УДК 51.681.3

Д.А.Зайцев

О реализации композиционных алгоритмов решения систем линейных уравнений

1. Введение

Представленные в работе алгоритмы могут быть применены для ускорения решения систем линейных уравнений (СЛУ), как в кольцах, так и в полях, однако наиболее существенные ускорения вычислений получены для диофантовых систем, в особенности, при нахождении их решений в неотрицательной области.

К решению систем линейных диофантовых уравнений (СЛДУ) могут быть сведены многие задачи компьютерных наук [1-4]. Специальные методы решения СЛДУ изучены в [5]. Эффективные алгоритмы основаны на построении множеств решений в полях классов вычетов по модулям простых чисел с последующим восстановлением искомого решения в кольце целых [6]. Наиболее сложными с вычислительной точки зрения являются задачи нахождения решений СЛДУ в неотрицательной области (СЛДУН). Все известные в настоящее время алгоритмы решения таких задач [4,7-11] имеют экспоненциальную вычислительную сложность [11].

Следует отметить практическую важность задачи решения линейных диофантовых систем в неотрицательной области. Инварианты сетей Петри [12] являются решениями однородных систем, а фундаментаментальное уравнение сети Петри представляет собой неоднородную систему. Инвариантный анализ [1,12] применяют для исследования структурных свойств сетей Петри, в то время как существование целых неотрицательных решений фундаментального уравнения является необходимым условием достижимости заданного состояния. Большинство задач исследования свойств сетей Петри может быть представлено в виде систем неравенств и сведено к решению систем уравнений [1,9-12].

Кроме теории сетей Петри аналогичные задачи возникают в области искусственного интеллекта, функционального программирования, схемотехники [1-4].

Наиболее часто для решения СЛДУН в настоящее время применяют методы Тудика [7], Контежана [8], Кривого [4,9]. Следует отметить, что метод Кривого обобщён [10,11] для систем неравенств и решений в области {0,1}, имеющих важное прикладное значение для теории сетей Петри. Однако асимптотически экспоненциальная вычислительная сложность затрудняет применение указанных методов для решения систем большой размерности.

В [13] представлены методы и алгоритмы декомпозиции сетей Петри на функциональные подсети. Последующая композиция функциональных подсетей позволяет ускорить процесс поиска инвариантов [14,15] и решения фундаментального уравнения [16]. Полученное ускорение вычислений оценивается экспоненциальной функцией по отношению к размерности сети. Композиционные методы обобщены для произвольных линейных систем [17], в которых роль функциональных подсетей играют формально введенные подмножества уравнений, названные кланами. Однако композиционные методы решения [14-17] сформулированы в обобщённой математической форме, требующей их дальнейшей детализации в процессе эффективной реализации соответствующих алгоритмов.

Целью настоящей работы является построение алгоритмов композиционного решения СЛУ и представление их программной реализации Adriana инвариантной по отношению к выбранному базовому алгоритму решения СЛУ. Следует отметить, что обеспечено встраивание программы в известную систему автоматизированного исследования свойств сетей Петри Tina [18].

В разделах 2, 3 выполнено сжатое изложение основных результатов, представленых в [17], необходимых для дальнейшей разработки алгоритмов. В разделе 4 обоснован выбор структур данных, обеспечивающих инвариантность описаний алгоритмов по отношению к алгебраическим структурам элементов матрицы системы и множества её решений, и, кроме того, предназначенных для представления разрежённых матриц большой размерности;

построена общая схема алгоритма композиционного решения системы и алгоримы работы её основных модулей. В разделе 5 выполнено описание особеностей программной реализации алгоритмов в среде операционной системы Unix, обеспечивающей использование внешних модулей для решения систем. В разделе 7 представлен пример решения системы с помощью разработанной программы Adriana.

2. Основные понятия и определения

Рассмотрим линейную систему m уравнений с n неизвестными

$$A \cdot \bar{x} = \bar{b}\,, \tag{1}$$

где A – матрица коэффициентов размерности $m \times n$, $\bar{x}$ – вектор-столбец неизвестных размерности n, $\bar{b}$ – вектор-столбец свободных членов размерности m; при $\bar{b} = \bar{0}$ будем рассматривать однородную систему, а при $\bar{b} \neq \bar{0}$ – неоднородную. Не будем указывать точно множества значений переменных и коэффициентов. Предположим только, что известен метод, позволяющий решить систему (1) и представить общее решение в форме

$$\bar{x} = \bar{x}' + G \cdot \bar{y}\,, \tag{2}$$

где G – матрица базисных решений однородной системы, $\bar{y}$ – вектор-столбец свободных переменных, $\bar{x}'$ – частное решение неоднородной системы ($\bar{x}' = \bar{0}$ для однородной системы). Представим систему (1) в виде предиката

$$S(\bar{x}) = L_1(\bar{x}) \wedge L_2(\bar{x}) \wedge \ldots \wedge L_m(\bar{x})\,, \tag{3}$$

где $L_i(\bar{x})$ – уравнения системы:

$$L_i(\bar{x}) = (\bar{a}^i \cdot \bar{x} = b_i)\,,$$

$\bar{a}^i$ – i-я строка матрицы A. Будем предполагать также, что $\bar{a}^i$ – ненулевой вектор, то есть, по крайней мере, один из компонентов $\bar{a}^i$ ненулевой. Обозначим X множество неизвестных системы. Рассмотрим множество уравнений $\Im = \{L_i\}$ системы S. Введём отношения на множестве $\Im$.

Отношение близости. Два уравнения $L_i, L_j \in \Im$ *близки* и обозначаются как $L_i \circ L_j$ если и только если $\exists x_k \in X:\ a_{i,k}, a_{j,k} \neq 0,\ sign(a_{i,k}) = sign(a_{j,k})$.

Отношение клана. Два уравнения $L_i, L_j \in \Im$ *удовлетворяют отношению клана*, что обозначается как $L_i \overset{\circ}{\circ} L_j$, если и только если существует последовательность (возможно пустая) уравнений $L_{l_1}, L_{l_2}, ..., L_{l_k}$ таких что: $L_i \circ L_{l_1} \circ ... \circ L_{l_k} \circ L_j$. Заметим, что отношение клана представляет собой транзитивное замыкание отношения близости.

В [17] доказано, что отношение близости рефлексивно и симметрично, отношение клана является отношением эквивалентности и задаёт разбиение множества $\Im$: $\Im = \bigcup_j C^j$, $C^i \bigcap C^j = \varnothing,\ i \neq j$.

Элемент разбиения $\{\Im, \overset{\circ}{\circ}\}$ будем называть *кланом* и обозначать C^j. Переменные $X^j = X(C^j) = \{x_i \,|\, x_i \in X, \exists L_k \in C^j : a_{k,i} \neq 0\}$ будем называть *переменными клана* C^j. Переменные $x_i \in X(C^j)$ являются *внутренними переменными клана* C^j, если и только если для всех остальных кланов C^l, $l \neq j$ выполняется $x_i \notin X^l$. Множество внутренних переменных клана C^j будем обозначать $\widehat{X}^j$. Переменная $x_i \in X$ является *контактной переменной* если и только если существуют такие кланы C^j и C^l, что $x_i \in X^j$, $x_i \in X^l$. Множество всех контактных переменных обозначим X^0. Обозначим также множество контактных переменных клана C^j как $\breve{X}^j$ таким образом что $X^j = \widehat{X}^j \bigcup \breve{X}^j$ и $\widehat{X}^j \bigcap \breve{X}^j = \varnothing$.

В [17] доказано, что контактная переменная $x_i \in X^0$ не может принадлежать различным кланам с одним и тем же знаком, кроме того, контактная переменная $x_i \in X^0$ содержится ровно в двух кланах.

Клан C^j будем называть *входным кланом контактной переменной* $x_i \in X^0$ и обозначать $I(x_i)$, если и только если он содержит эту переменную со знаком плюс. Клан C^j будем называть *выходным кланом контактной переменной* $x_i \in X^0$ и обозначать $O(x_i)$, если

и только если он содержит эту переменную со знаком минус. Аналогичную классификацию на входные и выходные можно ввести также и для контактных переменных.

Таким образом, получено с одной стороны разбиение множества уравнений на кланы, а с другой стороны, разбиение переменных на внутренние и контактные.

Следует отметить, что для произвольной система с матрицей A можно построить матрицу $\mathrm{C} = \mathrm{sign(A)}$, которую можно рассматривать как матрицу инцидентности некоторой ординарной сети Петри N, содержащей m позиций и n переходов. Тогда клан линейной системы соответствует функциональной подсети сети Петри N. Методы декомпозиции сетей Петри на функциональные подсети изучены в [13]; особенности их программной реализации описаны в [19].

3. Свойства кланов линейных систем

Выпишем систему уравнений для произвольного клана C^j:

$$A^j \cdot \bar{x}^j = \bar{b}^j, \tag{4}$$

В соответствии с (2) решение системы (4) имеет вид:

$$\bar{x}^j = \bar{x}'^j + G^j \cdot \bar{y}^j, \tag{5}$$

В [17] доказано, что следующая система (6) эквивалентна системе (1), где k – количество кланов, полученных при разбиении исходной системы (1).

$$\begin{cases} \bar{x}^j = \bar{x}'^j + \bar{y}^j \cdot G^j, \quad j = \overline{1,k}, \\ \bar{x}'^j + G_i^j \cdot \bar{y}^j = \bar{x}'^l + G_i^l \cdot \bar{y}^l, \quad x_i \in X^0, \quad C^j = O(x_i), \quad C^l = I(x_i). \end{cases} \tag{6}$$

В соответствии с системой (6) необходимо вначале решить системы для каждого из кланов, а затем построить и решить систему композиции для контактных переменных. Представим процесс решения в матричной форме. Выпишем уравнения для контактных переменных

$$\bar{x}_i'^j + G_i^j \cdot \bar{y}^j = \bar{x}_i'^l + G_i^l \cdot \bar{y}^l$$

и далее

$$G_i^j \cdot \bar{y}^j - G_i^l \cdot \bar{y}^l = \bar{b}_i' ,\ \bar{b}_i' = \bar{x}_i'^l - \bar{x}_i'^j$$

либо в матричной форме

$$F \cdot \bar{y} = \bar{b}' .$$

Общее решение этой системы в соответствии с (2) можно представить как

$$\bar{y} = \bar{y}' + R \cdot \bar{z} .$$

и, далее

$$\bar{x} = \bar{y}'' + H \cdot \bar{z} ,\ \bar{y}'' = \bar{x}' + G \cdot \bar{y}' ,\ H = G \cdot R . \qquad (7)$$

Для однородной системы выражения (7) имеют вид

$$\bar{x} = H \cdot \bar{z} ,\ H = G \cdot R . \qquad (8)$$

Следует отметить, что в соответствии с [10] в области целых неотрицательных чисел необходимо рассмотреть множество минимальных частных решений неоднородной системы. Пусть $M(q)$ – сложность решения линейной системы размера q. Оценим общую сложность решения линейной системы с помощью декомпозиции. Пусть исходная система (1) состоит из k кланов. Тогда размер каждого из кланов можно оценить, как $p = q/k$. Будем предполагать также, что количество контактных позиций также около p. В [17] показано, что полученное ускорение вычислений оценивается функцией:

$$Acc(k \cdot p) = \frac{M(k \cdot p)}{k^3 \cdot p^3 + k \cdot M(p)} .$$

Таким образом, даже для полиномиальных методов степени, превышающей кубическую, получаем ускорение большее единицы. Для методов, имеющих экспоненциальную сложность $M(q) = 2^q$, таких, как, например, метод Тудика [7] ускорение экспоненциально:

$$AccE(q) = \frac{2^q}{k^3 \cdot p^3 + k \cdot 2^p} \approx \frac{2^q}{2^p} = 2^{q-p} .$$

4. Описания алгоритмов

Метод композиционного решения линейной системы, представленный в [17], можно сформулировать в виде последовательности действий, состоящей из четырёх основных этапов:

Этап 1. Выполнить декомпозицию системы (1) на множество кланов: $\{C^j\}$.

Этап 2. Для каждого из кланов C^j решить систему (4): $\bar{x}^j = \bar{x}'^j + \bar{y}^j \cdot G^j$.

Этап 3. Построить и решить систему (9) для контактных переменных: $\bar{y} = \bar{y}' + R \cdot \bar{z}$.

Этап 4. Скомпоновать матрицу G и вычислить матрицу H базисных решений $H = R \cdot G$; вычислить частное решение $\bar{y}'' = \bar{x}' + G \cdot \bar{y}'$.

Построим алгоритмы нахождения композиционного решения (8) однородных систем, так как поиск частного решения неоднородной системы может быть выполнен с помощью алгоритмов решения однородных систем [4-6]. Композиционные алгоритмы предназначены для решения систем большой размерности, как правило, представленных разрежёнными матрицами. В качестве основной формы представления данных будем использовать поэлементное представление разрежённых матриц. Таким образом, файлы данных являются последовательностью строк вида:

i j A(i,j)

где *i* – номер строки, *j* – номер столбца, *A(i,j)* – значение элемента. Будем указывать только значения ненулевых элементов. Кроме того, не будем предъявлять дополнительные требования к порядку следования строк и нумерации элементов. Описанный формат представление матриц назовём SPM (sparse matrix) и будем использовать его как для представления матриц систем, так и для представления матриц базисных решений.

Для минимизации требований к оперативной памяти все промежуточные решения будем сохранять в отдельных файлах. Схема взаимосвязи программных модулей и файлов данных представлена на рис. 1.

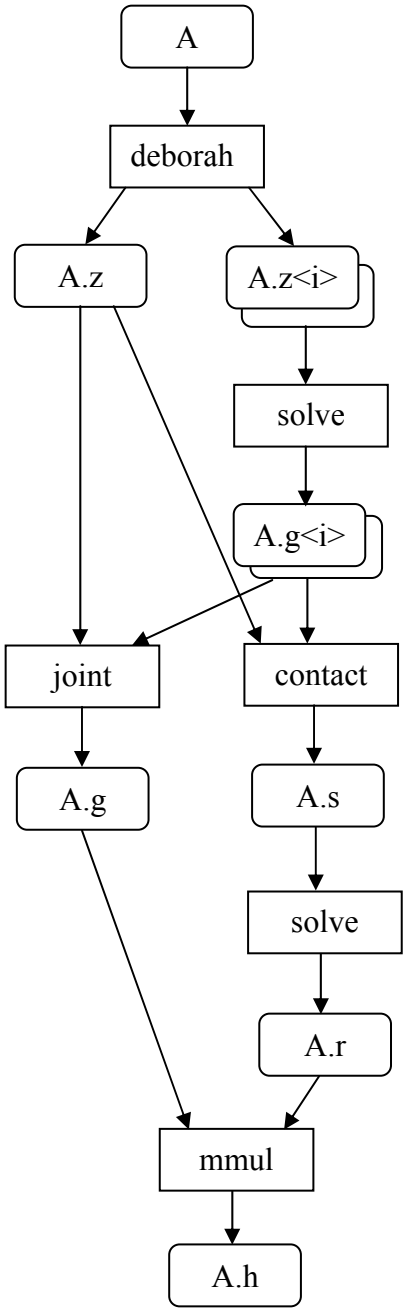

Рис. 1. Общая схема алгоритма композиционного решения системы

Рассмотрим назначение модулей, использованных в указанной схеме. Модуль **deborah** выполняет декомпозицию исходной системы на кланы. Каждый из кланов сохраняется в отдельном файле с расширением *.z<i>*, где *i* равно порядковому номеру клана, кроме того, создаётся файл с расширением *.z*, содержащий описание входных и выходных переменных кланов. Модуль **solve** находит базисные решения системы. Модуль **contact** строит систему уравнений для контактных переменных. Он считывает файл *.z* с номерами контактных переменных, а также файлы базисных решений кланов *.g<i>* и выводит систему композиции

в файл с расширением *.s*. Модуль **joint** строит объединённую матрицу базисных решений кланов. Он считывает файл *.z* и файлы *.g<i>* и создаёт файл *.g*. Модуль **mmul** выполняет умножение разрежённых матриц. Кроме того, при решении диофантовых систем в неотрицательной области может быть использован дополнительный модуль для фильтрации полученных решений в целях минимизации базиса в решётке неотрицательных целых. Как указано в [8,10] минимальный базис состоит их минимальных в решётке элементов.

Следует обратить внимание, что схема, изображённая на рис. 1, инвариантна по отношению к применяемому методу решения линейной системы, инкапсулированному в модуле **solve**. Кроме того, алгоритмы работы остальных модулей инвариантны по отношению к множеству значений элементов матриц систем. Главным требованием является возможность определения и изменения знака элемента *sign(e)*, а также возможность вычисления суммы и произведения элементов при умножении матриц. Таким образом, с помощью композиционных алгоритмов можно решать системы на множествах целых и вещественных чисел, а также в произвольных полях и кольцах.

Опишем файлы данных, используемые при композиционном решении систем:

A – файл матрицы исходной системы, тип SPM;

A.z – файл контактных переменных декомпозиции, тип CVA;

A.z<i> – файл матрицы системы клана *i*, тип SPM;

A.g<i> – файл базисных решений клана *i*, тип SPM;

A.g – файл объединённой матрицы базисных решений кланов, тип SPM;

A.s – файл матрицы системы для контактных переменных, тип SPM;

A.r – файл базисных решений системы для контактных переменных, тип SPM;

A.h – файл базисных решений исходной системы, тип SPM.

Тип файла CVA (contact variable) используется для представления дополнительной информации о контактных переменных декомпозиции на кланы. Он представляет собой последовательность строк вида:

v x y

где *v* – номер контактной переменной, *x* – номер её входного клана, а *y* – номер её выходного клана. Выбор такой формы представления информации позволяет в дальнейшем оптимизировать алгоритмы работы модулей **contact** и **joint**.

Особенности реализации модуля декомпозиции **deborah** описаны в [19], алгоритм работы модуля умножения разрежённых матриц **mmul** тривиален, поэтому остановимся более подробно на описании модулей **contact** и **joint**.

```
static int is=0;
#define lnum( j ) (is+(j))

contact( nz; файл A.z; файлы A.g<i> )
{
 считать файл A.z в C; положить nc равным количеству записей;
 открыть выходной файл sFile;

 для z=1..nz
 {
   открыть файл A.g<z> как zFile;

   nl=0;
   пока( не конец_файла( zFile ) )
   {
     считать из zFile строку i, j, e;
     для ic=1..nc
     {
       если( C[ic].v==j && C[ic].x==z )
         записать в sFile строку lnum(i), j, opposite_sign( e );
       если( C[ic].v==j && C[ic].y==z )
         записать в sFile строку lnum(i), j, e;
     }
     nl=max(nl,i);
   }
   is+=nl;
 }
} /* contact */
```

Рис. 2. Алгоритм работы модуля **contact**

Алгоритм работы модуля **contact**, выполняющего построение системы уравнений композиции, на Си подобном псевдоязыке представлен на рис. 2. Каждое уравнение создаваемой системы соответствует контактной переменной, указанной в файле *A.z*. Переменные системы композиции соответствует базисным решениям для кланов. Файл декомпозиции *A.z*, содержащий описания контактных переменных, считывается целиком и

размещается в памяти. Затем последовательно считываются файлы базисных решений кланов, причём в памяти размещается только одна запись. Если запись содержит контактную переменную, являющуюся входной либо выходной для обрабатываемого клана, то выводится терм в файл системы композиции. Использование макроса *lnum(l)* позволяет организовать нумерацию всех базисных решений кланов за счёт накопления количества решений ранее обработанных кланов в переменной *is*. Информация о природе элемента матрицы инкапсулирована в функции *opposite_sign(e)*, которая определяет значение элемента противоположного знака. Сам элемент вводится и храниться как строка, и не интерпретируется алгоритмом.

```
static int is=0;
#define vnum( j ) (is+(j))

joint( nz; файл A.z; файлы A.g<i> )
{
 считать файл A.z в C; положить nc равным количеству записей;
 открыть выходной файл gFile;

 для z=1..nz
 {
   открыть файл A.g<z> как zFile;

   nl=0;
   пока( не конец_файла( zFile ) )
   {
     считать из zFile строку i, j, e;
     yes=1;
     для ic=1..nc
       если( C[ic].v==j && C[ic].x!=z ) { yes=0; прерватьцикл; }
     если(yes) вывести в gFile строку vnum(i), j, e;
     nl=max(nl,i);
   }
   is+=nl;
 }
} /* joint */
```

Рис. 2. Алгоритм работы модуля **joint**

Алгоритм работы модуля **joint**, выполняющего построение объединённой матрицы базисных решений кланов, на Си подобном псевдоязыке представлен на рис. 3. Особенность алгоритма заключается в том, что базисные решения для каждой из контактных переменных должны быть вычислены либо в соответствии с базисными решениями входного клана, либо

в соответствии с базисными решениями выходного клана. Для определённости выбрано вычисление в соответствии с базисными решениями входного клана. Таким образом, не все элементы базисных решений кланов попадают в объединённую матрицу: переписывать следует либо решения для внутренних переменных, либо для входных контактных переменных. Кроме того, алгоритмом выполняется перенумерация базисных решений. Так как элемент матрицы только копируется, он не интерпретируется алгоритмом.

5. Особенности программной реализации

При реализации композиционных алгоритмов учитывались такие требования как решение систем большой размерности, представленных разрежёнными матрицами, а также инвариантность по отношению к природе элемента матрицы и алгоритму решения системы. Исходя из этих требований каждый из модулей, указанных на рис. 1, реализован как отдельная программа, использующая интерфейс командной строки. Программирование выполнено на языке ANSI C в среде операционной системы Unix. Для объединения модулей использован командный файл, позволяющий легко заменять модуль **solve** фактически применяемым для решения линейной системы модулем.

Командный файл (скрипт) оболочки bash операционной системы Unix представлен на рис. 4. Значение переменной *nz* равно количеству кланов, полученных при декомпозиции. Переменная цикла *z* используется для нумерации кланов. Предусмотрен специальный случай неразложимости системы на кланы (*nz=1*). Следует отметить, что промежуточные файлы *A.** за исключением результата *A.h* могут быть удалены.

Тестирование командного файла выполнено для модулей **solve** представляющих алгоритм Гаусса для вещественных чисел (**gauss**) и алгоритм Тудика [7], для решения диофантовых уравнений в неотрицательной области (**toudic**). Примеры композиционного решения систем представлены в следующем разделе. Использование ANSI C позволяет легко переносить систему на другие платформы. Кроме того, при необходимости командный файл

также может быть закодирован на языке Си и специализирован для определённого множества алгоритмов решения системы.

```
#!/bin/bash
nz=`deborah $1`
echo decomposed into $nz subsystems
z=1
while [ $z -le $nz ]
do
  echo sybsystem $z is solved...
  solve $1.z$z $1.g$z
  z=`expr $z + 1`
done
if [ $nz -eq 1 ]
then
  mv $1.g1 $1.h
  exit
fi
contact $1
echo 'composition system has been created'
solve $1.s $1.r
echo 'composition system has been solved'
joint $1
echo 'joint matrix has been constructed'
mmul $1.r $1.g $1.h
echo 'result has been obtained'
#end
```

Рис. 4. Командный файл Unix композиционного решения системы

Рассмотрим более подробно пример такой реализации, предназначенный для встраивания в известную систему автоматизированного анализа сетей Петри Tina [18]. Программа Adriana выполняет поиск инвариантов сетей Петри с помощью композиционных алгоритмов. Для решения систем выбран алгоритм Тудика (модуль **toudic**) [7]. Инварианты сети Петри представляют собой целые неотрицательные решения однородной линейной диофантовой системы уравнений. Матрицей системы является матрица инцидентности сети Петри [12] для инвариантов позиций, либо транспонированная матрица инцидентности для инвариантов переходов. Как было отмечено ранее, решение таких систем является наиболее трудоёмким с вычислительной точки зрения. Поэтому ускорения вычислений, получаемые при решении таких систем, являются наиболее ощутимыми. В литературе [1-4,20] отмечены примеры сетей, насчитывающих несколько десятков элементов и требующих для вычисления инвариантов более 10^{20} операций.

При реализации программы Adriana кроме кодирования командного файла на языке ANSI C и встраивания в программу перечисленных модулей, решена также задача перекодирования форматов файлов представляющих сети Петри в системе Tina. Система Tina использует два основных формата файлов для представления сетей Петри: абстрактный NET и графический NDR. Формальное описание форматов приведено в дистрибутиве системы [18]. Отметим, что оба формата используют мнемонические имена элементов сети и содержат ряд дополнительных характеристик, несущественных для вычисления инвариантов. Кроме того, формат NDR содержит положение элементов сети на плоскости. Наиболее простым представляется перекодирование NET/NDR файла в формат SPM с сохранением таблиц имён позиций и переходов сети. После композиционного решения системы таблицы имён используются для мнемонического представления инвариантов, принятого в системе Tina. Алгоритм работы программы Adriana представлен на рис. 5.

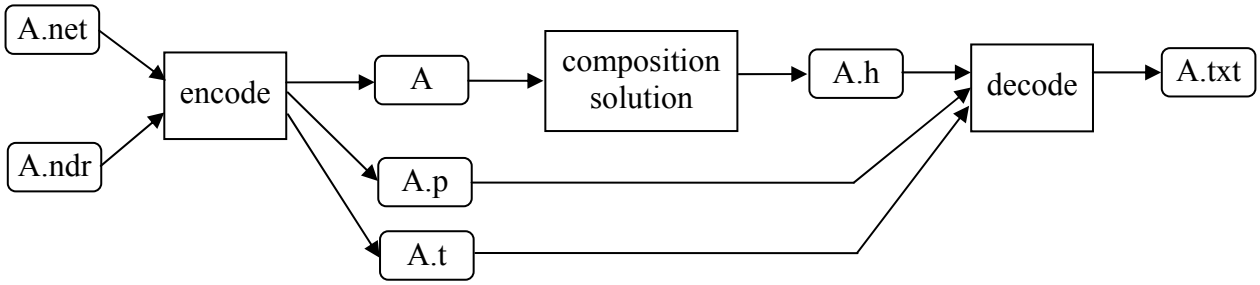

Рис. 5. Алгоритм работы программы Adriana

Файлы *A.p* и *A.t* содержат таблицы имён позиций и переходов сети Петри соответственно и представляют собой последовательность строк вида:

n name

где *n* – номер элемента, а *name* – его имя. Модуль **encode** нумерует элементы сети и создаёт файл матрицы инцидентности *A*, а также таблицы имён. Модуль **decode** представляет инварианты в мнемонической форме системы Tina:

name1[*val1] name2[*val2] ... namek[*valk]

где *namei* – имя элемента, *vali* – значение инварианта; единичные значения элементов опускаются. Промежуточные файлы создаются в каталоге временных файлов и удаляются по завершению работы программы. Программы Deborah и Adriana могут быть загружены с сайта www.geocities.com/zsoftua.

6. Пример композиционного решения системы

Решим СЛДУН с помощью программы Adriana, реализующей композиционные алгоритмы. Следует отметить, что для получения существенных ускорений вычислений необходимо рассматривать сравнительно большие системы уравнений. Пример системы, представленной на рис. 6 достаточно интересен. Система предназначена для поиска инвариантов позиций модели Петри телекоммуникационного протокола TCP, описанной в [20]. Кроме того, поиск базисных инвариантов с помощью стандартного модуля системы Tina [18] для целочисленных свободных переменных потребовал нескольких суток; программа Adriana решила систему за десяток секунд; использован персональный компьютер с 256 мегабайт оперативной памяти, процессор Duron 800, ОС Linux.

На рис. 6 кроме исходной матрицы системы A и результирующей матрицы базисных решений H представлены описанные в предыдущем разделе промежуточные матрицы, создаваемые в процессе композиционного решения. Следует отметить, что для наглядности матрицы представлены стандартно, а не в разрежённом формате SPM, используемом программой. Исходная система с матрицей A разделяется на 4 клана. Полученная в результате применения композиционных алгоритмов матрица *A.h0* содержит 48 решений; после фильтрации в базисе остаётся 24 минимальных решения, представленных в матрице *A.h*.

A	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	-1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	-1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	-1	0	1	0	0	0	0	0	0	0	1	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	-1	1	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	-1	0	1	0	0	0	0	0	0	0	-1	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0
13	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	1	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	-1	0	-1	1	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	-1	1	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	-1	0	1	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	0	1	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	0	0	-1	0	1	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	-1	0	0	1	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	-1	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	-1	0	1
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0	0	0	0	0	0	-1	1
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	-1

A.z	V	1	5	12	13	14	15	16	17	18	19	20	24
	X	1	2	3	1	3	1	4	2	4	2	3	4
	Y	2	1	1	3	1	3	2	4	2	4	4	3

A.z1: L1 - L7
A.z2: L8 - L16
A.z3: L17 - L23
A.z4: L24 - L32

A.g1	1	2	3	4	5	6
1	1	0	0	1	0	1
2	1	0	0	1	0	1
3	0	0	0	1	0	1
4	0	0	0	0	1	1
5	0	1	0	0	1	1
12	1	0	0	0	0	0
13	0	0	1	0	1	0
14	0	0	1	1	0	0
15	0	1	0	0	0	0

A.g2	1	2	3	4	5	6
1	0	1	0	0	1	1
5	0	0	0	1	0	1
6	1	0	0	0	1	1
7	0	0	0	1	0	1
8	0	0	0	0	1	1
9	0	0	0	0	1	1
10	0	1	0	1	0	1
11	0	1	0	0	1	1
16	1	0	0	0	0	0
17	0	0	1	0	1	0
18	0	0	1	1	0	0
19	0	1	0	0	0	0

A.g3	1	2	3	4	5	6
12	0	0	1	0	1	0
13	0	1	0	0	0	0
14	1	0	0	0	0	0
15	0	0	1	1	0	0
20	0	1	0	1	0	1
21	0	1	0	1	0	1
22	0	0	0	1	0	1
23	0	0	0	0	1	1
24	1	0	0	0	1	1

A.g4	1	2	3	4	5	6
16	0	0	1	0	1	0
17	0	1	0	0	0	0
19	0	0	1	1	0	0
18	1	0	0	0	0	0
20	1	0	0	0	1	1
24	0	1	0	1	0	1
25	0	1	0	0	1	1
26	0	0	0	1	0	1
27	0	0	0	0	1	1
28	0	0	0	0	1	1
29	1	0	0	1	0	1
30	1	0	0	0	1	1

A.s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	-1	0	0	-1	0	-1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	1	1	-1	0	0	-1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0
12	1	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	-1	0	0	0	0	0	0	0
13	0	0	-1	0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
14	0	0	1	1	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0
15	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	-1	0
17	0	0	0	0	0	0	0	0	0	-1	0	-1	0	0	0	0	0	0	0	1	0	0	0	0
18	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	-1	0	0	0	0	0
19	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	-1	0	-1	1	0	0	0	1	1
24	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	-1	0	-1	0	-1

Рис. 6. Пример композиционного решения системы

A.r	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
1	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
2	0	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
5	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
6	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
8	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
10	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
13	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
14	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
15	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
17	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
18	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
20	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
21	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
22	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

A.h0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
1	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
2	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
4	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
5	1	1	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	1	1	0	0	1	1	1	1	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0	0	2	2	1	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
8	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
9	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
10	1	1	1	1	1	1	0	0	0	0	0	0	1	1	0	0	1	1	2	2	1	1	2	2	0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
13	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
14	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
15	0	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
16	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
17	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
18	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
19	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
20	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
21	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
22	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
23	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
24	0	0	1	1	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
25	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	1	1	1	0	0	1	1	0	0	2	2	1	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
26	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
29	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1	1	1	1	2	2	1	1	2	2	0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
30	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

A.g	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
13	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
15	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
17	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
19	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1

A.h	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
3	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0
4	1	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	1	0	1
5	1	1	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1
6	1	1	0	0	1	1	1	1	0	0	1	1	0	0	0	0	0	0	2	2	1	1	1	1
7	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0
8	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
9	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
10	1	1	1	1	1	1	0	0	0	0	0	0	1	1	0	0	2	2	0	0	1	1	1	1
11	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
12	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1
13	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
14	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	1	1	0	1	0
15	0	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1	0	1	1	0	1	0
16	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1
17	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1
18	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1
19	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
20	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
21	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
22	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
23	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0	0	1	0	1
24	0	0	1	1	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1
25	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	1	0	0	2	2	1	1	1	1
26	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1	0	0
27	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0
28	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0
29	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1	1	2	2	0	0	1	1	1	1
30	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Рис. 6. Пример композиционного решения системы (продолжение)

7. Заключение

Таким образом, построены композиционнае алгоритмы решения СЛУ, предназначеные для ускорения процессов решения систем методами с вычислительной сложностью, превышающей кубическую. Для экспоненциальных методов решения систем, таких, как,

например, методы решения диофантовых уравнений в неотрицательной области, ускорение вычислений экспоненциально.

Выполнена программная реализация построенных алгоритмов. Обеспечено встраивание модуля композиционного поиска инвариантов Adriana в систему автоматизированного анализа сетей Петри Tina.

1. Слепцов А.И., Юрасов А.А. Автоматизация проектирования управляющих систем гибких автоматизированных производств / Под ред. Б.Н.Малиновского.- К. Техніка, 1986.- 160 с.
2. Cortadella J., Kishinevsky M., Kondratyev A., Lavagno L., Yakovlev A. Logic synthesis of asynchronous controllers and interfaces. Springer, 2002, 273 p.
3. Girault C., Valk R. Petri nets for systems engineering. Springer-Verlag, 2003, 607 p.
4. Крывый С.Л. О некоторых методах решения и критериях совместимости систем линейных диофантовых уравнений в области натуральных чисел // Кибернетика и системный анализ, 1999, № 4, с. 12-36.
5. Схрейвер А. Теория линейного и целочисленного программирования. В 2-х т. М.: Мир, 1991, 360с.
6. Вотяков А.А., Фрумкин М.А. Алгоритм нахождения общего целочисленного решения системы линейных уравнений // Исследования по дискретной оптимизации, М.: Наука, 1976, с. 128-140.
7. Toudic J.M.: Linear Algebra Algorithms for the Structural Analysis of Petri Nets. Rev. Tech. Thomson CSF (1982) no. 1, Vol. 14, 136-156.
8. Contejan E., Ajili F. Avoiding slack variables in solving linear Diophantine equations and inequations // Theor. Comp. Sci., 173, 1997, 183-208.
9. Крывый С.Л., Чугаенко А.В., Богак Н.А., Бура В.В. О реализации алгоритмов проверки совместимости систем линейных диофантовых уравнений в области натуральных чисел // Управляющие системы и машины, № 3, 1999, с. 26-32.
10. Крывый С.Л. Совместимость систем линейных неравенств на множестве натуральных чисел // Кибернетика и системный анализ, №1, 2002, с. 17-27.
11. Крывый С.Л. Об алгоритмах решения систем линейных диофантовых констрейнтов в области {0,1} // Кибернетика и системный анализ, №5, 2003, с. 58-69.
12. Мурата Т. Сети Петри: Свойства, анализ, приложения // ТИИЭР, т. 77, №4, 1989, с. 41-85.
13. Зайцев Д.А. Декомпозиция сетей Петри // Кибернетика и системный анализ, №5, 2004, с. 131-140.
14. Зайцев Д.А. Инварианты функциональных подсетей // Труды Одесской национальной академии связи им. А.С.Попова, № 4, 2003, с. 57-63.
15. Zaitsev D.A. Decomposition-based calculation of Petri net invariants // Proceedings of Workshop on Token based computing of the 25-th International conference on application and theory of Petri nets, Bologna, Italy, June 21-25, 2004, с. 79-83.
16. Zaitsev D.A. Solving the fundamental equation of Petri net using the decomposition into functional subnets // 11th Workshop on Algorithms and Tools for Petri Nets, September 30 - October 1, 2004, University of Paderborn, Germany, p. 75-81.
17. Зайцев Д.А. Ускорение решения линейных систем с помощью декомпозиции на кланы // Искусственный интеллект. Интеллектуальные и многопроцессорные системы-2004. Материалы международной научно-технической конференции. Т.1. Таганрог: Изд-во ТРТУ, 2004, с.259-264.
18. Berthomieu B., Ribet O.-P., Vernadat F. The tool TINA – construction of abstract state space for Petri nets and Time Petri nets. International Journal of Production Research (2004) (www.laas.fr/tina)
19. Зайцев Д.А. Программное обеспечение для декомпозиции двудольных орграфов // Научные труды Донецкого государственного технического университета, серия "Информатика, кибернетика и вычислительная техника", Вып. 93, 2005, с. 60-70.
20. Zaitsev D.A. Verification of protocol TCP via decomposition of Petri net model into functional subnets // Proceedings of the Poster session of 12th Annual Meeting of the IEEE / ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, October 5-7, 2004, Volendam, Netherlands, p. 73-75.