УДК 519.7

НЕПОЛНЫЕ АЛГОРИТМЫ В КРУПНОБЛОЧНОМ ПАРАЛЛЕЛИЗМЕ КОМБИНАТОРНЫХ ЗАДАЧ

А. А. Семенов[1], **О. С. Заикин**[1]

Для некоторых типов NP-трудных комбинаторных задач исследуются технологии поиска их решений посредством неполных алгоритмов, т.е. алгоритмов, конечность работы которых в общем случае не гарантируется. Речь идет об алгоритмах "программирования в ограничениях", использующих в своей работе идею пополнения базы ограничений с отбрасыванием нерелевантных ограничений. Сравниваются два подхода к решению ряда комбинаторных задач посредством неполных алгоритмов. Основой первого подхода является крупноблочное распараллеливание исходной задачи с последующим решением получаемых задач неполным алгоритмом на кластере. Во втором подходе исходная задача решается на одном процессора кластера (фактически на ПК) тем же самым алгоритмом. Показано, что в ряде случаев коэффициент ускорения, которое достигается в первом подходе, может существенно превосходить число процессоров кластера. Работа выполнена при финансовой поддержке РФФИ (код проекта 07–01–00400-а) и при частичной поддержке гранта Президента РФ НШ 1676.2008.1. Статья подготовлена по материалам доклада авторов на международной научной конференции "Параллельные вычислительные технологии" (ПаВТ-2008; http://agora.guru.ru/pavt).

Ключевые слова: база ограничений, неполнота, крупноблочный параллелизм, декомпозиция, программирование в ограничениях, параллельные программы, прогнозная функция.

1. Введение. В настоящей статье исследуется возможность использования концепции крупноблочного распараллеливания применительно к некоторым вычислительно сложным комбинаторным задачам.

К сожалению, для большого числа комбинаторных задач, практическая значимость которых неоспорима, не известны гарантированно эффективные (полиномиальные) алгоритмы поиска их решений. Речь идет, главным образом, о NP-трудных задачах. Все известные подходы к практическому решению таких задач задействуют разного рода приемы, ускоряющие процесс поиска. Существенного ускорения можно достичь, например, в результате частичного сохранения предыстории поиска, для чего к системе ограничений, формирующих исходную задачу, добавляются новые ограничения, получаемые в ходе поиска. В этом заключается стратегия пополнения базы ограничений, отрицательной стороной которой является вероятность чрезмерного роста объема требуемой памяти. В современных индустриальных алгоритмах "программирования в ограничениях" данная проблема решается посредством процедур "чистки базы ограничений", т.е. путем отбрасывания ограничений, из каких-либо соображений признаваемых неэффективными (принятый термин — "нерелевантные"). Процедуры такого рода, позволяющие добиваться реального ускорения поиска, как правило, носят характер эвристик. Данный факт означает принципиальную возможность потери исходным алгоритмом его полноты, т.е. потери гарантии того, что алгоритм за конечное число шагов приведет к решению задачи.

Основная идея подхода, излагаемого в данной работе, состоит в том, что при декомпозиции исходной задачи на семейство аналогичных подзадач меньшей размерности эвристический алгоритм описанного выше типа, решающий задачи из полученного семейства, грубо говоря, не успевает потерять полноту. Следствием этого является резкий рост производительности при решении задачи на вычислительном кластере по сравнению с ее решением на обычном "последовательном" компьютере. Описанный эффект демонстрируется на примере задачи криптоанализа одной системы поточного шифрования (пороговый генератор).

2. Комбинаторные алгоритмы с модифицируемыми базами ограничений. В данном разделе приводятся характерные примеры алгоритмов, в которых используется концепция модифицируемой базы ограничений. Все рассматриваемые далее комбинаторные проблемы относятся к классу задач с булевыми ограничениями — именно на таких задачах удобно демонстрировать описываемый в последующих разделах статьи подход к крупноблочному распараллеливанию.

[1] Институт динамики систем и теории управления Сибирского отделения РАН, ул. Лермонтова, 134, 664033, Иркутск; e-mail: biclop@rambler.ru, oleg.zaikin@icc.ru

Первый класс задач составляют так называемые SAT-задачи. Булевыми переменными называются переменные, принимающие значения в множестве $\{0,1\}$. Через $\{0,1\}^n$, $n \in N$, обозначается множество всех двоичных слов длины n. Пусть $X = \{x_1, \ldots, x_n\}$ — множество булевых переменных. Термы x_i, $\overline{x}_i$, $i \in \{1, \ldots, n\}$, называются литералами над X. Литералы x и $\overline{x}$ называются контрарными. Дизъюнктом над X называется произвольная дизъюнкция литералов над X, в которой нет повторяющихся и контрарных литералов. Конъюнктивной нормальной формой (КНФ) над X называется произвольная конъюнкция различных дизъюнктов над X. Пусть L — произвольная формула алгебры логики [1] от переменных $x_1, \ldots, x_n$. Тот факт, что L при подстановке $x_1 = \alpha_1$, …, $x_n = \alpha_n$, $\alpha_i \in \{0,1\}$, $i \in \{1, \ldots, n\}$, принимает значение $\beta \in \{0,1\}$, обозначаем через $L\big|_{(\alpha_1,\ldots,\alpha_n)} = \beta$.

Пусть C — произвольная КНФ над множеством булевых переменных $X = \{x_1, \ldots, x_n\}$. Вектор $(\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$ называется набором, выполняющим C, если $C\big|_{(\alpha_1,\ldots,\alpha_n)} = 1$. КНФ над X, для которой существует выполняющий набор, называется выполнимой; в противном случае КНФ называется невыполнимой. К SAT-задачам, рассматриваемым далее, относится задача распознавания выполнимости произвольной КНФ (исторически первая NP-полная задача [2]), а также задача поиска выполняющего набора произвольной выполнимой КНФ.

Первый рассматриваемый алгоритм решения SAT-задач — это алгоритм DPLL [3]. Данный алгоритм является модифицированным вариантом процедуры Девиса и Патнема [4]. Основное отличие DPLL от алгоритма Девиса–Патнема состоит в стратегии распространения булевых ограничений (Boolean constraint propagation), использующей правило единичного дизъюнкта (unit clause). В ходе работы DPLL угадываются значения некоторых булевых переменных, при этом строится иерархия уровней угадывания. Поскольку основное внимание практической части работы уделено именно DPLL, подробно поясним особенности его работы на примерах.

Пример 1. Рассмотрим КНФ

$$(x_1 \vee \overline{x}_2) \cdot (\overline{x}_1 \vee x_2) \cdot (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \cdot (\overline{x}_2 \vee \overline{x}_3 \vee x_4) \cdot (x_3 \vee \overline{x}_4).$$

Угадываем “$x_4 = 1$”. После подстановки угаданного значения в исходную КНФ получаем КНФ

$$(x_1 \vee \overline{x}_2) \cdot (\overline{x}_1 \vee x_2) \cdot (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \cdot (x_3).$$

Очевидно, что если данная КНФ выполнима, то в любом выполняющем ее наборе переменная x_3 принимает значение 1. В подобных случаях говорят, что угадывание “$x_4 = 1$” индуцирует присвоение “$x_3 = 1$” по правилу единичного дизъюнкта. Итак, можно заключить, что в любом наборе, выполняющем исходную КНФ, в котором $x_4 = 1$, переменная x_3 также принимает значение 1, а значения переменных x_1 и x_2 определяют набор, выполняющий КНФ

$$(x_1 \vee \overline{x}_2) \cdot (\overline{x}_1 \vee x_2).$$

На данном этапе требуется осуществлять новое угадывание. Например, угадывание “$x_1 = 1$” и последующее применение правила единичного дизъюнкта дает набор $x_1 = x_2 = 1$, выполняющий КНФ

$$(x_1 \vee \overline{x}_2) \cdot (\overline{x}_1 \vee x_2).$$

После этого относительно исходной КНФ заключаем, что она выполнима на наборе (1111).

В некоторых случаях результатами применения правила единичного дизъюнкта могут быть так называемые “конфликты”. Конфликтом называется ситуация, когда по правилу единичного дизъюнкта (из угаданных на текущий момент присвоений) выводятся одновременно “$x = 0$” и “$x = 1$” (для некоторой булевой переменной x). Например, для КНФ

$$(x_1 \vee x_2) \cdot (\overline{x}_1 \vee x_2) \cdot (x_1 \vee \overline{x}_2) \cdot (\overline{x}_1 \vee \overline{x}_2)$$

угадывание “$x_1 = 1$” дает вывод из второго дизъюнкта “$x_2 = 1$”, а из четвертого дизъюнкта “$x_2 = 0$”. Если такое происходит, то вступает в действие процедура разрешения конфликта. В рассматриваемом примере следует изменить присвоение “$x_1 = 1$” на противоположное, т.е. положить “$x_1 = 0$”.

Переменные, значения которых угадываются, называют переменными уровней решений (в англоязычных источниках “decision level”); уровни решений при этом нумеруются. На самом деле иерархию уровней решений можно представить в виде бинарного дерева, корнем которого является первый уровень, помечаемый соответствующей булевой переменной. Потомками корня являются переменные, значения которых

угадываются и индуцируются (по правилу единичного дизъюнкта) на последующих уровнях. Ветви, выходящие из произвольной вершины, помечаются значениями соответствующих переменных. Листья дерева помечаются как “sat” либо как “conflict”. Путь из корня в лист, помеченный как “sat”, определяет набор, выполняющий исходную КНФ. Путь из корня в лист, помеченный как “conflict”, определяет последовательность угадываний, из которой по правилу единичного дизъюнкта был выведен конфликт. Несложно понять, что если КНФ выполнима, то для любой альтернативы корня данного дерева всегда найдется путь в лист, помеченный как “sat”. Если же КНФ невыполнима, то все пути из любой альтернативы корня будут оканчиваться листьями, помеченными как “conflict”.

Первоначально в DPLL был принят элементарный механизм разбора конфликтов — хронологический бэктрекинг. При хронологическом бэктрекинге разрешение конфликта происходит за счет изменения последнего (перед конфликтом) уровня решения — значение угаданной на этом уровне переменной меняется на противоположное. Позже была сформулирована концепция нехронологического бэктрекинга (или бэкджампинга), которая привела к созданию по-настоящему скоростных SAT-решателей. Ключевым механизмом бэкджампинга является процедура “clause learning”, позволяющая запоминать информацию о конфликтах. Далее мы иллюстрируем основные перечисленные моменты, используя обозначения, принятые в [5].

Пример 2. Рассмотрим КНФ

$$\big(x_1 \vee \overline{x}_2 \vee x_3\big) \cdot \big(\overline{x}_1 \vee x_2\big) \cdot \big(x_1 \vee x_2 \vee x_3\big) \cdot \big(\overline{x}_2 \vee \overline{x}_3 \vee x_4\big) \cdot \big(x_3 \vee \overline{x}_4\big).$$

Угаданные присвоения далее обозначаются через $x = \alpha@s$; здесь $\alpha \in \{0,1\}$ — угаданное значение переменной x, а $s \in N$ — номер уровня решения; индуцированные присвоения обозначаются через $x = \beta$, $\beta \in \{0,1\}$. Процесс вывода индуцированных присвоений по правилу единичного дизъюнкта можно представить в виде специального ориентированного графа, называемого графом вывода (implication graph).

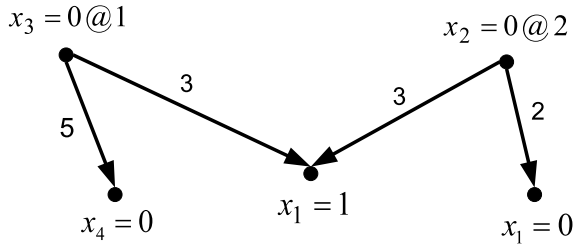

Рис. 1. Пример графа вывода

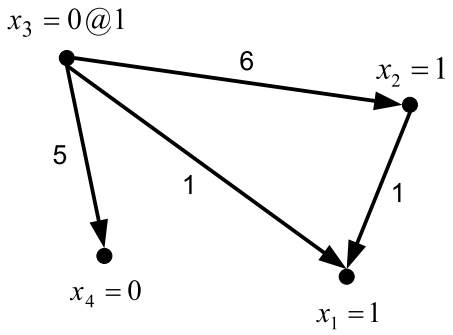

Рис. 2. Граф вывода в предположении “$x_3 = 0$”

На рис. 1 изображен граф вывода для двух уровней решения: на первом уровне угадывается “$x_3 = 0$”, после чего по правилу единичного дизъюнкта из пятого дизъюнкта выводится присвоение “$x_4 = 0$”; на втором уровне угадывается “$x_2 = 0$”, после чего из третьего и второго дизъюнктов выводится конфликт (номера дизъюнктов, на которых срабатывает правило единичного дизъюнкта, приписываются дугам графа вывода). Процедура “clause learning” запоминания информации о конфликте (далее “CL-процедура”) заключается в конъюнктивном приписывании к исходной КНФ нового ограничения — дизъюнкта, запрещающего присвоения, приведшие к конфликту. В рассматриваемом случае это одновременные присвоения $x_3 = 0$ и $x_2 = 0$; таким образом, от исходной КНФ переходим к КНФ

$$\big(x_1 \vee \overline{x}_2 \vee x_3\big) \cdot \big(\overline{x}_1 \vee x_2\big) \cdot \big(x_1 \vee x_2 \vee x_3\big) \cdot \big(\overline{x}_2 \vee \overline{x}_3 \vee x_4\big) \cdot \big(x_3 \vee \overline{x}_4\big) \cdot \big(x_2 \vee x_3\big).$$

Несложно видеть, что данная КНФ выполнима на тех и только тех наборах, на которых выполнима исходная КНФ. Построим граф вывода для данной КНФ (рис. 2), предположив, что на первом уровне происходит угадывание “$x_3 = 0$”. Таким образом, исходная КНФ выполнима на наборе $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0$.

В общем случае использование CL-процедуры позволяет точно выявлять присвоения, ответственные за конфликт, вследствие чего возможны откаты не к последнему уровню решения (как в хронологическом бэктрекинге), а к более ранним в иерархии угадываний уровням. В этом и состоит основной конструктивный момент нехронологического бэктрекинга (бэкджампинг). Описанный механизм послужил основой

целой серии результатов, приведшей к созданию высокоскоростных SAT-решателей, успешно решающих SAT-задачи с десятками тысяч переменных и сотнями тысяч дизъюнктов [6 – 8]). С другой стороны, можно заметить, что многократное использование CL-процедуры приводит к росту объема памяти, занимаемой базой ограничений-дизъюнктов. Во всех современных SAT-решателях предусмотрены процедуры "чистки базы ограничений", в результате которых некоторые ограничения объявляются нерелевантными и удаляются из базы. Данный факт в общем случае может привести к потере алгоритмом полноты — в ходе одного из последующих угадываний алгоритм может пойти по ветке, которая была пройдена ранее, тогда как информация об этом содержалась в отброшенном ограничении.

Еще одним часто используемым методом решения SAT-задач является метод резолюций. Данный метод впервые был описан в работе [9] и долгое время считался одним из наиболее перспективных подходов к автоматическому доказательству теорем. Кратко опишем его основу. Рассматривается произвольная КНФ

$$C = D_1 \cdot \ldots \cdot D_m, \tag{1}$$

где D_j — дизъюнкты над множеством булевых переменных $X = \{x_1, \ldots, x_n\}$, $j \in \{1, \ldots, m\}$. Ставится вопрос о выполнимости C. Если дизъюнкты D_{k_1} и D_{k_2}, $k_1, k_2 \in \{1, \ldots, m\}$, содержат контрарные литералы x и $\overline{x}$ (например, первый дизъюнкт содержит x, а второй — $\overline{x}$), то говорят, что D_{k_1} и D_{k_2} контрарны по паре $\{x, \overline{x}\}$. Если D — произвольный дизъюнкт, а a —литерал, входящий в D, то через $D \backslash \{a\}$ обозначается дизъюнкт, полученный из D вычеркиванием a. Пусть D_{k_1} и D_{k_2}, $k_1, k_2 \in \{1, \ldots, m\}$, контрарны по паре $\{x, \overline{x}\}$ для некоторого $x \in X$ (для определенности полагаем, что D_{k_1} содержит x, а D_{k_2} содержит $\overline{x}$). Дизъюнкт $D' = \left(D_{k_1} \backslash \{x\} \vee D_{k_2} \backslash \{\overline{x}\}\right)$ называется резольвентой дизъюнктов D_{k_1} и D_{k_2}. Несложно видеть, что КНФ $C' = C \cdot D'$, где D' — резольвента некоторой пары дизъюнктов из C, выполнима на тех и только тех наборах значений истинности переменных из X, на которых выполнима C. Далее ставится вопрос о выполнимости C'. Описанная процедура представляет собой одну итерацию метода резолюций. Дж. А. Робинсоном в [9] было показано, что C невыполнима тогда и только тогда, когда существует такая конечная последовательность итераций метода резолюций, итогом которой является дизъюнкт, не содержащий литералов и называемый пустым (данный дизъюнкт есть резольвента единичных контрарных дизъюнктов вида a и $\overline{a}$).

Из сказанного видно, что метод резолюций, так же как и DPLL, дополненный CL-процедурой, использует идею пополнения базы ограничений. При этом обычно порождается много "бесполезных" (в терминологии [10]) ограничений-дизъюнктов. В ходе решения данной задачи разработано большое число специальных стратегий, позволяющих пополнять базу ограничений наиболее значимыми (релевантными) дизъюнктами. Далеко не все стратегии гарантируют сохранение полноты базового алгоритма. Эти вопросы подробно отражены в фундаментальной монографии [10].

Следующий класс комбинаторных проблем, при практическом решении которых используются неполные алгоритмы, образуют задачи целочисленного линейного программирования (ЦЛП). Для сохранения целостности изложения будем рассматривать класс ЦЛП-задач, к которым эффективно сводятся SAT-задачи. В соответствии с [11], задача о выполнимости (поиске выполняющего набора) произвольной КНФ вида (1) над множеством булевых переменных $X = \{x_1, \ldots, x_n\}$ полиномиально сводится к задаче определения совместности (поиска решения) следующей системы линейных неравенств:

$$Ax \geqslant b, \tag{2}$$

где A — матрица размерности $m \times n$ с элементами из множества $\{-1, 0, 1\} \cap Z$ и b — целочисленный вектор размерности m. Требуется найти вектор x с компонентами из $\{0, 1\} \cap Z$, удовлетворяющий (2).

Ярким представителем алгоритмов, использующих идею пополнения базы ограничений, является метод Гомори решения ЦЛП-задач [11]. На каждой итерации данного метода решается соответствующая (ослабленная) задача линейного программирования с отброшенным требованием целочисленности (решение строится над полем рациональных чисел Q). Если система неравенств несовместна над[2] Q, то нет решения и у целочисленной задачи. В противном случае находится точка с рациональными координатами, являющаяся решением ослабленной задачи. После этого строится новое ограничение в форме линейного неравенства, которое добавляется к первоначальной системе. Новое ограничение должно отсекать решение текущей ослабленной задачи над Q и не отсекать при этом ни одного целочисленного решения исходной задачи. Процедура, предложенная Р. Е. Гомори, позволяет эффективно строить такого

[2] Данный факт можно определить за полиномиальное время, например при помощи алгоритма Хачияна или методов внутренних точек.

рода ограничения, гарантируя при этом решение исходной целочисленной задачи за конечное число шагов (полнота). Однако на практике многократное применение схемы Гомори приводит к чрезмерному росту базы ограничений и, как следствие, к необходимости использования различных техник отбрасывания нерелевантных ограничений-неравенств.

В 1965 году Б. Бухбергером был предложен алгоритм, позволяющий определять совместность систем полиномиальных уравнений над произвольным полем K [12, 13]. В основе данного алгоритма лежит обобщение процедуры деления с остатком на случай полиномов от многих переменных. Описание алгоритма Бухбергера (название, предложенное самим Бухбергером, — метод базисов Гребнера) довольно громоздко, поэтому ограничимся лишь акцентированием основных моментов.

Базовая идея метода Бухбергера состоит в упорядочении кольца полиномов $K\big[x_1,\ldots,x_n\big]$ специальным образом, результатом чего является отношение старшинства на множестве всевозможных мономов (например, при лексикографическом порядке $x_1 \succ \ldots \succ x_n$ любой моном, содержащий x_1, старше любого монома, не содержащего x_1). Такого рода порядок дает возможность построения процедуры исключения мономов, являющуюся обобщением обычного деления с остатком для полиномов от одной переменной. Например, если для пары полиномов $f,\ g \in K\big[x_1,\ldots,x_n\big]$ старший моном полинома g делит некоторый моном полинома f, то одно элементарное деление f на g определяется как переход от f к полиному $h = f - g\cdot u$, в котором нет монома w, $u \in K\big[x_1,\ldots,x_n\big]$. Данный факт обозначается через "$f \xrightarrow{g} h = f - g\cdot u$" (полином g называется элементарным делителем f). Будем говорить, что полином f сводится к полиному r относительно конечного множества полиномов $G \subset K\big[x_1,\ldots,x_n\big]$, если существует последовательность элементарных делений, начинающаяся с f и заканчивающаяся r, делителями в которой выступают полиномы из G:

$$f \xrightarrow{g_{i_1}} f_1 = f - g_{i_1}\cdot u_1 \xrightarrow{g_{i_2}} \ldots \xrightarrow{g_{i_t}} r\,.$$

Таким образом, $r = f - g_{i_1}\cdot u_1 - \ldots - g_{i_t}\cdot u_t$. Полином r, к которому нельзя применить процедуру элементарного деления ни для какого $g_{i_j} \in G$, называется остатком (его также называют несводимым относительно G). Несложно понять, что цепочка элементарных делений не может быть бесконечной. Таким образом, всякий полином f кольца $K\big[x_1,\ldots,x_n\big]$ может быть представлен в виде

$$f = r + g_{i_1}\cdot u_1 + \ldots + g_{i_t}\cdot u_t \tag{3}$$

относительно некоторого множества полиномов $G = \big\{g_1,\ldots,g_s\big\}$, $s \geqslant t$. Легко привести примеры таких полиномов $f \in K\big[x_1,\ldots,x_n\big]$ и множеств G, что представления вида (3) не единственны. Однако всегда существует такое конечное множество полиномов G, что любой полином f кольца $K\big[x_1,\ldots,x_n\big]$ может быть однозначно представлен в виде (3). Если I — некоторый ненулевой идеал кольца $K\big[x_1,\ldots,x_n\big]$, то конечное множество полиномов $G \subset I$, такое, что любой $f \in K\big[x_1,\ldots,x_n\big]$ допускает однозначное представление вида (3), $g_{i_j} \in G$, $j \in \big\{1,\ldots,t\big\}$, называется базисом (базой) Гребнера идеала I. Всякий ненулевой идеал упорядоченного специальным образом (например, лексикографически) кольца полиномов $K\big[x_1,\ldots,x_n\big]$ имеет базис Гребнера.

Пусть дана некоторая система полиномиальных уравнений над полем K вида

$$S = \Big\{P_i\big(x_1,\ldots,x_n\big) = 0\Big\}_{i=1}^{m}, \tag{4}$$

где $P_i\big(x_1,\ldots,x_n\big) \in K\big[x_1,\ldots,x_n\big]$, $i \in \big\{1,\ldots,m\big\}$. Множество полиномов $\Big\{P_i\big(x_1,\ldots,x_n\big)\Big\}_{i=1}^{m}$ порождает некоторый идеал I_S кольца $K\big[x_1,\ldots,x_n\big]$. Можно показать (следствие теоремы Гильберта о корнях), что явный вид базиса Гребнера идеала I_S позволяет ответить на вопрос о совместности системы (4). Бухбергером был предложен алгоритм, осуществляющий за конечное число шагов построение базиса Гребнера произвольного идеала I_S кольца $K\big[x_1,\ldots,x_n\big]$ по конечной системе порождающих этот идеал полиномов. Главным конструктивным моментом данного алгоритма является рекурсивная процедура пополнения текущего множества элементарных делителей новыми полиномами.

Все сказанное означает, что и в этом случае мы имеем дело с алгоритмом, основанном на идее пополнения базы ограничений (ограничениями здесь являются полиномиальные уравнения над полем K). Практическая реализация описанного алгоритма, как и в случаях, рассмотренных выше, требует процедур исключения нерелевантных ограничений.

Несложно показать, что задачу поиска выполняющего набора произвольной КНФ вида (1) можно свести к задаче поиска решения системы полиномиальных уравнений вида (4), левые части которых содержат полиномы Жегалкина (полиномы над полем GF(2)). Задачи криптоанализа некоторых систем шифрования допускают прямые сводимости к задачам поиска решений таких систем [14].

3. Крупноблочный параллелизм в булевых задачах; прогнозирование параметров распараллеливания. В данном разделе мы обобщаем технологию крупноблочного параллелизма в SAT-задачах, представленную в [15], на все типы булевых задач, описанных в предыдущем разделе.

Рассматриваем далее все перечисленные выше задачи в следующей общей постановке. Дан набор формул-условий вида

$$\Phi\big(x_{i_1^1},\ldots,x_{i_{r_1}^1}\big)\triangleright A_1,\ldots,\Phi\big(x_{i_1^m},\ldots,x_{i_{r_m}^m}\big)\triangleright A_m, \tag{5}$$

$\bigcup\limits_{j=1}^{m}\big\{x_{i_1^j},\ldots,x_{i_{r_j}^j}\big\}=\big\{x_1,\ldots,x_n\big\}=X$ — множество булевых переменных, $\triangleright\in\big\{=,\geqslant\big\}$, A_j, $j\in\big\{1,\ldots,m\big\}$ являются целыми числами. Формулы (5) — это выражения вида либо "КНФ=1", либо (2), либо полиномиальные уравнения над полем GF(2). Требуется найти вектор значений истинности переменных из X, удовлетворяющий всем условиям (5). Такой вектор будем называть решением системы (5).

Для произвольного множества $X^d=\big\{x_{i_1},\ldots,x_{i_d}\big\}$, $\big\{i_1,\ldots,i_d\big\}\subseteq\big\{1,\ldots,n\big\}$, определим подстановку набора значений переменных

$$x_{i_1}=\alpha_{i_1},\ldots,x_{i_d}=\alpha_{i_d},\quad \alpha_{i_j}\in\big\{0,1\big\},\quad j\in\big\{1,\ldots,d\big\}, \tag{6}$$

в произвольную формулу вида (5) как замену вхождения переменной x_{i_j}, $j\in\big\{1,\ldots,d\big\}$, константой α_{i_j} с последующим выполнением соответствующих преобразований (когда это возможно). Преобразования, являющиеся результатами применения подстановок типа (6) к выражениям вида (5), — это вычеркивания соответствующих литералов и дизъюнктов из КНФ, перенос целых чисел (с противоположным знаком) в правую часть линейных неравенств, а также вычеркивание мономов из булевых полиномов либо замена некоторых мономов на константу "1" с последующим сложением по mod 2 нескольких констант. Обозначим через 2^{X^d} множество, образованное всевозможными двоичными векторами длины d, каждый из которых определяет значения истинности переменных из X^d. Каждому вектору $\alpha=\big(\alpha_1,\ldots,\alpha_d\big)\in 2^{X^d}$ поставим в соответствие результат следующей подстановки в выражения (5): $x_{i_1}=\alpha_1,\ldots,x_{i_d}=\alpha_d$. Тем самым вектору $\alpha=\big(\alpha_1,\ldots,\alpha_d\big)\in 2^{X^d}$ сопоставляется набор преобразованных выражений типа (5), который будем обозначать через Φ^α. Сказанное означает, что множество X^d задает естественное соответствие между набором формул вида (5) и семейством формул $\big\{\Phi^\alpha\big\}_{\alpha\in 2^{X^d}}$. Каждая формула данного семейства есть либо логическое уравнение вида "КНФ=1", либо система линейных неравенств вида (2), либо система полиномиальных уравнений над полем GF(2). Будем также говорить, что множество X^d задает декомпозицию задачи решения системы (5) на семейство задач поиска решений систем $\big\{\Phi^\alpha\big\}_{\alpha\in 2^{X^d}}$. Дополнительно полагаем, что при $d=0$ результатом такой декомпозиции является исходная система типа (5).

Несложно видеть, что если двоичный вектор β — решение системы Φ^α при фиксированном $\alpha\in 2^{X^d}$, то компоненты векторов α и β дают компоненты некоторого решения исходной системы (5). Наоборот, если γ — произвольное решение (5), то часть его компонент определяет некоторый вектор $\alpha\in 2^{X^d}$, а оставшиеся компоненты определяют вектор β, являющийся решением системы Φ^α. Таким образом, задача поиска решения системы (5) сводится к задаче поиска решений систем Φ^α, $\alpha\in 2^{X^d}$, причем (5) не имеет решений (несовместна) тогда и только тогда, когда при всех $\alpha\in 2^{X^d}$ системы Φ^α несовместны.

Задачи поиска решений систем вида Φ^α — это, по сути, частные случаи исходной задачи, имеющие меньшую размерность, и для их решения можно использовать многопроцессорный вычислительный кластер. Более точно, упорядоченное некоторым образом семейство систем $\big\{\Phi^\alpha\big\}_{\alpha\in 2^{X^d}}$ может рассматриваться как очередь заданий для r-процессорного кластера. Именно такой подход был использован в [15] применительно к решению SAT-задач.

В этой же работе была предложена процедура статистического прогнозирования размерности множества X^d, т.е. значения параметра d, при котором суммарное время решения исходной задачи на вычислительном кластере минимально. Здесь мы переносим этот результат работы [15] на общую задачу поиска решений систем вида (5).

Предположим, что рассматривается некоторая задача типа (5). Через S обозначаем решатель, используемый для поиска ее решений. Решатель — это некоторая программа, использующая помимо базового алгоритма разнообразные эвристики, в том числе и эвристики удаления нерелевантных ограничений. Термин "прогнозная функция" обозначает семейство функций, параметрически зависящих от решателя S. Аргументами данных функций являются случайные выборки систем из множеств $\big\{\Phi^\alpha\big\}_{\alpha\in 2^{X^d}}$. Прогнозная функция является частично определенной на заданном множестве выборок и каждой "точке" своей области определения ставит в соответствие некоторое положительное рациональное число. Знание глобального минимума данной функции на ее области определения позволяет получить представление (прогноз) отно-

сительно минимального общего объема вычислений, требуемого для решения распараллеленной исходной задачи.

Случайные выборки из множеств $\{\Phi^\alpha\}_{\alpha\in 2^{X^d}}$ имеют смысл лишь в тех случаях, когда мощность данного множества (величина 2^d) слишком велика для его полного перебора. Поэтому при определении прогнозной функции имеет смысл рассматривать две ситуации: когда число 2^d больше, чем некоторая положительная константа ρ_0, и когда 2^d не превосходит ρ_0. За ρ_0 можно принять, например, число, сопоставимое с числом процессоров в кластере.

Далее полагаем, что все случайные выборки имеют фиксированный объем q. Каждому значению параметра $d \in \{0,1,\ldots,n\}$, такому, что $2^d > \rho_0$ ставится в соответствие множество векторов $\{\alpha^1,\ldots,\alpha^q\}$, выбираемых из 2^{X^d} в соответствии с заданным на нем равномерным распределением, а также множество (выборка) систем $\Theta_d = \{\Phi^{\alpha_1},\ldots,\Phi^{\alpha_q}\}$. Каждому значению параметра $d \in \{0,1,\ldots,n\}$, такому, что $2^d \leqslant \rho_0$, ставится в соответствие множество 2^{X^d} и множество систем $\{\Phi^\alpha\}_{\alpha\in 2^{X^d}}$.

Фиксируем некоторый решатель S. Обозначим через $t(\Phi')$ время работы (число битовых операций) решателя S на произвольном входе $\Phi' \in \Theta_d$. Введем в рассмотрение функцию $\tau_S(\Theta_d) = \sum\limits_{\Phi'\in\Theta_d} t(\Phi')$. Значением данной функции при каждом фиксированном $d \in \{0,1,\ldots,n\}$ является суммарное время (число битовых операций) работы решателя S по всем системам из Θ_d.

Следует учитывать, что при некоторых значениях параметра d (например, при $d = 0$) системы Θ_d могут оказаться очень сложными для решателя S; в этом случае время подсчета соответствующего значения прогнозной функции может превысить разумные границы. Для учета данного факта вводится в рассмотрение специальная функция $g(\Phi) = p(|\Phi|)$, где $p(\cdot)$ — некоторый полином, а через $|\Phi|$ обозначен объем двоичной кодировки исходной системы вида (5).

Допустим, что в соответствии с перечисленными правилами построено следующее семейство выборок $\Theta = \{\Theta_d\}_{d\in\{0,1,\ldots,n\}}$ (при фиксированном ρ_0). Определим прогнозную функцию в виде

$$T(\Theta_d) = \begin{cases} \dfrac{2^d}{q}\,\tau_S(\Theta_d), & 2^d > \rho_0, \quad \tau_S(\Theta_d) < g(|\Phi|), \\ \tau_S(\Theta_d), & 2^d \leqslant \rho_0, \quad \tau_S(\Theta_d) < g(|\Phi|), \\ \infty, & \tau_S(\Theta_d) \geqslant g(|\Phi|). \end{cases} \tag{7}$$

Запись "$T(\Theta_d) = \infty$" означает, что функция не определена на выборке Θ_d. Рациональное число $T(\Theta_d)$ является прогнозом общего объема битовых операций, требуемого для решения исходной SAT-задачи при декомпозиции исходной системы вида (5) на семейство систем $\{\Phi^\alpha\}_{\alpha\in 2^{X^d}}$. Тем самым задача прогнозного планирования оптимального по трудоемкости параллельного вычисления сводится к задаче минимизации функции T на множестве $\mathrm{dom}\, T \subseteq \Theta$. Знание глобального минимума функции T на $\mathrm{dom}\, T$ дает представление о возможности параллельного решения SAT-задачи для КНФ "за разумное время".

Далее приведем основной результат статьи [15], который напрямую переносится на более общий класс прогнозных функций, рассматриваемых в настоящей работе.

Теорема. *Пусть Φ — произвольная булева система вида (5), $\Theta = \{\Theta_d\}_{d\in\{0,1,\ldots,n\}}$ — произвольное семейство выборок систем из множества $\{\Phi^\alpha\}_{\alpha\in 2^{X^d}}$. Для любой прогнозной функции $T : \Theta \to Q$ вида (7) справедливы следующие свойства:*

— $\mathrm{dom}\, T \neq \emptyset$;

— глобальный минимум T на $\mathrm{dom}\, T$ находится в общем случае за время, ограниченное полиномом от $|\Phi|$.

В заключение отметим, что характерной особенностью предлагаемой технологии является то, что алгоритм решения исходной булевой задачи никак не адаптируется для решения задач из декомпозиционного семейства. Данный факт позволяет использовать на каждом узле кластера отлаженные и оттестированные на множестве примеров решатели. Свойство неполноты этих решателей, являющееся следствием процедур удаления нерелевантных ограничений, компенсируется уменьшением размерности решаемой задачи. Ниже данный тезис будет подкреплен результатами численных экспериментов.

4. Крупноблочный параллелизм в SAT-задачах, пакет D-SAT, численные эксперименты. В качестве частного случая технологии, описанной в разделе 3, рассматривается технология крупноблочного распараллеливания SAT-задач. В соответствии с данной технологией [15] осуществляется декомпозиция исходной КНФ C на семейство КНФ $C_1,\ldots,C_k$, где $k = 2^d$, $d \in N$. Во всех рассматриваемых SAT-задачах использовалась следующая (общая) схема формирования семейств множеств X^d: для лю-

бой КНФ C над упорядоченным множеством булевых переменных $X = \{x_1, \ldots, x_n\}$ определяем $X^0 = \emptyset$, $X^d = \{x_1, \ldots, x_d\}$, $d \in \{1, \ldots, n\}$. Таким образом, основным показателем эффективности декомпозиции является значение параметра d.

Для реализации данной технологии был разработан пакет прикладных программ Distributed-SAT (D-SAT). Пакет функционирует на вычислительном кластере под управлением инструментального комплекса discomp [16, 17].

Библиотека программ пакета D-SAT включает модуль расщепления, модуль SAT-решателя, модуль прогнозирования, модуль-анализатор и транспортный модуль. Модули пакета реализованы на языке C++, при этом все они являются платформо-независимыми.

Входные данные модуля расщепления — это файл с исходной КНФ в формате DIMACS, диапазон значений параметра d (левая и правая границы), а также фиксированное значение q, определяющее верхнюю границу объема случайной выборки КНФ. Обозначим через d_* и d^* натуральные числа, определяющие соответственно верхнюю и нижнюю границы интервала, в котором изменяются значения d. Для каждого значения диапазона параметров декомпозиции строится отдельное семейство КНФ. В случае $2^d > q$ из декомпозиционного семейства случайным образом выбираются q КНФ. Если же $2^d \leqslant q$, то выборкой является все рассматриваемое семейство. Результатом декомпозиции исходной КНФ является набор файлов с КНФ декомпозиционного семейства, представленными в формате DIMACS.

Вычислительное ядро модуля SAT-решателя составляет программа minisat версии 2.0 [8]. Основу данного решателя составляет подробно описанный выше алгоритм DPLL. По входному файлу, в котором КНФ представлена в формате DIMACS, модуль SAT-решателя осуществляет поиск выполняющего данную КНФ набора либо выдает значение *NULL* в случае отсутствия такового. В выходной файл каждой копии модуля SAT-решателя записывается информация о результатах его работы. Если в процессе обработки параллельного списка для какой-либо КНФ найден выполняющий набор, то все вычисления прекращаются и запускается модуль-анализатор.

Модуль прогнозирования обрабатывает файлы с результатами работы модуля SAT-решателя и производит их статистический анализ, на основе которого строится прогноз оптимальных параметров декомпозиции, при этом дополнительно учитываются все транспортные расходы.

Аналитический модуль проверяет найденный выполняющий набор на корректность, вычисляет среднее время решения SAT-задач для списка КНФ, минимальное и максимальное время, а также сравнивает прогнозируемое время решения с реальным временем. Решение SAT-задачи и результаты анализа записываются в итоговый файл.

Процесс решения SAT-задачи состоит из двух этапов. Первый — это этап прогнозирования оптимальных (с точки зрения вычислительных затрат) параметров декомпозиции. Второй — это этап решения всех SAT-задач из декомпозиционного семейства, определяемого найденными на этапе прогнозирования параметрами декомпозиции.

На этапе прогнозирования выполняются следующие шаги.

1. На вход модулю расщепления подается файл с исходной КНФ в формате DIMACS, значения d_*, d^* и q. Модуль расщепления для каждого значения d строит соответствующую выборку КНФ с учетом значения q. Полученные файлы КНФ в формате DIMACS объединяются в параллельный список и передаются на модуль SAT-решателя.

2. Для каждой КНФ данного списка решается соответствующая SAT-задача — доказывается невыполнимость либо находится выполняющий набор.

3. После того как решены SAT-задачи для всех КНФ из списка, при помощи модуля прогнозирования производится статистический анализ полученных результатов и на его основе вычисляется d_0 — прогнозное значение оптимального параметра декомпозиции.

В процессе прогнозирования существует возможность того, что будет найден набор, выполняющий исходную КНФ. В этом случае решение задачи считается найденным и вычислительный процесс завершается.

На рис. 3 изображена схема взаимодействия модулей пакета D-SAT на этапе прогнозирования.

На этапе решения задачи выполняются следующие шаги.

1. На вход модулю расщепления подается файл с исходной КНФ в формате DIMACS и полагается $d_* = d^* = d_0$, где значение d_0 найдено на этапе прогнозирования, а также $q = 0$. Модуль расщепления строит семейство файлов КНФ $C_1, \ldots, C_k$. Полученные КНФ объединяются в параллельный список и передаются на модуль SAT-решателя.

2. Для каждой КНФ полученного списка решается соответствующая SAT-задача — доказывается невыполнимость либо находится выполняющий набор.

3. Запускается модуль-анализатор.

На рис. 4 изображена схема взаимодействия модулей пакета D-SAT на этапе решения.

Обмен данными между вычислительными узлами кластера осуществляет специальный транспортный модуль пакета D-SAT, основной задачей которого является рассылка КНФ из параллельного списка. Каждая КНФ представляется в виде файла в формате DIMACS, состоящего из следующих трех частей:

— заголовок вида “p cnf [число переменных] [число дизъюнктов]”;

— список из d однолитеральных дизъюнктов, построенных по соответствующим значениям переменных декомпозиции;

— список дизъюнктов исходной КНФ.

В таком представлении у всех КНФ декомпозиционного семейства первая и третья части одинаковы, а различаются лишь вторые части. Назовем файлом различий файл со списком однолитеральных дизъюнктов произвольной КНФ семейства.

Работа транспортного модуля включает в себя этап препроцессинга, в ходе которого на каждый вычислительный узел передается файл с исходной КНФ. В ходе дальнейшей работы на вычислительные узлы передаются только файлы различий КНФ. Каждая отдельная SAT-задача формируется на основе информации из файла различий и файла с исходной КНФ. Описанная схема позволяет значительно сократить транспортные расходы.

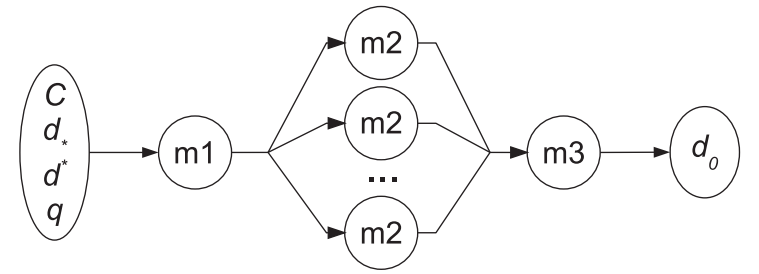

Рис. 3. Схема взаимодействия модулей пакета D-SAT на этапе прогнозирования

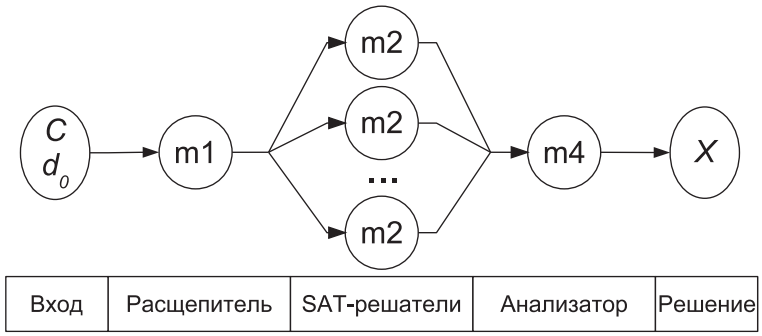

Рис. 4. Схема взаимодействия модулей пакета D-SAT на этапе решения

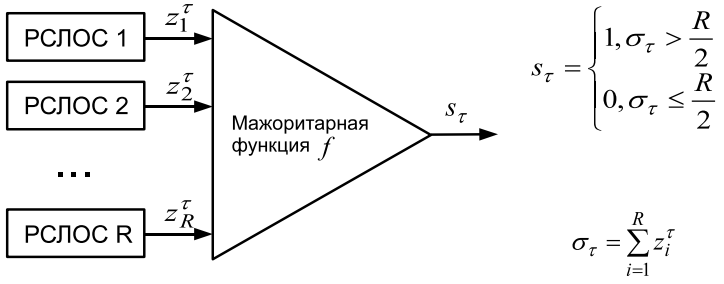

Рис. 5. Схема работы порогового генератора

Пакет D-SAT был использован в логическом криптоанализе порогового генератора. Базовые идеи логического криптоанализа, а также основные механизмы сводимости задач криптоанализа к SAT-задачам подробно описаны в [18]. Собственно пороговый генератор описан в работе [19]. Данный генератор использует принцип смешивания нескольких двоичных последовательностей, генерируемых линейными сдвиговыми регистрами (РСЛОС), посредством нелинейной булевой функции (рис. 5). В каждый момент времени τ , $\tau \in \{1, 2, \ldots\}$, биты, снимаемые синхронно с $R \geqslant 3$ регистров сдвига с линейной обратной связью (РСЛОС [20]), подаются на вход мажоритарной функции f, значением которой является бит выходной последовательности, имеющий номер τ. Мажоритарная функция выдает бит 1, если среди битов, поступивших ей на вход, большинство составляют единицы, и выдает бит 0 в противном случае. Параллельный логический криптоанализ был применен к пороговому генератору на основе следующих пяти РСЛОС:

1) (11, $X^{11} + X^{10} + X^8 + X^3 + 1$),
2) (13, $X^{13} + X^9 + X^8 + X^2 + 1$),
3) (15, $X^{15} + X^{14} + X^{12} + X^2 + 1$),
4) (16, $X^{16} + X^{14} + X^8 + X^3 + 1$),
5) (17, $X^{17} + X^{15} + X^{13} + X^{12} + X^{11} + X^{10} + 1$).

Длина инициализирующей последовательности — 72 бита. Анализировался фрагмент ключевого потока длиной 150 бит.

На рис. 6 показан пример графика оптимизации прогнозной функции для порогового генератора. Для рассматриваемого случая было найдено прогнозное значение параметра оптимальной декомпозиции $d_0 = 12$. Заштрихованные сеткой столбцы означают, что для соответствующих значений параметра d вычисление прогнозной функции было прервано, так как превышалось текущее пороговое значение [15].

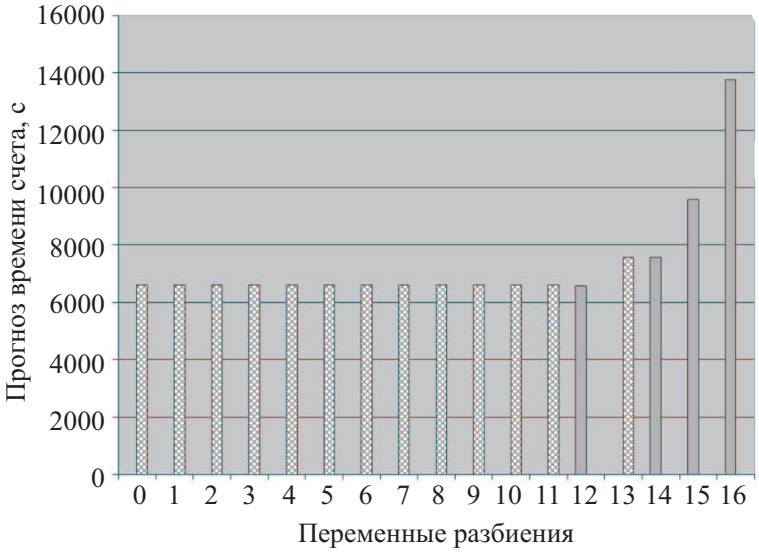

Рис. 6. Оптимизация прогнозной функции для порогового генератора с инициализирующей последовательностью длиной в 72 бита (один тест)

Параллельный криптоанализ порогового генератора осуществлялся на кластере Blackford Института динамики систем и теории управления (ИДСТУ) СО РАН [21]. Данный ресурс имеет следующие характеристики: пять вычислительных узлов; 10 двухядерных процессоров Intel E5060 Xeon Dual Core 3.2 GHz; суммарный объем оперативной памяти на узлах — 20 GB; суммарный объем дисковой памяти на узлах — 800 GB; интерконнект — Gigabit Ethernet.

Последовательный криптоанализ порогового генератора осуществлялся на одном ядре процессора Intel E5060 Xeon Dual Core 3.2 GHz (см. таблицу).

Результаты криптоанализа порогового генератора (серия тестов); инициализирующая последовательность 72 бита, выходная последовательность 150 бит

Время последовательного криптоанализа	Прогноз времени параллельного криптоанализа	Реальное время параллельного криптоанализа
> 28 суток (вычисление не закончено)	1 ч. 20 мин. – 2 ч.	39 мин. – 3 ч. 17 мин.

Особо отметим тот факт, что ускорение, полученное при решении рассмотренной выше задачи на 20-ядерном кластере Blackford, превосходит время ее решения на одном ядре данного кластера существенно более, чем в 20 раз.

СПИСОК ЛИТЕРАТУРЫ

1. *Яблонский С.В.* Введение в дискретную математику. М.: Наука, 1986.
2. *Cook S.A.* The complexity of theorem-proving procedures// Proc. of the Third Ann. ACM Symp. on Theory of Computing. Shaker Heights (Ohio, USA), 1971. 151–158 (Русский перевод: Кук С.А. Сложность процедур вывода теорем. Кибернетический сборник: Новая серия. 1975. Вып. 12. 5–15).
3. *Davis M., Longemann G., Loveland D.* A machine program for theorem proving // Communications of the ACM. 1962. **5**. 394–397.
4. *Devis M., Putnam H.* A computing procedure for quantification theory // J. of ACM. 1960. **7**. 201–215.
5. *Marqeus-Silva J.P., Sakallah K.A.* GRASP: a search algorithm for propositional satisfiability // IEEE Trans. on Computers. 1999. **48**, N 5. 506–521.
6. Zchaff (http://www.princeton.edu/ chaff/zchaff.html).
7. Berkmin (http://eigold.tripod.com/BerkMin.html).
8. MiniSat (http://minisat.se/MiniSat.html).
9. *Robinson J.A.* A machine-oriented logic based on the resolution principle // J. ACM. 1965. **12**, N 1. 23–41.
10. *Чень Ч., Ли Р.* Математическая логика и автоматическое доказательство теорем. М.: Наука, 1983.
11. *Пападимитриу Х., Стайглиц К.* Комбинаторная оптимизация. Алгоритмы и сложность. М.: Мир, 1985.

12. *Buchberger B.* Groebner bases: an algorithmic method in polynomial ideal theory // Recent Trends in Multidimensional System Theory. Dordrecht: Reidel Publishing Company, 1985. 184–232.
13. *Кокс Д., Литтл Дж., О'Ши Д.* Идеалы, многообразия и алгоритмы. М.: Мир, 2000.
14. *Агибалов Г.П.* Логические уравнения в криптоанализе генераторов ключевого потока // Вестник Томского гос. ун-та. Приложение. 2003. № 6. 31–41.
15. *Заикин О.С., Семенов А.А.* Технология крупноблочного параллелизма в SAT-задачах // Проблемы управления. 2008. № 1. 43–50.
16. *Заикин О.С., Семенов А.А., Сидоров И.А., Феоктистов А.Г.* Параллельная технология решения SAT-задач с применением пакета прикладных программ D-SAT // Вестник Томского гос. ун-та. Приложение. 2007. № 23. 83–95.
17. *Заикин О.С., Сидоров И.А.* Технология крупноблочного распараллеливания в криптоанализе некоторых генераторов двоичных последовательностей // Труды международной научной конференции ПАВТ'07. Челябинск, ЮУрГУ, 2007. **1**. 158–169.
18. *Семенов А.А.* Логико-эвристический подход в криптоанализе генераторов двоичных последовательностей // Труды международной научной конференции ПАВТ'07. Челябинск, ЮУрГУ, 2007. **1**. 170–180.
19. *Bruer J.O.* On pseudo random sequences as crypto generators // Proc. Int. Zurich Seminar on Digital Communication. Zurich (Switzerland), 1984. 157–161.
20. *Menezes A., Van Oorschot P., Vanstone S.* Handbook of Applied Cryptography. Boca Raton: CRC Press, 1997.
21. Суперкомпьютерный центр ИДСТУ СО РАН (http://www.mvs.icc.ru).

Поступила в редакцию
27.03.2008