OSTIS-2015
(Open Semantic Technologies for Intelligent Systems)

УДК 004.89

ЭВРИСТИЧЕСКИЙ МЕТОД УДОВЛЕТВОРЕНИЯ ОГРАНИЧЕНИЙ НА ОСНОВЕ ИХ МАТРИЧНОГО ПРЕДСТАВЛЕНИЯ

Зуенко А.А.[*], Очинская А.А.[**]

[*] *Институт информатики и математического моделирования КНЦ РАН, г. Апатиты, Россия*
zuenko@iimm.ru

[**] *Кольский филиал Петрозаводского государственного университета, г. Апатиты, Россия*
ochinsk@yandex.ru

В статье описан разработанный авторами метод эвристического поиска, предназначенный для решения задач удовлетворения ограничений. Метод опирается на применение матрицеподобных структур алгебры кортежей, которые позволяют представлять и эффективно обрабатывать ограничения с конечными доменами. В отличие от большинства аналогов сложность метода зависит не от количества значений в доменах переменных и суммарного числа переменных в исследуемой системе ограничений, а определяется количеством уравнений и средним числом используемых в них переменных.
Ключевые слова: эвристический поиск, задача удовлетворения ограничений; алгебра кортежей.

Введение

В современном программировании можно выделить несколько основных парадигм: императивное или алгоритмическое программирование, логическое программирование, функциональное программирование и др. Важное место в этом ряду занимает программирование в ограничениях (constraint programming) [Рассел и др., 2006], [Bartak, 1999].

Программирование в ограничениях как самостоятельное научное направление сложилось в конце 60-х – начале 70-х годов прошлого века. Первыми приложениями были задачи обработки изображений и параметрического моделирования пространственно-двумерных сцен. С тех пор направление существенно эволюционировало, охватывая новые классы задач, начиная от решения судоку и ребусов и заканчивая решением систем линейных уравнений и задач искусственного интеллекта. Например, в [Ченцов, 2013] рассматриваются задачи маршрутизации перемещений с ограничениями в виде условий предшествования, причем допускаются также ограничения других типов, диктуемые особенностями прикладных задач. В частности, описывается задача, связанная со снижением облучаемости при работе в радиационных полях, а также задача листовой резки деталей на станках с числовым программным управлением.

Парадигма удовлетворения ограничений является обобщением методов пропозициональной логики на случай, когда переменные могут принимать не одно из двух фиксированных значений (“истина”/”ложь”), а любые значения из некоторой конечной области определения.

Программирование в ограничениях отличается от традиционного логического программирования, в первую очередь, тем, что решение задачи рассматривается не как доказательство истинности/ложности некоторого утверждения, а сводится к поиску вектора тех значений переменных, которые удовлетворяют условиям задачи. В большинстве случаев при решении задач удовлетворения ограничений (Constraint Satisfaction Problem – CSP) требуется получить не все возможные решения задачи, а найти хотя бы одно из них. Система программирования в ограничениях, как правило, стремится сократить перебор вариантов с целью минимизации отката в случае неуспеха. Для ускорения вычислительных процедур могут быть использованы различные методы, в частности эвристические.

Задачи удовлетворения ограничений относятся к классу задач комбинаторного поиска. Самый общий подход к решению подобных задач основан на обходе дерева поиска. Процессу поиска придается рекурсивный характер, при котором на каждом шаге текущая задача заменяется несколькими подзадачами, рассматриваемыми затем в некотором

порядке, причем менее перспективные из них могут отбрасываться. Таким образом, при обходе дерева поиска переплетаются процессы декомпозиции возникающих ситуаций и их редуцирования. Вполне оправдано стремление к декомпозиции на возможно меньшее число подзадач и к редуцированию, по возможности, более глубокому. Поэтому основные методы решения задач удовлетворения ограничений могут быть разбиты на два класса [Ruttkay, 1998]. Первый класс содержит различные варианты алгоритмов поиска в глубину с возвратами, которые строят решение путем расширения частичного присваивания шаг за шагом, используя различные эвристики и применяя разумные стратегии возврата из тупиковых вершин. Ко второму классу относятся алгоритмы распространения ограничений, которые исключают из пространства поиска некоторые элементы, не входящие в решение, обеспечивая снижение размерности задачи.

В данной работе предлагается моделировать ограничения в виде матрицеподобных структур алгебры кортежей (АК) [Кулик и др., 2010], [Зуенко и др., 2011], а также рассматривается разработанный авторами метод вывода на ограничениях, использующий оригинальные эвристики для эффективного поиска решений задачи CSP. В отличие от большинства аналогов применение АК-объектов позволяет обойтись без предварительной бинаризации системы ограничений. По сравнению с ранее предложенными одним из авторов методами вывода в структурах АК [Зуенко, 2013], [Зуенко, 2014] предлагаемый вниманию метод строит дерево поиска, сопоставляя его уровням не переменные (атрибуты), а уравнения системы ограничений.

Далее приведем некоторые сведения из алгебры кортежей в объеме, необходимом для дальнейшего изложения.

1. Представление ограничений в виде структур алгебры кортежей

В [Кулик и др., 2010] приводятся основы АК и демонстрируется ее применение для унификации представления и обработки различных видов данных и знаний, а также решения различных задач логического и логико-вероятностного анализа. Близкий подход применяется также в [Zakrevskij, 2012] для решения задач распознавания образов и упрощения баз знаний.

Конечные предикаты в АК сжато описываются с помощью двух типов структур: *C*-систем и *D*-систем. Конкретные экземпляры этих структур называются АК-объектами. Системы ограничений с конечными доменами, обычно, удобно представлять в виде *D*-систем, а корни системы ограничений (решения) искать в виде *C*-систем. В АК имеются алгоритмы преобразования *D*-систем в *C*-системы и наоборот.

Рассмотрим *D*-систему *A*, содержащую *n* *D*-кортежей (строк) и *m* атрибутов (столбцов):

$$A[X_1X_2\ \ldots\ X_m] = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{bmatrix},$$

где $a_{k,l}$ – компоненты *D*-системы *A*, удовлетворяющие ее схеме, среди которых могут встречаться и пустые компоненты.

Известно [Кулик и др., 2010], что *D*-система эквивалентна пересечению *n* *C*-систем, каждая из которых сопоставляется одной из строчек исходной *D*-системы. Множество *C*-систем, полученных в результате преобразования каждого *D*-кортежа из *A* в *C*-систему, обозначим как $\{C_i\}=\{C_1, C_2, \ldots, C_n\}$. Причем, количество непустых строк в каждой *C*-системе совпадает с количеством непустых компонент соответствующей строки *D*-системы, то есть, если $a_{k,l} = \emptyset$, то строка с номером *l в C*-системе с номером *k* будет пуста. С учетом вышеизложенного имеем:

$$A[X_1X_2\ldots X_m] \equiv \bigcap_{i=1}^{n} C_i \text{ , где}$$

$$C_i[X_1X_2\ldots X_m] = \begin{bmatrix} a_{i,1} & * & \cdots & * \\ * & a_{i,2} & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & a_{i,m} \end{bmatrix}.$$

Заметим, что строки *C*-систем $\{C_1, C_2, \ldots, C_n\}$ содержат по одной нефиктивной компоненте. Другими словами, каждая компонента *i*-ой строки *D*-системы *A* определяет некоторую строку в *C*-системе $C_i \in \{C_1, C_2, \ldots, C_n\}$.

Приведем конкретный пример. Пусть имеется *D*-система *P*, заданная в $\boldsymbol{S} = \{a, b, c, d\}^3$:

$$P = \begin{bmatrix} \{a,c\} & \{d\} & \{b,d\} \\ \varnothing & \{a,d\} & \{a,c\} \\ \{b,c\} & \varnothing & \{b\} \end{bmatrix}.$$

Рассмотрим преобразование *D*-системы в *C*-систему:

$$P = \begin{bmatrix} \{a,c\} & * & * \\ * & \{d\} & * \\ * & * & \{b,d\} \end{bmatrix} \cap \begin{bmatrix} \varnothing & \varnothing & \varnothing \\ * & \{a,d\} & * \\ * & * & \{a,c\} \end{bmatrix} \cap \begin{bmatrix} \{b,c\} & * & * \\ \varnothing & \varnothing & \varnothing \\ * & * & \{b\} \end{bmatrix}.$$

Вычисляя пересечение первых двух *C*-систем (пустые *C*-кортежи здесь и далее для экономии места не показаны), получим:

$$\begin{bmatrix} \{a,c\} & \{a,d\} & * \\ \{a,c\} & * & \{a,c\} \\ * & \{d\} & * \\ * & \{d\} & \{a,c\} \\ * & \{a,d\} & \{b,d\} \end{bmatrix} \begin{matrix} (1,2) \\ (1,3) \\ (2,2) \\ (2,3) \\ (3,2) \end{matrix}.$$

Компоненты вектора чисел означают номера перемножаемых строк первой и второй матриц.

Теперь найдем пересечение полученной промежуточной *C*-системы с третьей *C*-системой:

$$\begin{bmatrix} \{c\} & \{a,d\} & * \\ \{a,c\} & \{a,d\} & \{b\} \\ \{c\} & * & \{a,c\} \\ \{b,c\} & \{d\} & * \\ * & \{d\} & \{b\} \\ \{b,c\} & \{d\} & \{a,c\} \\ \{b,c\} & \{a,d\} & \{b,d\} \\ * & \{a,d\} & \{b\} \end{bmatrix} \begin{matrix} (1,2,1) \\ (1,2,3) \\ (1,3,1) \\ (2,2,1) \\ (2,2,3) \\ (2,3,1) \\ (3,2,1) \\ (3,2,3) \end{matrix}.$$

Компоненты вектора чисел, по-прежнему, означают номера строк перемножаемых матриц (в данном случае перемножаются три матрицы).

Принимая во внимание особенности преобразования *D*-систем в *C*-системы, можно заранее по виду исходной *D*-системы *A* посчитать количество строк результирующей *C*-системы в наихудшем случае (когда не появляется пустых строк). Для этого достаточно перемножить количества непустых компонент, находящихся в каждой строке *D*-системы *A*. Для нашего примера имеем: $l = 3*2*2=12$.

2. Метод лексикографического перебора

При последовательном перемножении *C*-систем промежуточная *C*-система может оказаться слишком большой, и ее трудно зафиксировать в памяти компьютера. Рассмотрим метод, позволяющий не хранить в памяти большие объемы промежуточных результатов. По сути, предлагаемый метод является развитием метода, предложенного в [Закревский, 2003]. Его отличие состоит в том, что в перемножении (пересечении) участвуют не дизъюнктивные нормальные формы (ДНФ) булевых функций, а дизъюнктивные нормальные формы конечных предикатов. К тому же, перемножаемые ДНФ имеют определенный вид, а конечная цель перемножения – преобразование конъюнктивной нормальной формы конечного предиката (*D*-системы) в его ДНФ (*C*-систему).

Метод состоит в лексикографическом переборе комбинаций строк, принадлежащих различным пересекаемым *C*-системам $\{C_1, C_2, \ldots, C_n\}$, причем множество *C*-систем $\{C_1, C_2, \ldots, C_n\}$ заранее упорядочено. Рассмотрим метод более детально.

Ранее при перемножении *C*-систем каждой строке результирующей *C*-системы приписывался определенный вектор чисел $\boldsymbol{a}$, т.е. линейно упорядоченное множество компонент a_i, где $i = 1, 2, \ldots, p$, p – количество перемножаемых на данном шаге *C*-систем. Каждая из компонент принимала значения из некоторого линейно упорядоченного множества A_i – множества индексов непустых строк *C*-системы C_i.

Теперь пусть имеется произвольная *D*-система *P* размерности $n \times m$. Пронумеруем строки (кортежи) и столбцы (атрибуты) этой *D*-системы. Строке с номером i сопоставим линейно упорядоченное множество A_i – множество номеров столбцов, содержащих непустые компоненты в данной строке. Рассмотрим векторы вида $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$, где $a_i \in A_i$ – номера непустых компонент i-ой строки *D*-системы. Заметим, что размерность вектора равна количеству кортежей *D*-системы (количеству перемножаемых *C*-систем). Каждый такой вектор определяет либо одно из решений системы ограничений, представленной в виде *D*-системы, либо пустой кортеж. Действительно, если пересечь строки a_1 из C_1, a_2 из C_2, …, a_n из C_n, то получим некоторую строку результирующей *C*-системы или пустую строку.

Будем считать, что $\boldsymbol{a} < \boldsymbol{a}'$, если существует такое i, что $a_i < a_i'$ и $a_j = a_j'$ для любого j, такого, что $j<i$, причем $i = 1, 2, \ldots, n$.

Для упорядоченного таким способом множества всевозможных значений вектора $\boldsymbol{a}$ введем следующее обозначение: $\mathrm{lex}(A_1, A_2, \ldots, A_n)$.

На основе лексикографического упорядочения значений вектора $\boldsymbol{a}$, то есть множества потенциальных решений, и строится дерево поиска.

Лексикографический перебор может быть сокращенным, когда перебираются не все значения вектора $\boldsymbol{a}$, а лишь часть из них. Однако порядок при этом не нарушается: каждое последующее из рассматриваемых значений вектора $\boldsymbol{a}$ должно быть больше предыдущего. Именно сокращенный перебор и будет нас интересовать, поскольку нет необходимости рассматривать некоторые участки полной лексикографически упорядоченной последовательности значений вектора $\boldsymbol{a}$, в которых заведомо не содержится решений поставленной задачи.

На очередном шаге рассматривается некоторое частичное значение вектора $\boldsymbol{a}$, где определены лишь значения компонент $a_1, a_2, \ldots, a_i$, а значения других компонент $a_{i+1}, a_{i+2}, \ldots, a_p$ остаются неопределенными. Неопределенность будем отмечать символов «–». Если окажется, что пересечение компонент, соответствующих указанным в векторе $\boldsymbol{a}$ индексам, на данном шаге пусто, то очевидно, что всякое расширение рассматриваемой комбинации путем подбора новых

компонент из оставшихся кортежей *D*-системы, так же даст пустое пересечение. Поэтому можно не перебирать все возможные варианты таких расширений, а изменить текущее частичное значение вектора ***a***, выбрав следующее возможное значение для компоненты a_i.

Может оказаться, что компонента a_i уже обладает последним из возможных значений. В таком случае следует сократить рассматриваемую частичную комбинацию, выбросив из нее последний элемент (значение i уменьшается на единицу), и выбрать следующее возможное значение для предпоследней компоненты.

Если же пересечение компонент, соответствующих рассматриваемой частичной комбинации, не пусто, то дополняем ее индексом первой непустой компоненты следующего кортежа *D*-системы, и переходим к рассмотрению полученной новой комбинации.

Очевидно, что решение может быть найдено лишь на том шаге, когда $i=p$.

Рассмотрим, как строится дерево поиска для примера, разобранного в предыдущем подразделе.

Запишем возможные значения вектора ***a***, то есть lex(A_1, A_2, A_3). Учитывая, что $A_1 = \{1,2,3\}$, $A_2 = \{2,3\}$, $A_3 = \{1,2\}$, имеем : (1, 2, 1), (1, 2, 3), (1, 3, 1), (1, 3, 3), (2, 2, 1), (2, 2, 3), (2, 3, 1), (2, 3, 3), (3, 2, 1), (3, 2, 3), (3, 3, 1), (3, 3, 3).

Сам описанный процесс поиска решений задачи CSP для данного примера может быть представлен с помощью таблицы 1, где знаком (“+”) отмечены решения задачи CSP.

Таблица 1 – Лексикографический перебор решений CSP

значение вектора ***a***	частичное/полное решение задачи CSP
1 – –	[{*a*, *c*} * *]
1 2 –	[{*a*, *c*} {*a*, *d*} *]
1 2 1	[{*c*} {*a*, *d*} *]+
1 2 3	[{*a*, *c*} {*a*, *d*} {*b*}]+
1 3 –	[{*a*, *c*} * {*a*, *c*}]
1 3 1	[{*c*} * {*a*, *c*}]+
1 3 3	∅
2 – –	[* {*d*} *]
2 2 –	[* {*d*} *]
2 2 1	[{*b*, *c*} {*d*} *]+
2 2 3	[* {*d*} {*b*}]+
2 3 –	[* {*d*} {*a*, *c*}]
2 3 1	[{*b*, *c*} {*d*} {*a*, *c*}]+
2 3 3	∅
3 – –	[* * {*b*, *d*}]
3 2 –	[* {*a*, *d*} {*b*, *d*}]
3 2 1	[{*b*, *c*} {*a*, *d*} {*b*, *d*}]+
3 2 3	[* {*a*, *d*} {*b*}]+
3 3 –	∅

Из строк таблицы, помеченных знаком “+”, получаем ту же итоговую *C*-систему, что и в предыдущем подпункте.

3. Динамическое формирование дерева поиска

Часто нет необходимости искать все возможные решения поставленной задачи, а достаточно найти лишь одно из них. В этом случае можно воспользоваться описанным ранее методом с учетом того, что перебор будет прекращен при первом найденном решении. Однако для поиска одного корня *D*-системы метод лексикографического перебора недостаточно эффективен. Дело в том, что в рассмотренном методе дерево поиска формируется статически: каждому уровню дерева заранее приписывается некоторая строка *D*-системы (уравнение системы ограничений) и заранее установлен порядок просмотра компонент выбранной строки *D*-системы. Другими словами, метод лексикографического перебора относится к классу методов “слепого” поиска и обладает всеми недостатками методов данного класса.

Как уже упоминалось во введении, предлагаемый в работе метод строит дерево поиска, сопоставляя его уровням не переменные (атрибуты), а уравнения системы ограничений. Каждый конкретный узел в пределах уровня соответствует некоторой непустой компоненте выбранной строки.

В настоящей статье предлагается для выбора преемника текущего узла дерева поиска использовать эвристики. Это позволяет отложить исследование менее перспективных ветвей дерева поиска, отдав предпочтение более перспективным с точки зрения получения скорейшего решения. Сам перебор при этом престает быть лексикографическим, так как каждое последующее рассматриваемое значений вектора ***a*** может как-угодно отличаться от предыдущего.

Текущее состояние решаемой задачи CSP полностью характеризуется частичным решением, записанным в виде набора пар “атрибут – усеченный домен атрибута”, а также остатком *D*-системы, получаемым в ходе процедуры распространения ограничений на основе правил редукции, подробно представленных в работе [Зуенко, 2014].

Здесь, ввиду требований к объему работы, правила редукции не приводятся, а основное внимание уделяется эвристикам, применяемым в процессе поиска. Одни эвристики служат для выбора строки *D*-системы, другие для выбора компоненты строки, включаемой в решение.

Э1. Выбираем строку *D*-системы с наименьшим числом компонент.

Э2. В случае неоднозначности выбора, производимого согласно **Э1**, выбираем уравнение (строку), которое содержит наименьшее число корней. Причем, подсчет числа корней выполняется в предположении, что количество переменных уравнения равно числу непустых компонент соответствующей строки *D*-системы, а не общему числу атрибутов *D*-системы.

Э3. Выбираем ту компоненту, которая при предварительной проверке наименее ограничивает диапазон возможных значений для других атрибутов**.**

Э4. В случае неоднозначности выбора, производимого на основе **Э3**, находим компоненту, которая позволяет элиминировать из *D*-системы наибольшее количество строк.

Так как значения вектора ***a*** при данном подходе не будут лексикографически упорядочены, то требуется дополнительная переменная для хранения уже рассмотренных значений вектора ***a***. Это позволяет в случае обнаружения тупиковой ветви дерева поиска совершать возврат без повторного рассмотрения значений вектора ***a***.

Рассмотрим особенности применения этого метода на примере. Пусть имеется *D*-система, заданная в $\boldsymbol{S} = X_1 \times X_2 \times X_3 \times X_4 = \{1, 2, 3, 4\}^4$:

$$\begin{array}{c|cccc} 1 & \{2,3\} & \varnothing & \{1,2,3\} & \varnothing \\ 2 & \{2\} & \{1,2,5\} & \{4\} & \varnothing \\ 3 & \varnothing & \{1,2,4\} & \{3,4\} & \{3,5\} \\ 4 & \{1,2\} & \{3,4,5\} & \varnothing & \{3\} \\ 5 & \varnothing & \varnothing & \{1,5\} & \{4\} \\ 6 & \varnothing & \{2,3\} & \{1,2,3\} & \{1\} \\ 7 & \{2,4,5\} & \varnothing & \{3,5\} & \{3,5\} \\ 8 & \{5\} & \varnothing & \{2,3,4\} & \{1,3,4\} \\ 9 & \{1,3\} & \varnothing & \{3,4,5\} & \varnothing \end{array}$$

Воспользуемся эвристикой **Э1**. Строки с номерами 1, 5, 9 имеют по две непустых компоненты. Оценим число корней уравнений, задаваемых этими строками, как того требует **Э2.** Строка №1 может быть представлена как следующая ортогональная *C*-система в подпространстве $\boldsymbol{S}_1 = X_1 \times X_3 = \{1, 2, 3, 4\}^2$:

$$\begin{bmatrix} \{2,3\} & * \\ \{1,4\} & \{1,2,3\} \end{bmatrix} = \begin{bmatrix} \{2,3\} & \{1,2,3,4\} \\ \{1,4\} & \{1,2,3\} \end{bmatrix}.$$

Для данной *C*-системы число корней подсчитывается очевидным образом:

$$k_1 = 2*4+2*3=14.$$

Для пятой и девятой строк исходной *D*-системы имеем, соответственно: $k_5 = 10$; $k_9 = 14$. Значит, выбираем строку №5 и начинаем просматривать ее компоненты. Сначала берем компоненту №3, полагая ее новым доменом атрибута X_3. Но при таком выборе нельзя, воспользовавшись правилами редукции из работы [Зуенко, 2014], сузить области возможных значений ни для одного атрибута, отличного от третьего. Аналогично, обстоят дела и при проверке компоненты №4. Следовательно, применение эвристики **Э3** не позволяет осуществить выбор компоненты.

Теперь применим **Э4**. При выборе компоненты №4 из *D*-системы, помимо самой выбранной строки №5, уходит строка № 8. Действительно четвертая компонента восьмой строки поглощает четвертую компоненту пятой строки: $\{4\} \subseteq \{1,3,4\}$. Учитывая то, что компонента {4} становится новым доменом атрибута X_4, то четвертые компоненты обеих этих строк становятся полными, а строки *D*-системы, содержащие полные компоненты могут быть удалены [Кулик и др., 2010]. Итак, мы описали один шаг поиска, остальные шаги выполняются по аналогии.

Процесс поиска решения отображен в таблице 2. Каждый шаг поиска представлен отдельной строкой, где записаны два вектора: вектор ***a*** и вектор частичного решения задачи CSP. В отличие от таблицы 1, теперь на каждом шаге поиска для вектора ***a*** может быть конкретизирована более чем одна позиция. Как и ранее, позиции вектора ***a*** соответствуют строкам исходной *D*-системы, но значения, находящиеся в этих позициях, теперь интерпретируются как номера компонент, которые становятся полными на текущем шаге поиска в результате осуществления выбора.

Таблица 2 – Перебор уравнений (ограничений) с использованием эвристик

значение вектора ***a***	частичное/полное решение задачи CSP
– – – – **4** – – 4 –	[* * * {4}]
3 – – – 4 3 – 4 –	[* * {1, 2, 3} {4}]
3 – 3 – 4 3 3 4 **3**	[* * {3} {4}]
3 1 3 1 4 3 3 4 **3**	[{2} * {3} {4}]

В каждой строке таблицы 2 в точности одна позиция вектора ***a*** выделена жирным шрифтом. В этой позиции содержится номер выбранной на текущем шаге поиска компоненты *D*-системы. Сама позиция указывает на строку, где эта компонента находится.

Заключение

Предложены оригинальные эвристики, отличающиеся от принятых в теории удовлетворения ограничений. Оказалось, что динамически управляя выбором строк и компонент *D*-системы на каждом шаге поиска, то есть формируя дерево поиска на основе анализа текущего состояния задачи, можно существенно снизить вычислительную сложность процедуры поиска решений задачи CSP, сводя к минимуму вероятность выполнения возвратов.

Работа выполнена при поддержке РФФИ (проекты № 13-07-00318-а, № 14-07-00205-а, 14-07-00256-а, 14-07-00257-а), Президиума РАН (проект 4.3 Программы № 15), ОНИТ РАН (проект 2.3 в рамках текущей Программы фундаментальных научных исследований**).**

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

[Закревский, 2003] Закревский, А. Д. Логические уравнения / А. Д. Закревский //– 2-е изд., стереотип. – М. : Эдиториал УРСС, 2003 . – 96 с.

[Зуенко и др., 2011] Зуенко, А.А. Интеграция баз данных и знаний интеллектуальных систем на основе алгебраического подхода / А.А. Зуенко, Б.А. Кулик, А.Я. Фридман // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2011): материалы Междунар. научн.-техн. конф. (Минск, 10-12 февраля 2011 г.) – Минск: БГУИР, 2011. – С.59-70.

[Зуенко, 2013] Зуенко, А.А. Матрицеподобные вычисления в задачах удовлетворения огра-ничений /А.А. Зуенко // Шестая Всероссийская мультиконференция по проб-лемам управления, 30 сентября – 5 октября 2013 г., г. Геленджик, с. Дивно-морское: материалы мультиконференции в 4 т. – Ростов-на-Дону: Изд-во Южного федерального университета, 2013. -Т.1. – С.30-34.

[Зуенко, 2014] Зуенко, А.А. Вывод на ограничениях с применением матричного представления конечных предикатов / А.А. Зуенко // Искусственный интеллект и принятие решений, 2014. – Вып. 3. – С.21-31.

[Кулик и др., 2010] Кулик, Б.А. Алгебраический подход к интеллектуальной обработке данных и знаний / Б.А. Кулик, А.А. Зуенко, А.Я. Фридман. – СПб.: Изд-во Политехн. ун-та, 2010. – 235 с.

[Рассел и др., 2006] Рассел, С. Искусственный интеллект: современный подход. 2-е изд. / С. Рассел, П. Норвиг // пер. с англ.; ред. К.А. Птицына. – М.: Изд. дом «Вильямс», 2006. -1408 с.

[Ченцов, 2013] Ченцов, А.Г. Экстремальные задачи маршрутизации: теория и приложения /А.Г. Ченцов // Шестая Всероссийская мультиконференция по проблемам управления, г. Геленджик, с. Дивноморское, 30 сентября – 5 октября 2013 г.: материалы мультиконференции: в 4 т. – Ростов-на-Дону: Изд-во Южного федерального университета, 2013. -Т.3. – С.213-220.

[Bartak, 1999] Bartak R. Constraint Programming: In Pursuit of the Holy Grail // Proceedings of the Week of Doctoral Students (WDS99), Part IV. – Prague: MatFyzPress, 1999. P. 555–564.

[Ruttkay, 1998] Ruttkay Zs. Constraint satisfaction a survey // CWI Quarterly.1998. V. 11. P. 163–214.

[Zakrevskij, 2012] Arkadij Zakrevskij. Integrated Model of Inductive-Deductive Inference Based on Finite Predicates and Implicative Regularities. In: “Diagnostic Test Approaches to Machine Learning and Commonsense Reasoning Systems”, IGI Global, P 1-12.

HEURISTIC METHOD OF CONSTRAINT SATISFACTION BASED ON MATRIX REPRESENTATION OF CONSTRAINTS

Zuenko A.A. [*], Ochinskaya A.A. [**]

** Institute for Informatics and Mathematical Modelling, Kola Science Centre of RAS, Apatity, Russia*

zuenko@iimm.ru

*** Kola Branch of Petrozavodsk State University, Apatity, Russia*

ochinsk@yandex.ru

This paper describes a heuristic search method developed by the authors for solving of constraint satisfaction problems. The method relies on the using of matrix structures of n-tuple algebra (NTA). The structures allow to represent and to effectively handle constraints with finite domains. The computational complexity of the method is determined by the number of logical equations and the average number of variables in equations.

Introduction

There are several major paradigms in modern programming: the algorithmic programming, logic programming, functional programming, and etc. The constraint programming takes an important place in this series.

In this paper we propose to model constraints in the form of matrix structures of n-tuple algebra: *C*-systems and *D*-systems. A method developed by the authors for constraint inference uses the original heuristics for effective search of solutions of the constraint satisfaction problem.

Unlike most analogues using of NTA-objects provide an opportunity to solve constraint satisfaction problem without reducing constraint set into the set of binary relations. Compared with the constraint inference method that previously proposed by the one of the author, the new method builds a search tree by matching its levels not with variables (attributes), but with the equation of the system constraints.

Main Part

The state of the constraint satisfaction problem is completely characterized by a partial solution, written as a set of pairs of "attribute – domain of attribute ", as well as the remainder of *D*-systems, received during the constraint propagation procedure.

In this article we propose to use heuristics for choosing of successor the current node of the search tree. It allows to defer the analysis of less promising branches of the search tree, giving preference more promising ones to get a rapid decision. Some heuristics are used to select the line of *D*-system, and others are designed to select components of this line that will be included into the decision.

The process of finding solutions is displayed in a special table. Each step of the search is presented as a separate line with two vectors: vector and the vector of the partial solution of the problem.

Conclusion

Our heuristics are original and differ from heuristics of constraint satisfaction theory. It is clear that we can significantly reduce computational complexity of search procedure by dynamic building of search tree basing on the analysis of the current state of a constraint satisfaction problem. As a result, the probability of backtracking during the search is minimized.