 0 до 10 % от длины массива;

• 150 массивов с элементами, расположенными в обратном порядке, с незначительным количеством перестановок. Генерация упомянутых массивов производится аналогично предыдущим, но в этом случае в массив вначале заносится обратная последовательность чисел: n, n–1, ..., ..., 2, 1.

В разработанной программе сортировки для каждого массива из упомянутого набора вначале вычисляется степень неотсортированности и затем по очереди за-

пускаются 5 различных алгоритмов сортировки. Длина входного массива, величина *runs'*, а также название наилучшего алгоритма (который выполнился за наименьшее количество времени), записывается в файл в виде таблицы (см. табл. 1). Данный файл представляет собой набор обучающих данных, передаваемых в систему Weka. Поле "Наилучший алгоритм", указанное в таблице, является целевым атрибутом, значение которого будет прогнозироваться на основе значений других атрибутов обрабатываемого массива. В результате эксперимента наилучшими в различных случаях оказались два из пяти рассматриваемых алгоритмов – сортировка вставками и быстрая сортировка.

На рис. 2 и рис. 3 показаны графики распределения величин "размер массива" и "степень неотсортированности" для рассматриваемого набора обучающих данных.

В системе Weka к полученным экспериментальным данным был применен алгоритм построения дерева решений J4.8 (реализация алгоритма C4.5 на языке Java [12]), результирующее дерево показано на рис. 4. Листья дерева соответствуют выбору одного из алгоритмов. Из рисунка видно, в частности, что при небольших значениях степени неотсортированности, $runs' \leq 0.4$, наилучшим является алгоритм сортировки вставками не зависимо от размера массива. В остальных случаях выбор алгоритма зависит от размера и значения *runs'*. Точность классификации с помощью данной модели при проверке на рассматриваемых обучающих данных составила 93.625 %.

Далее на основе построенного дерева система Weka может использоваться для прогнозирования наилучшего алгоритма для произвольного входного массива. Для этого в систему необходимо передать файл с таблицей, аналогичной таблице 1, в которой приведена строка (или несколько строк) со значением размера массива, степени неотсортированности, а в поле "Наилучший алгоритм" указать символ "?". Однако, при вызове данной системы из программы сортировки возникают дополнительные накладные расходы по времени. Поэтому в данной работе дерево решений преобразуется сначала в САА-схему, а затем в подпрограмму на целевом языке программирования (C++), которая включается в программу сортировки.

По полученному дереву была построена САА-схема выбора алгоритма сортировки `adaptiveSort(A, n)`, приведенная далее.

В приведенной схеме для входящего массива, подлежащего сортировке, вначале вычисляется степень неотсортированности. Затем в зависимости от длины массива и величины *runs'* вызывается соответствующий алгоритм. Полученный алгоритм был назван *адаптивным* алгоритмом сортировки.

Таблица 1. Пример обучающих данных, поступающих на вход системы Weka

Номер массива	Размер массива (n)	Степень неотсортированности ($runs'$)	Наилучший алгоритм
1	10.0	0.1	insertion
2	10.0	0.3	insertion
3	10.0	0.4	insertion
4	10.0	0.5	insertion
5	10.0	0.6	insertion
6	10.0	0.7	insertion
7	10.0	0.9	quick
8	10.0	1.0	quick

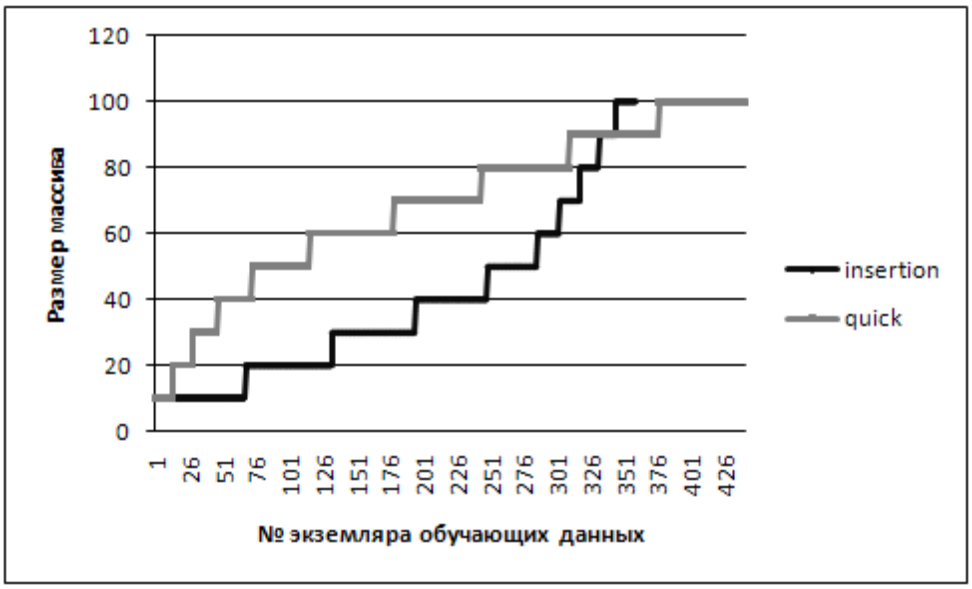

Рис. 2. Распределение величины “размер массива”
для обучающих данных со значениями
“insertion” и “quick”

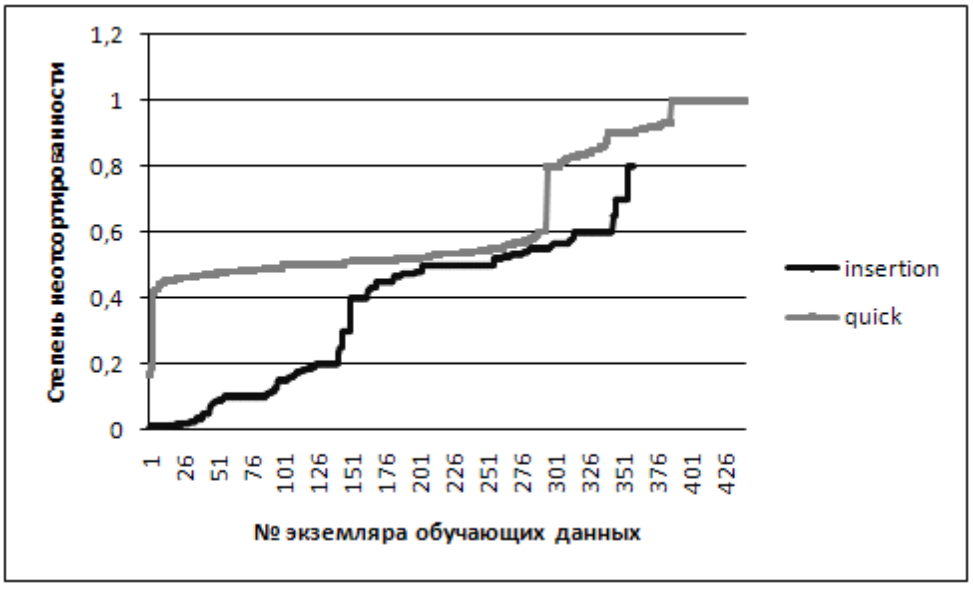

Рис. 3. Распределение величины “степень неотсортированности”
для обучающих данных со значениями
“insertion” и “quick”

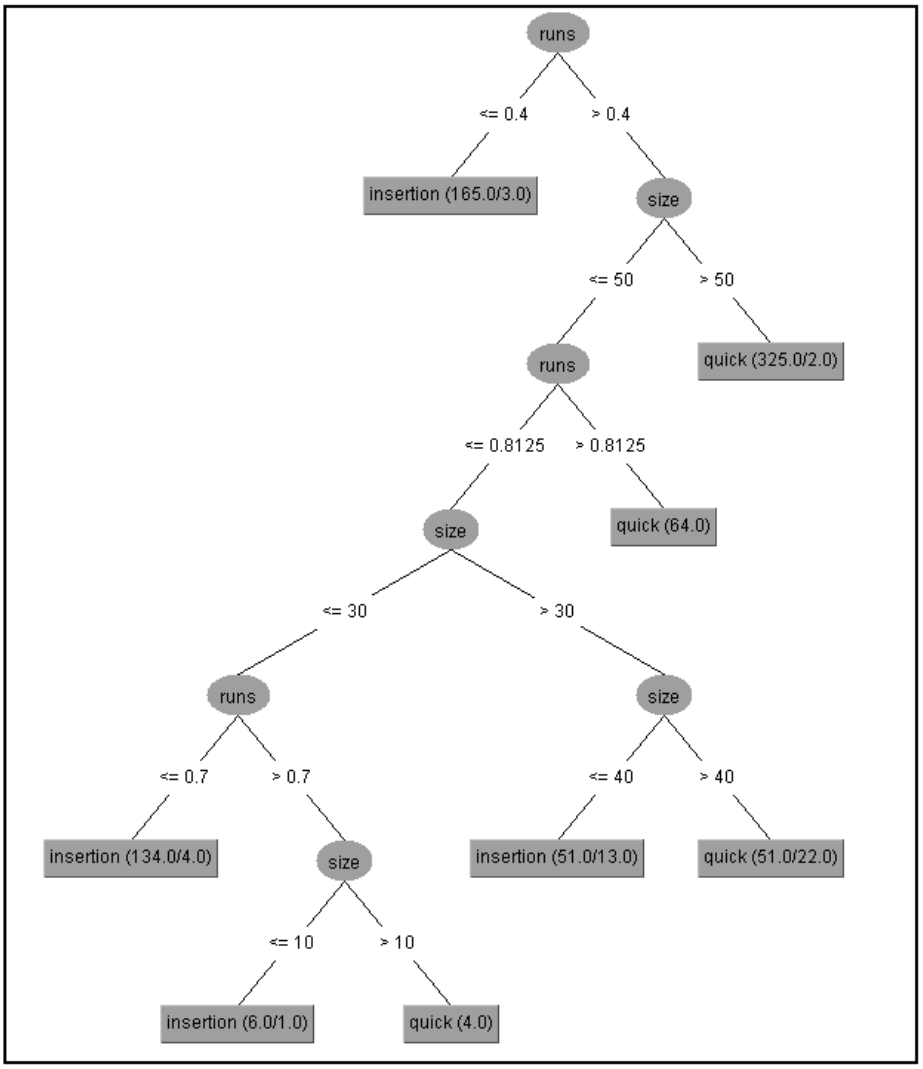

Рис. 4. Дерево решений для сортировки

```
СХЕМА adaptiveSort(A, n)
====
"(runs := "Вычислить степень неотсортированности массива (A) длины (n)")"
ЕСЛИ ‘runs <= 0.4’ ТО "insertionSort(A, n)"
ИНАЧЕ
   ЕСЛИ ‘runs > 0.4’ ТО
     ЕСЛИ ‘size <= 50’ ТО
        ЕСЛИ ‘runs <= 0.8125’ ТО
           ЕСЛИ ‘size <= 30’ ТО
              ЕСЛИ ‘runs <= 0.7’ ТО "insertionSort(A, n)"
              ИНАЧЕ ЕСЛИ ‘runs > 0.7’ ТО
                       ЕСЛИ ‘size <= 10’ ТО "insertionSort(A, n)"
                       ИНАЧЕ ЕСЛИ ‘size > 10’ ТО "quickSort(A, 0, n-1)"
                             КОНЕЦ ЕСЛИ
                       КОНЕЦ ЕСЛИ
                 КОНЕЦ ЕСЛИ
              КОНЕЦ ЕСЛИ
           ИНАЧЕ ЕСЛИ ‘size > 30’ ТО
                    ЕСЛИ ‘size <= 40’ ТО "insertionSort(A, n)"
                    ИНАЧЕ ЕСЛИ ‘size > 40’ ТО "quickSort(A, 0, n-1)"
                          КОНЕЦ ЕСЛИ
                    КОНЕЦ ЕСЛИ
                КОНЕЦ ЕСЛИ
           КОНЕЦ ЕСЛИ
```

```
        ИНАЧЕ ЕСЛИ 'runs > 0.8125' ТО "quickSort(A, 0, n-1)"
              КОНЕЦ ЕСЛИ
        КОНЕЦ ЕСЛИ
      ИНАЧЕ ЕСЛИ 'size > 50' ТО "quickSort(A, 0, n-1)"
            КОНЕЦ ЕСЛИ
      КОНЕЦ ЕСЛИ
    КОНЕЦ ЕСЛИ
КОНЕЦ ЕСЛИ
КОНЕЦ СХЕМЫ adaptiveSort(A, n)
```

Для проверки эффективности полученного адаптивного алгоритма был подготовлен тестовый набор из 140 массивов трех вышеупомянутых видов. Эксперимент производился на Intel Core 2 Quad CPU, 2.51 ГГц; ОС Windows XP. На рис. 5 показано суммарное время выполнения на всех массивах каждого из 5 алгоритмов сортировки и адаптивного алгоритма в микросекундах. Время выполнения `adaptiveSort(A, n)` является наименьшим, что свидетельствует об эффективности предлагаемого подхода.

Заключение

Проведен эксперимент по разработке адаптивной программы сортировки на основе использования метода выбора алгоритмов, системы машинного обучения и алгеброалгоритмического подхода. Средства машинного обучения позволяют автоматизировать построение адаптивного алгоритма на основе анализа экспериментальных данных, связанных с выполнением исходных алгоритмов, имеющихся в распоряжении разработчика. Для проектирования алгоритмов использован язык САА-схем, преимуществом которого являются простота в обучении и использовании, а также независимость от языка программирования и возможность переведения в произвольный целевой язык. Проведенный вычислительный эксперимент показал большую производительность разработанного адаптивного алгоритма в сравнении с исходными, что свидетельствует об эффективности разработанного подхода.

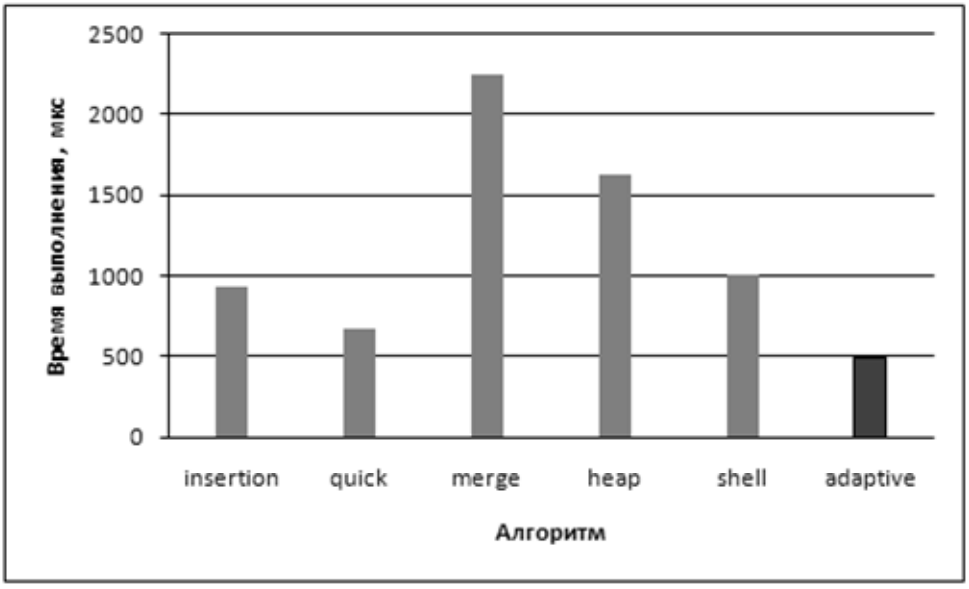

Рис. 5. Суммарное время выполнения алгоритмов сортировки
на тестовом наборе из 140 массивов
(длина каждого массива – 100 элементов)

Перспективами дальнейших исследований в данном направлении являются интеграция системы ИПС и системы Weka, а также использование предложенного подхода для других предметных областей.

1. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.* Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академпериодика, 2007. – 631 с.
2. *Дорошенко А.Е., Жереб К.А., Яценко Е.А.* Об оценке сложности и координации вычислений в многопоточных программах // Проблеми програмування. – 2007. – № 2. – С. 41–55.
3. *Дорошенко А.Е., Яценко Е.А.* Средства сервисно-ориентированного программирования параллельных программ // Проблеми програмування. – 2009. – № 2. – С. 12–21.
4. *Дорошенко А.Е., Шевченко Р.С.* Система символьных вычислений для программирования динамических приложений // Проблемы программирования. – 2005. – № 4. – С. 718–727.
5. *Степанов Р.Г.* Технология Data Mining: Интеллектуальный анализ данных. – 2008. – http://m8.ksu.ru/EOS/dm.pdf
6. *Smith-Miles K.A.* Cross-Disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv., 41, 1, Article 6 (December 2008). – 2008. – 25 p.
7. *Wang X., Smith K.A., Hyndman R.* Characteristic-Based clustering for time series data. Data Mining Knowl. Discov. 13. – 2006. – P. 335–364.
8. *Samulowitz H., Memisevic R.* Learning to solve QBF. In Proceedings of the 22nd AAAI Conference on Artificial Intelligence. – 2007. – P. 255–260.
9. *Smith-Miles K.A.* Towards insightful algorithm selection for optimisation using meta-learning concepts. In Proceedings of the IEEE Joint Conference on Neural Networks. – 2008. – P. 4118–4124.
10. *Lagoudakis M., Littman M., Parr R.* Selecting the right algorithm. In Proceedings of the AAAI Fall Symposium Series: Using Uncertainty within Computation. – 2001.
11. *Guo H.* Algorithm selection for sorting and probabilistic inference: A machine learning-based approach. Ph.D. dissertation, Kansas State University. – 2003.
12. *Weka* Project home page. – http://www.cs.waikato.ac.nz/ml/weka/
13. *Деревья* решений – общие принципы работы. – http://www.basegroup.ru/library/analysis/tree/description/
14. *Кнут Д.* Искусство программирования для ЭВМ. – М.: Мир, 1978. – Т. 3. – 843 с.
15. *Durstenfeld R.* Algorithm 235: Random permutation. Communications of the Association for Computing Machinery, 7:420. – 1964.
16. *Slightly* Skeptical View on Sorting Algorithms. – http://www.softpanorama.org/Algorithms/sorting.shtml

Получено 16.03.2011

***Об авторе*:**

Яценко Елена Анатольевна,
кандидат физико-математических наук,
старший научный сотрудник.

***Место работы автора*:**

Институт программных систем
НАН Украины.
Тел.: (044) 424 4972,
e-mail: oayat@ukr.net