УДК 004.051:004.89: 519.712.2

В.И. ШИНКАРЕНКО

СТРУКТУРНАЯ АДАПТАЦИЯ АЛГОРИТМОВ НА ОСНОВЕ ПОЛИМОРФИЗМА

Abstract: *The technique of dynamic structural adaptation of algorithms is offered. It consists of such techniques: construction of meta-algorithm, analysis of efficiency of adaptive algorithm running and recommendations to algorithms synthesis. The application of method adaptation of sorting algorithm for with use of special design software is shown. Computational experiments were executed. Method adequacy was validated with results of adaptation in for theoretic researched special cases. Practical value of structural adaptation of algorithms was demonstrated at running with different input data.*
Key words: *algorithm, meta-algorithm, abstract operator, structural adaptation, sorting, computational experiment.*

Анотація: *Запропоновано метод динамічного формування структурно адаптивних алгоритмів. Він включає методи формування метаалгоритму, аналізу ефективності виконання алгоритму, що адаптується, та надання рекомендацій щодо його синтезу. Показано застосування методу для адаптації алгоритмів сортування з використанням розробленого програмного забезпечення. Виконані обчислювальні експерименти. Достовірність методу підтверджена результатами адаптації в часткових, теоретично досліджених, випадках. Показана практична доцільність структурно адаптивних алгоритмів при виконанні на різноманітних потоках початкових даних.*
Ключові слова: *алгоритм, метаалгоритм, абстрактний оператор, структурна адаптація, сортування, обчислювальний експеримент.*

Аннотация: *Предложен метод динамического формирования структурно адаптивных алгоритмов. Он включает методы формирования метаалгоритма, анализа эффективности выполнения адаптируемого алгоритма и выработки рекомендаций для его синтеза. Показано применение метода к адаптации алгоритмов сортировки на основе разработанного программного обеспечения. Выполнены численные эксперименты. Достоверность метода подтверждена результатами адаптации в частных, теоретически изученных, случаях. Показана практическая ценность структурно адаптивных алгоритмов при выполнении на различающихся потоках входных данных.*
Ключевые слова: *алгоритм, метаалгоритм, абстрактный оператор, структурная адаптация, сортировка, численный эксперимент.*

1. Введение

В отличие от материальной продукции один и тот же экземпляр программного продукта может эксплуатироваться в различных местах и при различных условиях функционирования. В процессе эксплуатации условия функционирования могут существенно или незначительно изменяться, что часто влечет необходимость соответствующих изменений программного обеспечения (ПО).

Для облегчения сопровождения ПО в подобных случаях применяются специальные технологии. Одно из таких направлений представляет адаптация ПО. Достоинством такого подхода является исключение вмешательства человека в процесс эксплуатации. Широко применяется адаптация интерфейса пользователя. Адаптация алгоритмов менее изучена и гораздо реже применяется в практике программирования.

Под адаптацией понимают приспособление системы к среде функционирования. Адаптация является "процессом целенаправленного изменения параметров и структуры системы" [1].

Под адаптацией алгоритмов будем понимать изменение параметров или структуры алгоритмов с целью оптимизации, сохранения в определенных пределах или улучшения некоторых параметров их качества в конкретных условиях функционирования.

Основными показателями качества или эксплуатационными характеристиками алгоритмов является временная [2] и функциональная [3] эффективность алгоритмов.

Внешней средой функционирования алгоритмов являются:

– исполнительный механизм (технические средства) их выполнения;

– входные данные для конкретного выполнения алгоритма;

– потоки данных для многократного выполнения алгоритма в некоторых (технологических) условиях их использования.

Известные отдельные случаи адаптации являются специфическими, применимыми к конкретным классам алгоритмов. Например, параметрическая адаптация алгоритмов сжатия данных [4] учитывает статистические показатели поступающих на вход данных. Так как параметры алгоритмов могут изменяться в процессе выполнения, то адаптация является динамической.

Основной особенностью структурной адаптации алгоритмов является их способность к самоизменению с целью повышения эффективности. Характерным примером является адаптация на основе алгебр Глушкова [5 – 7]. Она требует предварительных алгебраических исследований конкретных алгоритмов и является статической. Таким образом были получены адаптивные алгоритмы маятниковой сортировки [6].

Статической структурной адаптацией можно считать специализацию алгоритмов. Структура универсального алгоритма модифицируется с целью более эффективного применения в частных специфических случаях.

Задача изменения структуры алгоритма в процессе выполнения является чрезвычайно сложной ввиду того, что требует преобразований машинных кодов в условиях динамически изменяемой адресации.

В данной работе рассматривается возможность динамической структурной адаптации алгоритмов в процессе эксплуатации (многократного выполнения). Структура алгоритмов изменяется в промежутках между выполнениями либо предварительно перед выполнением. Адаптируется алгоритм к последовательности исходных данных, поочередно поступающих ему на вход. Критерием качества принята временная эффективность.

2. Формирование структурно адаптивного алгоритма

При формировании адаптивного алгоритма необходимо решить следующие задачи и иметь соответствующие программные средства для:

– изменения структуры либо синтеза адаптивного алгоритма на основании знаний об эффективности его составляющих;

– измерения эксплуатационных характеристик алгоритма;

– анализа эксплуатационных характеристик в результате его выполнения, извлечения знаний об эффективности составляющих адаптивного алгоритма и выработки рекомендаций для изменения его структуры или синтеза.

Можно выделить две части адаптивного алгоритма: адаптирующую и адаптируемую. Адаптируемая часть отвечает за требуемую функциональность в процессе эксплуатации. Функциональное назначение адаптирующей части – изменение структуры адаптируемой части с целью повышения ее эффективности в конкретных условиях эксплуатации.

Для апробации представленного далее метода формирования структурно адаптивного алгоритма применительно к алгоритмам сортировки разработано соответствующее программное

обеспечение. Выбор алгоритмов сортировки для апробации предлагаемого метода связан с тем, что они достаточно хорошо изучены теоретически и практически.

В нашем случае структура алгоритма не изменяется при каждом выполнении алгоритма, а синтезируется новый алгоритм.

Адаптирующий алгоритм состоит из ряда синтаксически и семантически независимых частей: синтезатора, транслятора, измерительной системы и анализатора (рис. 1).

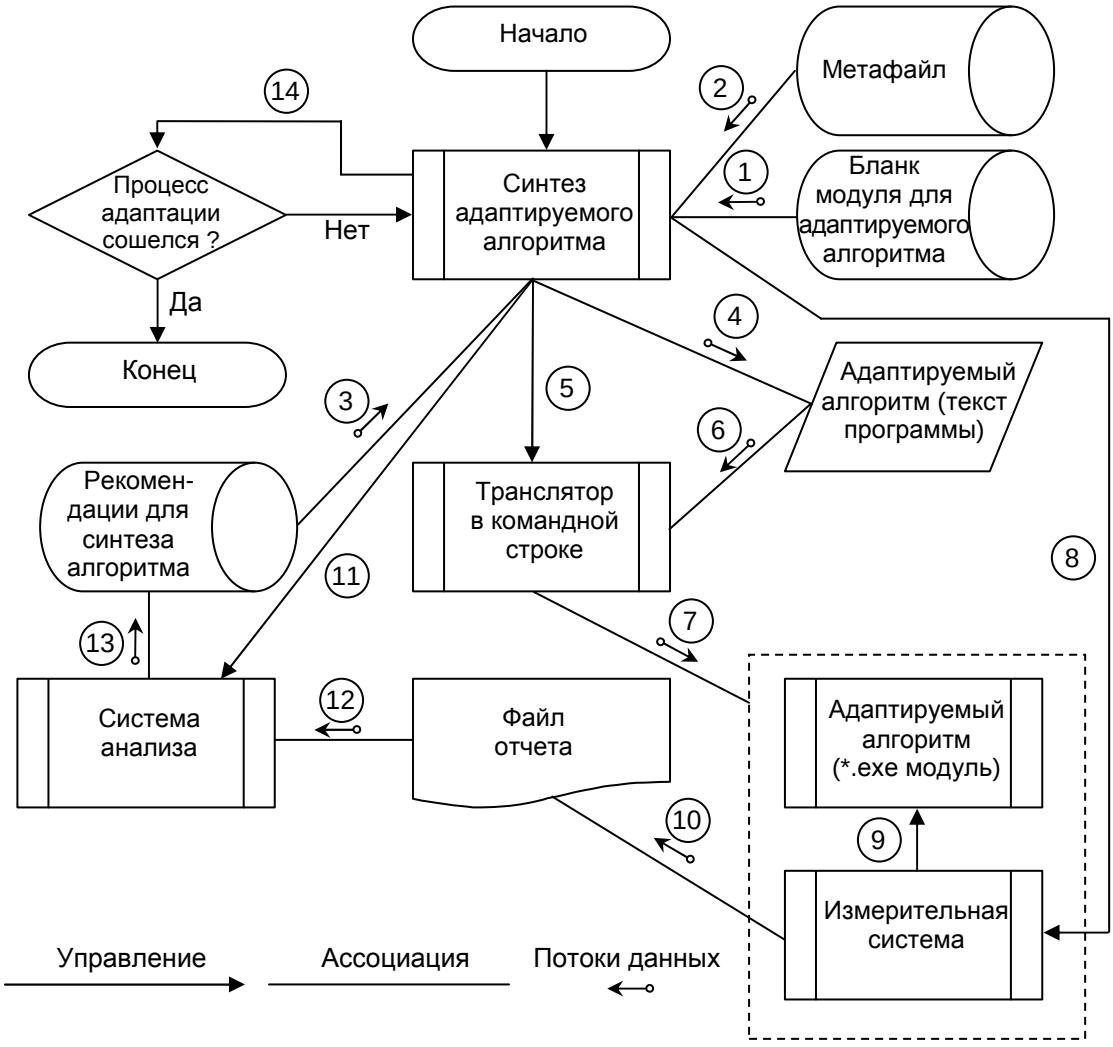

Рис. 1. Схема формирования адаптируемого алгоритма

Текст адаптируемого алгоритма синтезируется на основе так называемого метафайла и рекомендаций системы анализа, транслируется и под управлением измерительной системы выполняется. Такая последовательность продолжается до тех пор, пока стабильно будет синтезироваться одинаковый алгоритм. Считаем, что процесс адаптации сходится к некоторому алгоритму, если 7 из 10 последних синтезированных алгоритмов совпадают.

Далее рассмотрим особенности всех составляющих адаптирующего алгоритма.

3. Метаалгоритм и его формирование

Метаалгоритм – специальным образом заданный алгоритм, на основе которого могут быть

построены конкретные алгоритмы, обобщенный алгоритм решения некоторой задачи.

Рассмотрим методику формирования метаалгоритма.

Одним из ключевых методов проектирования и разработки алгоритмов (и программ) является метод пошаговой детализации, предложенный Н. Виртом [8, 9]. Из множества интерпретаций и средств представления метода [8–11] отметим наиболее лаконичные и целостные в [8].

Начальный (нулевой) уровень реализации алгоритма представлен его спецификацией, которая считается начальным абстрактным оператором (АО). Далее он детализируется с помощью других абстрактных операторов, абстрактных предикатов (далее просто АО, имея в виду и то, и другое) и исполнительных конструкций (операторов) языка программирования. На каждом шаге детализации выполняется реализация АО предыдущего уровня до тех пор, пока алгоритм не будет разработан и представлен полностью на основе конструкций некоторого языка программирования.

В целях адаптации алгоритмов этот метод модифицируем следующим образом. При детализации АО используем полиморфизм, то есть возможность для абстрактного оператора принимать различные формы. АО может иметь несколько вариантов детализации, отличающихся друг от друга по существу. При раскрытии АО представляются все известные варианты его выполнения.

Порядок детализации при построении метаалгоритма сортировки представлен на рис. 2.

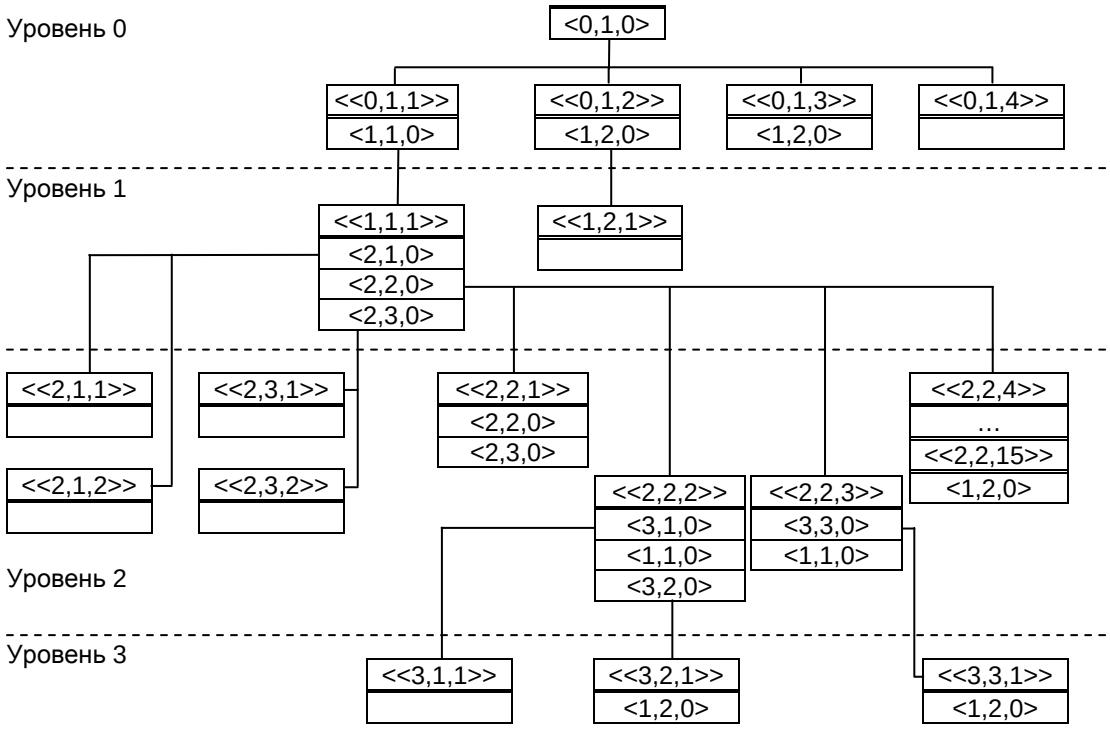

Рис. 2. Порядок детализации абстрактных операторов метаалгоритма сортировки

Здесь приняты следующие обозначения АО: $O_{i,j,k}$ – k-й вариант j-го абстрактного оператора i-го уровня детализации. Если $k = 0$, то это описание (спецификация, идентификация) абстрактного оператора, при $k > 0$ – реализация. На рис. 2 упрощенные обозначения $< i, j, k >$

для описания и $<< i, j, k >>$ для реализации АО.

Начальным является абстрактный оператор $O_{0,1,0}$: <<сортировка массива>>. Он имеет 4 варианта реализации:

– $O_{0,1,1}$: <<сортировка с вариантами>>;

– $O_{0,1,2}$: <<пирамидальная сортировка>>;

– $O_{0,1,3}$: <<быстрая сортировка>>;

– $O_{0,1,4}$: <<сортировка Шелла>> [12].

АО $O_{0,1,2}$ и $O_{0,1,3}$ используют АО $O_{1,2,0}$: <перестановка элементов массива>.

АО $O_{0,1,1}$ реализуется с использованием $O_{1,1,0}$ <варианты сортировки>, который, в свою очередь, реализуется с использованием следующих АО:

– $O_{2,1,0}$: <установить направление прохода>;

– $O_{2,2,0}$: <выполнить проход сортировки>;

– $O_{2,3,0}$: <изменить направление прохода>.

АО $O_{2,1,0}$ имеет две реализации с установкой прямого и обратного направлений прохода, а $O_{2,3,0}$ – с инверсией направления прохода и без. Реализация АО $O_{2,2,0}$ на втором уровне имеет несколько альтернативных реализаций:

– $O_{2,2,1}$: <<два прохода вместо одного>>. Используя АО $O_{2,3,0}$ и дважды $O_{2,2,0}$, обеспечивается возможность применения разных методов сортировки при последовательных проходах;

– $O_{2,2,2}$: <<проход разбиением>>. С помощью АО $O_{3,1,0}$ массив разбивается на две равные части, их начало и конец заносятся в стек, обе части сортируются с помощью АО $O_{1,1,0}$ и объединяются АО $O_{3,2,0}$: <слияние отсортированных массивов>;

– $O_{2,2,3}$: <<проход разбиением на два взаимно отсортированных массива>>. АО $O_{3,3,0}$ массив, как в быстрой сортировке, разбивается на две части. В первой части все элементы меньше любого элемента из второй части. Каждая часть массива сортируется АО $O_{1,1,0}$;

– $O_{2,2,4}$ … $O_{2,2,15}$ – реализуют сортировки "пузырьком", выбором, вставками, бинарными вставками с прямым, обратным и двусторонним направлениями проходов в следующем порядке: сначала двусторонние, затем парами с прямым и обратным направлением прохода. Они используют АО $O_{1,2,0}$.

Отметим, что детализация АО может содержать АО не только следующего уровня, но и предыдущих, даже тех, в раскрытии которых участвует он сам, т.е. допускается рекурсия.

Метаалгоритм является основой для формирования конкретных алгоритмов. Важно, чтобы все конкретные алгоритмы были корректными. Различные сочетания альтернатив различных АО не должны иметь синтаксических и семантических противоречий.

Для удовлетворения этого требования предусмотрены следующие средства:

– аппарат формальных и фактических параметров абстрактных операторов;

– препроцессорные выражения;

– дополнительные знания в виде правил, описывающих ограничения при формировании адаптируемого алгоритма.

Рассмотрим эти средства на примерах АО (рис. 3), т.к. весь метаалгоритм достаточно большой – порядка 700 строк кода.

```
{/0.1.1/ Сортировка (#a,#b)}
 {%%q=1}
 {%%n=1}
 Stack1[1]:=#a; {стек неотсортированных}
 Stack1[2]:=#b; {частей массива}
 IuStack1:=1;    {указатель стека}
 IuStack2:=-2;   {указатель стека объединяемых
                  частей массива}
 {/1.1/ Сортировка массива ()}
{/0.1.1/}

{/1.1.1/ Сортировка массива (mas)}
   q%q:=1; массив не отсортирован}
   if  IuStack1>0 then
    begin
     {/2.1/ Установить направление прохода (n%n)}
     while (q%q=1) do
      begin
       {/2.2/ Проход сортировки массива (q%q)}
         if  q%q=0  then
           begin
              IuStack1:= IuStack1-2;
              break;
           end;
       {/2.3/ Изменить направление прохода (n%n)}
      end;
    end;
  {/1.1.1/}

   {/2.2.1/ Проход сортировки массива (#c)}
         {с несколькими разными повторами}
     {/2.2/ Проход сортировки массива (q%q)}
     {/2.3/ Изменить направление прохода (n%n)}
     {/2.2/ Проход сортировки массива (q%q)}
     {/2.3/ Изменить направление прохода (n%n)}
  {/2.2.1/}
```

```
{/2.2.2/ Проход сортировки массива (#c)}
         {разбиением}
    {/3.1/ Разбить массив на два ()}
    {%%q++}
    {%%n++}
    {/1.1/ Сортировка массива ()}
    {/1.1/ Сортировка массива ()}
    {/3.2/ слияние отсортированных массивов ()}
    {%%q--}
    {%%n--}
    break;
  {/2.2.2/}

{/2.2.8/ Проход сортировки массива (#c)}
         {пузырьком с прямым проходом}
   IIb:=Stack1[IuStack1];
   IIe:=Stack1[IuStack1+1];
   if  IIb<IIe then
    begin
     i:=IIb;
     for j:=IIb to IIe-1 do
      if mas[j]>mas[j+1] then
       begin
        {/1.2/ Переставить элементы (j,j+1)}
        i:=j;
       end;
     IIe:=i;
     Stack1[IuStack1+1]:=IIe;
    end;
   if IIb>=IIe then   #c:=0; ;
  {/2.2.8/}
```

Рис. 3. Примеры реализации абстрактных операторов

Основой метаалгоритма являются несколько альтернативных функционально эквивалентных алгоритмов. Как оказалось, возможностей самих этих алгоритмов для построения метаалгоритма недостаточно. Дополнения алгоритмов сортировки, на которых основан

метаалгоритм, связаны с тем, что каждый алгоритм выполняет сортировку массива в целом, в то время как в метаалгоритм заложены возможности разбиения его на части и сортировки отдельных частей. Для реализации такой возможности в данном случае использован стек индексов неотсортированных частей массива (см. $O_{0,1,1}$), и в АО он отслеживается.

Синтаксис препроцессорных выражений аналогичен средствам языка программирования PL/1 [13], но значительно упрощен. Препроцессорные выражения и переменные начинаются с символа "%". В данном примере препроцессорная переменная %q принимает значения 1, 2, …, модифицируя идентификаторы $q1$, $q2$, …, которые используются в качестве признака: сортируемая часть массива отсортирована? (Окончание сортировки). Ввиду возможности рекурсивной вложенности АО $O_{2,2,0}$ (в т.ч. косвенной) препроцессорная переменная %q обеспечивает уникальность идентификаторов признаков. Для обеспечения совместимости с транслятором ЯП Паскаль препроцессорные выражения, как и идентификаторы описания и использования АО, находятся в комментариях.

Аппарат формальных и фактических параметров АО расширяет их возможности до уровня процедур. Для упрощения реализации в нашем случае формальные параметры начинаются с символа "#". Аппарат параметров применен для передачи признака окончания сортировки, направления прохода и в АО $O_{1,2,0}$ – индексов переставляемых элементов.

Связанные с метаалгоритмом правила синтеза конкретных алгоритмов ограничивают возможности использования АО. В предварительных и описанных далее численных экспериментах применялись следующие правила:

– "m" – ограничивает количество включений АО в синтезируемый алгоритм. $m(O_{2,2,1}) = 6$, $m(O_{2,2,2}) = 30$, $m(O_{2,2,3}) = 30$;

– "f" – ограничивает количество включений АО в себя, в том числе и косвенно. $f(O_{2,2,1}) = 5$, $f(O_{2,2,2}) = f(O_{2,2,3}) = N \log_2 N$ (N – количество элементов в массиве);

– "w" – запрещает вложенность АО в заданное. $w(O_{2,2,1}) = (O_{2,2,2}, O_{2,2,3})$, $w(O_{2,2,1}) = (O_{2,2,6}, O_{2,2,7}, O_{2,2,12}, O_{2,2,13}, O_{2,2,14}, O_{2,2,15})$ – последнее запрещает сочетать сортировку вставками с другими методами при нескольких проходах;

– "p" – ограничивает сверху рекомендуемую вероятность использования АО. $p(O_{2,2,1}) = 0{,}9$.

Таким образом, метаалгоритм учитывает возможности основных известных методов сортировок, с изменением и без направления прохода (в том числе и маятниковых) и применения различных их комбинаций.

4. Синтез адаптивного алгоритма

Синтез адаптируемого алгоритма выполняется следующим образом.

Имеется размеченный бланк алгоритма, в котором задана корректная структура модуля и

указаны места для вставки описания переменных и тела адаптируемого алгоритма.

В указанное место бланка вставляется начальный АО $O_{0,1,0}$. Он заменяется реализацией. Далее выполняется построчная обработка до последней записи.

Инструкции (операторы) ЯП пропускаются. Препроцессорные выражения транслируются и выполняются.

Если в текущей строке содержится описание АО, то оно заменяется реализацией, формальные параметры заменяются фактическими. Выполняется проверка возможности дальнейшего использования АО, связанная с ограничениями согласно правилам формирования алгоритма. Если, например, достигнут максимально допустимый уровень вложенности АО согласно правилу f, оператор помечается как недоступный, когда вложенность снижается, пометка снимается.

В случае, если альтернативных реализаций АО несколько, случайным образом выбирается одна из них на основе рекомендуемых вероятностей для каждой реализации (согласно функции распределения для дискретной случайной величины).

Для идентификации адаптируемого алгоритма запоминается последовательность замены описаний АО их реализациями. Конкретный адаптивный алгоритм однозначно определяется метафайлом и последовательностью выбора реализаций абстрактных операторов:

$$A \mid A_M = O_{i_1,j_1,k_1} \to O_{i_2,j_2,k_2} \to \ldots \to O_{i_n,j_n,k_n}.$$

Назовем такое выражение идентификационной последовательностью адаптируемого алгоритма.

5. Измерительная система

Измерительная система предназначена для запуска адаптируемого алгоритма, измерения времени его выполнения и формирования протокола выполнения в файле отчета.

Для измерения времени выполнения алгоритма можно воспользоваться средствами операционной системы и ЭВМ.

Время измерялось с помощью регистра-счетчика тактов [14]. Измерялось “грязное” время, т.е. время от момента начала до момента окончания выполнения алгоритма, включая и время, получаемое в этот промежуток другими процессами и потоками. Параллельно с адаптируемым алгоритмом выполнялось порядка 400 потоков ОС и различных приложений. Это в некоторой степени зашумляет измерения. Однако, с учетом того, что адаптируемый алгоритм предназначен для работы в той же среде, в которой выполняется и адаптация, такой подход к измерению вполне оправдан и не требует создания специальных условий для измерений, исключающих шумы.

Вопрос, насколько сильно на процесс адаптации и его результат будут влиять процессы, в значительной степени конкурирующие с адаптируемым алгоритмом за разделяемые ресурсы, требует специального изучения. Априори представляется, что если процессы окружения близки к стационарным, то они могут лишь увеличить продолжительность адаптации, незначительно влияя на конечный результат.

Если время выполнения алгоритмов на несколько порядков (минимум на два) превышает длительность кванта времени, выделяемого диспетчером ОС, для измерения времени можно воспользоваться API функцией Windows GetProcessTimes. Она позволяет измерять общее количество тактов, полученное процессом в режиме ядра и режиме пользователя, т.е. “чистое” время выполнения. Однако следует учитывать, что это значение времени обновляется достаточно редко (каждые 1/64 с), что значительно повышает погрешность измерения.

6. Система анализа

При выборе метода построения конкретных алгоритмов сортировки определимся сначала с размерностью задачи – числом возможных алгоритмов сортировки.

Утверждение. Множество возможных алгоритмов сортировки является счетным множеством.

Доказательство. Воспользуемся методом математической индукции. При размере сортируемого массива $N_0 = 3$ алгоритмы сортировки пузырьком, простыми вставками и выбором уже различаются. То есть при $N_0 = 3$ количество различных сортировок M_0 не менее трех, $M_0 \geq N_0$.

Пусть при размере массива N_i имеется M_i методов (алгоритмов) сортировки и $M_i \geq N_i$. При размере массива $N_{i+1} = 2N_i$ его можно разбить на две части, каждую отсортировать одним из M_i методов и затем объединить. Таким образом, можно получить новые гибридные методы. Это даже без учета возможности комбинировать проходы различными методами и изменения направления прохода.

Из того, что $M_{i+1} = M_i^2$ и $M_i \geq N_i$, при $N_i \geq 3$ следует, что $M_{i+1} \geq N_i^2$, а так как $N_{i+1} = 2N_i$, то $M_{i+1} \geq N_{i+1}$ при $N_i = 3 \cdot 2^i$.

Таким образом, для любого наперед заданного количества различных сортировок можно найти такой размер массива, что количество различных сортировок будет больше.

Что и доказывает утверждение.

Верхняя граница количества различных алгоритмов сортировки ограничена количеством различных перестановок из числа элементов массива $N!$. Считаем, что один и тот же порядок элементов в массиве в процессе сортировки не должен повторяться (иначе алгоритм между повторами выполняет ненужные действия и является избыточным).

Можно с большой долей уверенности предположить, что реальное количество алгоритмов сортировки близко к верхней границе. Однако нет гарантии, что все они могут быть реализованы на основе конкретного метаалгоритма.

Формально задачу нахождения адаптируемого алгоритма можно сформулировать следующим образом. Необходимо найти алгоритм

$$A : t(A) = \min_{B \in \Omega} t(B), \quad (1)$$

где Ω – множество алгоритмов сортировки, которые могут быть построены на основе заданного метаалгоритма; $t(A)$ – среднее время выполнения алгоритма A на заданном множестве входных данных (применительно к сортировке – на множестве возможных входных массивов).

Точное решение задачи (1) возможно только методом перебора. Такой метод является неприемлемым из-за большого количества возможных алгоритмов и входных данных. Для решения задачи применен метод направленного случайного поиска. Разработан гибридный алгоритм на основе максиминного метода кластеризации [15] и аналога метода градиентного спуска.

Управление синтезом алгоритмов заключается в том, что система анализа определяет рекомендуемые вероятности использования каждого варианта реализации всех АО. Генерация адаптируемого алгоритма выполняется согласно этим рекомендациям.

Рассмотрим методику определения рекомендуемых вероятностей.

Рекомендации основываются на допущении, что чем больше используется реализация АО в наиболее эффективном алгоритме, тем она эффективнее и её чаще нужно использовать. Считаем, что такое допущение незначительно сказывается на модели, если уровень вложенности АО не превосходит 2 … 3.

Применяя кластеризацию, рекомендуемые вероятности определяются в три этапа. Первый – предварительная обработка, второй – кластеризация, третий – расчет рекомендуемых вероятностей.

На этапе предварительной обработки файла отчета для каждого $r-$го выполнения адаптируемого алгоритма из отчета определяется время выполнения $t(r)$. 50% худших по времени выполнения алгоритмов отбрасываются. Определяется количество включений в каждый $r-$й выполненный алгоритм каждой $k-$й реализации $j-$го АО $i-$го уровня детализации $Q_{i,j,k,r}$.

Все $Q_{i,j,k,r}$ нормализуются:

$$\overline{Q}_{i,j,k,r} = \frac{Q_{i,j,k,r} - \min\limits_{r} Q_{i,j,k,r}}{\max\limits_{r} Q_{i,j,k,r} - \min\limits_{r} Q_{i,j,k,r}}. \qquad (2)$$

На этапе кластеризации все алгоритмы разбиваются на кластеры – группы, имеющие какие-то свои структурные особенности.

Образ алгоритма в E^n определяется в виде вектора $q(A_r) = [\overline{Q}_{i_1,j_1,k_1,r}, \overline{Q}_{i_2,j_2,k_2,r} \ldots \overline{Q}_{i_n,j_n,k_n,r}]$, где n – количество всех реализаций всех АО в метаалгоритме.

Расстояние между образами алгоритмов A_x и A_y определяется как

$$\rho(A_x, A_y) = \sqrt{\sum_i \sum_j \sum_k (\overline{Q}_{i,j,k,x} - \overline{Q}_{i,j,k,y})^2}. \qquad (3)$$

Ищется множество центров кластеров I. Находятся два максимально удаленных друг от друга образа алгоритмов. Они принимаются за центры двух начальных кластеров. Далее последовательно ищутся точки:

$$q(A_z): \min_{A_i \in I}(\rho(A_i, A_z) = \max_{\forall A_j}(\min_{A_i \in I}(\rho(A_i, A_j)).$$

Если $\min_{A_i \in I}(\rho(A_i, A_z) \geq \frac{1}{2} \sum_{A_i, A_j \in I, i \neq j} \rho(A_i, A_j)$, точка $q(A_z)$ принимается за новый центр кластера, и алгоритм A_z добавляется к множеству I, в противном случае формирование множества I закончено и остальные точки разносятся к кластерам по критерию минимума расстояния.

На третьем этапе рекомендуемые вероятности определяются следующим образом. Пусть F_i – множество алгоритмов, принадлежащих i-му кластеру, $\overset{\vee}{F}$ и $\hat{F}$ – множества алгоритмов, принадлежащих лучшему и худшему по среднему времени выполнения кластеру соответственно; худшее время выполнения алгоритма принадлежит лучшему кластеру:

$$\overset{\vee}{t}_{\max} = \max_{A_r \in \overset{\vee}{F}}(t_r); \tag{4}$$

лучшее время выполнения алгоритмов:

$$t_{\min} = \min_{\forall A_r}(t_r). \tag{5}$$

Основываясь на том, что если алгоритм не хуже худшего из лучшего кластера, его можно считать конкурентоспособным, относительное качество алгоритма определим как

$$l_r = \begin{cases} 1 + \dfrac{\overset{\vee}{t}_{\max} - t_r}{\overset{\vee}{t}_{\max} - t_{\min}}, \text{ если } t_r \leq \overset{\vee}{t}_{\max} \\ 0, \text{ в противном случае} \end{cases}. \tag{6}$$

Относительное качество алгоритмов в кластере *r* определим как

$$P_r = \sum_{A_i \in F_r} l_i. \tag{7}$$

Рекомендуемая вероятность использования кластера как образца для синтеза адаптивного алгоритма:

$$\overline{P}_r = \frac{P_r}{\sum_i P_i}. \tag{8}$$

Если во всех кластерах, кроме лучшего, нет конкурентоспособных алгоритмов, рекомендуемая вероятность лучшего кластера будет равна единице, остальных – нулю.

Рекомендуемая вероятность применения АО $O_{i,j,k}$ при условии выбора z-го кластера будет

$$\overline{P'}_{i,j,k,z} = \frac{P'_{i,j,k,z}}{\sum_k P'_{i,j,k,z}}, \text{ где } P'_{i,j,k,z} = \sum_{A_x \in F_z} (\overline{Q}_{i,j,k,x} \cdot l_x). \quad (9)$$

Если для какого-то АО $O_{i,j,k}$ есть правило синтеза алгоритмов $p(O_{i,j,k}) = g$ и $\overline{P'}_{i,j,k,*} > g$, то $\overline{P'}_{i,j,k,*}$ устанавливается равным g, остальные вероятности пропорционально изменяются.

В результате расчета рекомендуемых вероятностей использования реализаций АО, согласно (8) – (9), структура синтезируемых алгоритмов приближается к структуре алгоритмов в лучших кластерах.

Для ускорения направленного поиска оптимального адаптируемого алгоритма поочередно с (8) – (9) выполняется изменение вероятностей аналогично методу градиентного спуска. Как и в предыдущем случае, выполняются два первых этапа кластеризации и находятся лучший и худший кластеры.

Метод градиентного спуска предполагает изменение вероятности применения реализации АО по направлению наискорейшего изменения оптимизируемой функции. В нашем случае

$$P''_{i,j,k} = P''_{i,j,k} + \Delta P''_{i,j,k}, \quad (10)$$

где

$$\Delta P''_{i,j,k} = \frac{\frac{1}{\check{f}} \sum_{A_r \in \check{F}} \overline{Q}_{i,j,k,r} - \frac{1}{\hat{f}} \sum_{A_r \in \hat{F}} \overline{Q}_{i,j,k,r}}{\max_r \overline{Q}_{i,j,k,r}}, \quad (11)$$

где $\check{f}$ и $\hat{f}$ – количество алгоритмов в лучшем и худшем кластерах соответственно.

В случае, если какие-то $P''_{i,j,k}$ оказываются отрицательными, они обнуляются. Затем вероятности нормализуются:

$$\overline{P''}_{i,j,k} = \frac{P''_{i,j,k}}{\sum_k P''_{i,j,k}}. \quad (12)$$

Как и ранее, учитываются правила ограничения вероятностей использования АО.

7. Численные эксперименты

Выполнение численных экспериментов преследовало две цели. Во-первых, проверить, насколько хорошо адаптивный алгоритм может быть адаптирован в теоретически изученных известных частных случаях. Во-вторых, представляет интерес степень превосходства (положительная или отрицательная) адаптивного алгоритма над признанным лучшим алгоритмом в каких-то частных случаях, требующих дополнительных теоретических исследований или когда такие исследования сильно затруднены.

Для достижения первой цели выполнена адаптация алгоритма сортировки к следующим входным данным:

– массив заполнен случайными целыми числами с очень низкой вероятностью повторения;

– предварительно отсортированный массив;

– массив, предварительно отсортированный в обратном порядке.

Выполнялась сортировка массива из 10000 целых 4-байтовых чисел. Каждый эксперимент выполнен 9 раз. Во всех случаях процесс адаптации сошелся к одному алгоритму, но не обязательно одинаковому при параллельных (повторных) расчетах.

Результаты и показатели сходимости приведены в табл. 1. Представленные численные эксперименты выполнялись на компьютере с процессором AMD Duron XP, чипсетом системной платы VIA VT8375 ProSavage DDR KM266, кэшем L1 кода, L1 данных и L2 по 64Кб, тактовой частотой 1400 (5.25 x 267) МГц ,частотой системной шины 133 МГц, частотой памяти 1200 МГц под управлением ОС Windows XP.

Начальное состояние массивов из Ω_4 заключается в том, что первая половина массива заполнена равномерно возрастающей последовательностью целых чисел, а вторая – равномерно убывающей, Ω_5 – наоборот, сначала убывающей, затем возрастающей. Рассортировка на *S%* достигается тем, что в отсортированном массиве S% случайно выбранных элементов менялись местами.

Таблица 1. Результаты численных экспериментов

Особенности сортируемого массива (множество входных данных)	Количество итераций (общее количество алгоритмов)	Количество разных алгоритмов	Время выполнения адаптивного алгоритма, 10^7 тактов	Сравнительное время сортировки, 10^7 тактов	
				"пузырьком"	быстрой сортировкой
Заполнен случайными числами (Ω_1)	157 … 159	91 … 104	4,2 …5,3	1229 … 1320	4,2 … 4,9
Массив отсортирован (Ω_2)	344 … 372	132 … 157	0,20 …0,22	0,15 … 0,20	2,0 … 3,1
Отсортирован в обратном порядке (Ω_3)	161 … 455	90 … 149	0,48 … 1,2	1380 … 1685	2,0 … 2,4
Отсортирован к середине массива (Ω_4)	158 … 175	88 … 110	5,08 … 5,97	1018 … 1113	282 … 290
Отсортирован к концам массива (Ω_5)	157 … 159	91 … 108	4.9 … 5,7	854 … 943	17,1 … 18,4
Рассортирован на 0,1% (Ω_6)	210 … 359	98 … 167	0,71 … 2,26	407 … 669	1,9 … 2,0
Рассортирован на 0,2% (Ω_7)	158 … 458	93 … 186	0,92 … 2,10	557 … 719	1,9 … 2,3

В результате адаптации получены следующие адаптируемые алгоритмы (приводим их идентификационные последовательности на заданных множествах):

– на множестве Ω_1 – во всех параллельных опытах $A \mid A_M = O_{0,1,3}$. Синтезированный адаптируемый алгоритм – алгоритм быстрой сортировки;

– на Ω_2 – также во всех случаях $A \mid A_M = O_{0,1,1} \to O_{1,1,1} \to O_{2,1,2} \to O_{2,2,8} \to O_{1,2,1} \to O_{2,3,1}$. Процесс адаптации сошелся к алгоритму сортировки "пузырьком" с прямым проходом;

– на Ω_3 – при параллельных опытах процесс адаптации сходился к различным адаптивным алгоритмам с длиной идентификационной последовательности 29 … 91 и не сошелся к сортировке "пузырьком" с обратным проходом. Однако и найденные алгоритмы по времени значительно превзошли быструю сортировку. Один из синтезированных алгоритмов имеет следующую идентификационную последовательность:

$$A \mid A_M = O_{0,1,1} \to O_{1,1,1} \to O_{2,1,1} \to O_{2,2,3} \to O_{3,3,1} \to$$
$$O_{1,1,1} \to O_{2,1,2} \to O_{2,2,9} \to O_{1,2,1} \to O_{2,3,1} \to O_{1,1,1} \to O_{2,1,2}(\to O_{2,2,1} \to O_{2,2,4}(\to O_{1,2,1})^2 \to O_{2,3,1})^2 \to$$
$$O_{2,2,9} \to O_{1,2,1}(\to O_{2,3,1})^4 .$$

Массив разбивается на две взаимно отсортированные части, первая сортируется "пузырьком" с обратным проходом, а вторая – чередуя два прохода "пузырьком" двусторонним и один с обратным проходом.

– на Ω_4 и Ω_5 – во всех параллельных опытах $A \mid A_M = O_{0,1,2}$. Построенный адаптируемый алгоритм – алгоритм пирамидальной сортировки;

– на Ω_6 и Ω_7 – получены различные адаптивные алгоритмы, в том числе $A \mid A_M = O_{0,1,3}$, $A \mid A_M = O_{0,1,1} \to O_{1,1,1} \to O_{2,1,2} \to O_{2,2,13} \to O_{1,2,1} \to O_{2,3,1}$, реализующие алгоритмы быстрой сортировки и сортировки вставками с обратным проходом, а также алгоритмы гораздо более сложной структуры. При увеличении степени рассортировки даже до 0,4 … 0,5% процесс адаптации постоянно сходится к алгоритму быстрой сортировки.

Довольно актуальным является вопрос, насколько быстро сходится процесс адаптации? Задача его оптимизации в данной работе не ставилась. Однако полученные результаты показывают достаточно быструю сходимость (рис. 4, для сравнения времени сортировки показаны линии тренда для сортировки "пузырьком" и быстрой сортировки).

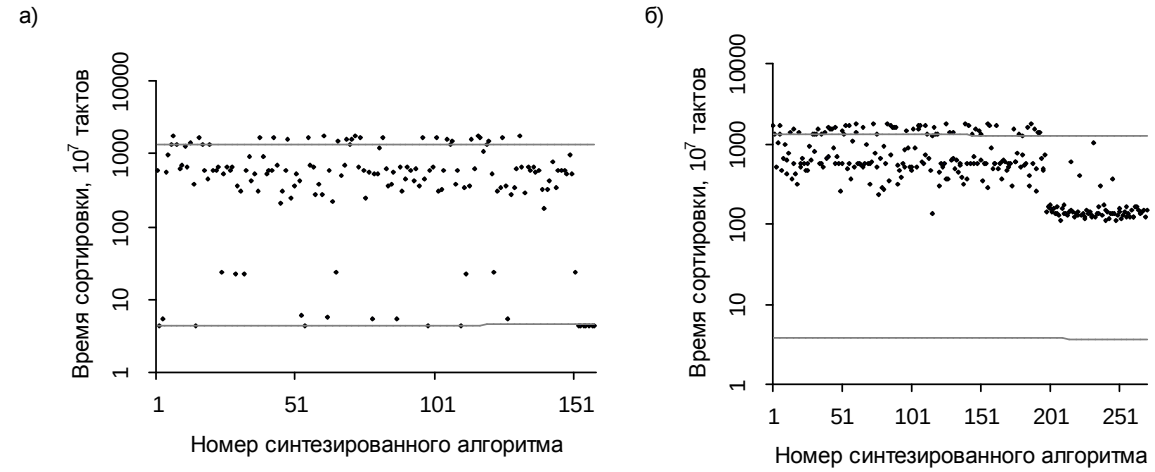

Рис. 4. Демонстрация сходимости процесса адаптации

Для обеспечения приемлемого времени сходимости процесса адаптации необходимо, чтобы на начальном этапе все реализации АО могли проявить свою эффективность, т.е. должно быть достаточно большое количество выполнений синтезированных адаптируемых алгоритмов с равномерным распределением вероятностей использования реализаций АО.

В нашем случае синтезировалось 150 алгоритмов с примерно равномерным распределением вероятностей (получить равномерное распределение достаточно сложно ввиду рекурсивной вложенности АО в метаалгоритме).

На следующем этапе, после каждых 50 синтезированных алгоритмов, выполнялся анализ с очередным формированием рекомендуемых вероятностей использования реализаций АО (чередуя применение кластеризации и аналога метода градиентного спуска).

Довольно типичным явлением была сходимость на первых же алгоритмах после первого выполнения анализа в результате кластеризации (рис. 4а).

На рис. 4б процесс адаптации сошелся после трехкратного применения системы анализа и является результатом применения как кластеризации, так и градиентного спуска. Эти результаты соответствуют экспериментам с заполнением массива случайными числами, но с метаалгоритмом без АО $O_{0,1,2}, O_{0,1,3}, O_{0,1,4}$. В принципе в АО $O_{0,1,1}$ заложены возможности быстрой сортировки (для сортировки Шелла нужен префикс массива, поэтому без дополнительных преобразований она не подходит для сортировки части массива, а пирамидальная не может сочетаться с другими методами сортировки). В этом случае алгоритм сходится практически к алгоритму быстрой сортировки, но без рекурсии и с развернутыми циклами. Поэтому время выполнения адаптивного алгоритма, очень близкого к быстрой сортировке, значительно хуже, чем у последней. Это объясняется тем, что синтезированная программа большего размера и значительное время теряются на кэширование команд.

Для обеспечения достоверности результатов выполнения каждого сгенерированного алгоритма сортировки выполнялись проверка правильности сортировки результирующего массива данных и контроль зацикливания (проверкой наличия результатов по истечении контрольного времени). В результате тестирования и отладки все случаи некорректности и зацикливания устранены.

8. Заключение

Адаптивный алгоритм, как и все другие, имеет свою сферу применения. Целесообразность применения структурно адаптивных алгоритмов предопределяется:

– наличием различных условий функционирования;

– недостаточной эффективностью применяемых алгоритмов;

– как правило, отсутствием ярко выраженного "лидера" среди альтернативных алгоритмов;

– некоторым критерием эффективности, приоритетным при эксплуатации алгоритма.

Можно предложить следующие схемы организации процесса адаптации:

– первоначально устанавливается лучший (по теоретическим данным) алгоритм. В процессе эксплуатации входные потоки данных архивируются. Работа адаптирующего алгоритма

выполняется во время понижения или отсутствия активности системы: во время профилактики, в ночное время. По результатам адаптации изменяется адаптируемый алгоритм;

– адаптация выполняется не на основных, а на вспомогательных или резервных вычислительных средствах;

– адаптация выполняется с использованием скрытых резервов. Например, на одном из ядер многоядерного процессора.

Адаптируемый алгоритм может эксплуатироваться постоянно, а может быть периодически переадаптирован. Процедуру адаптации можно повторять либо с фиксированным периодом времени, либо при наличии признаков изменения особенностей входных данных. В последнем случае желательно наличие объективных показателей, характеризующих входные данные и влияющих на эффективность алгоритма.

Рассмотренные частные случаи начального заполнения алгоритмов не являются типичными и в реальных условиях встречаются исключительно редко. Поэтому применение быстрого алгоритма сортировки в подавляющем большинстве случаев является единственно оправданным. Однако не всегда такой алгоритм по степени и области превосходства является значительно лучше других. В таких случаях адаптивный алгоритм может существенно повысить эффективность программного обеспечения. Заметим, что и в первом случае по окончании процесса адаптации адаптивная сортировка по времени выполнения будет не хуже быстрой сортировки.

Отметим также, что при наличии высокой степени вложенности АО в метаалгоритме при анализе адаптируемых алгоритмов для оценки их эффективности следует учитывать не только количество вхождений реализаций АО, но и порядок их следования.

В некоторых случаях повторная адаптация необходима при изменении исполнительных устройств (ПЭВМ). Как ранее показано [16], что с одними и теми же данными на одной ЭВМ некоторый алгоритм по временной эффективности может превосходить другой альтернативный, а на другой ЭВМ уже уступать ему.

Заметим, что при наличии соответствующего транслятора возможен вариант непосредственного выполнения метаалгоритма, при котором случайным образом или целенаправленно выбирается одна из возможных реализаций АО.

СПИСОК ЛИТЕРАТУРЫ

1. Растригин Л.А. Адаптация сложных систем. – Рига: Зинатне, 1981. – 375 с.
2. Шинкаренко В.И. Сравнительный анализ временной эффективности функционально эквивалентных алгоритмов // Проблемы программирования. – 2001. – № 3 – 4. – С. 31 – 39.
3. Шинкаренко В.И. Функциональная эффективность нечетко специфицированных алгоритмов // Проблемы программирования. – 2006. – № 1. – С. 31 – 39.
4. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео / Д. Ватолин, А. Ратушняк, М. Смирнов и др. – М.: Диалог-МИФИ, 2002. – 384 с.
5. Многоуровневое структурное проектирование программ: Теоретические основы, инструментарий / Е.Л. Ющенко, Г.Е. Цейтлин, В.П. Грицай и др. – М.: Финансы и статистика, 1989. – 208 с.
6. Цейтлин Г.Е. Введение в алгоритмику. – К.: Сфера, 1998. – 310 с.
7. Яценко О.А. Розробка інтегрованих алгебро-алгоритмічних моделей: елементи теорії, інструментарій, застосування: Автореф. дис... канд. фіз.-мат. наук / Київський національний ун-т ім. Тараса Шевченка. – К., 2005. – 17 с.
8. Джехани Н. Язык Ада. – М.: Мир, 1988. – 552 с.

9. Wirth N. Program Development by Stepwise Refinement // Communications of the ACM. – 1971. – Vol. 14, N 4. – P. 221 – 227.
10. Лингер Р. и др. Теория и практика структурного программирования / Р. Лингер, Х. Миллс, Б. Уитт. – М.: Мир, 1982. – 406 с.
11. Хьюз Дж., Мичтом Дж. Структурный подход к программированию. – М.: Мир, 1980. – 278с.
12. Вирт Н. Алгоритмы + структуры данных = программа. – М.: Мир, 1985. – 406 с.
13. http://www-306.ibm.com/software/awdtools/pli/plizos/library / Enterprise PL/I for z/OS. Programming Guide.
14. Касперски К. Техника оптимизации программ. Эффективное использование памяти. – СПб.: БХВ-Петербург, 2003. – 464 с.
15. Ту Дж., Гонсалес Р. Принципы распознавания образов. – М., 1978. – 411 с.
16. Шинкаренко В.И. Зависимость временной эффективности алгоритмов и программ обработки больших объемов данных от их кэширования // Математичні машини і системи. – 2007. – № 2. – С. 43 – 55.

Стаття надійшла до редакції 15.09.2008