УДК 519.68

Удовлетворение ограничений и программирование в ограничениях[1]

О.А. Щербина
University of Vienna,
Vienna 1090, Austria. *E-mail: oleg.shcherbina@univie.ac.at*

Аннотация. В статье представлен обзор основных направлений удовлетворения ограничений (УО) и программирования в ограничениях. Целью задачи УО является нахождение значений переменных, удовлетворяющих определенным ограничениям. В искусственном интеллекте парадигма УО признана в качестве удобного и эффективного способа моделирования и решения многих прикладных комбинаторных задач. Важная задача оценки конъюнктивных запросов в теории баз данных может рассматриваться как задача УО. Кроме того, задачи УО привлекают большое внимание в теории сложности, так как различные версии задач УО находятся в середине многих стандартных классов сложности, и поскольку у них имеется тенденция не иметь промежуточной сложности, то есть они бывают либо легко разрешимыми, либо полными для стандартных классов сложности. Программирование в ограничениях является парадигмой программирования для декларативного описания и эффективного решения комбинаторных задач, тесно связанной с теорией УО. Рассмотрены примеры использования моделей УО и конкретные приложения.

1. Введение

1.1. Удовлетворение ограничений

Использование подходов и алгоритмов искусственного интеллекта (ИИ) позволяет решать многие прикладные задачи, такие, как задачи теории расписаний [45], задачи проектирования экспертных систем и систем поддержки принятия решений [13], доказательство теорем, задачи тестирования электронных схем, обработка изображений. Одной из важных задач ИИ является задача удовлетворения ограничений (УО) (constraint satisfaction problem) [142], [171], [364].

Теория УО предлагает удобный аппарат и простую формальную схему для представления и решения комбинаторных задач искусственного интеллекта. Целью решения задачи УО является нахождение значений переменных, удовлетворяющих определенным ограничениям. Парадигма УО является обобщением пропозициональной логики, в которой переменным могут быть присвоены значения

[1]Работа выполнена при финансовой поддержке австрийского фонда FWF, грант No. P20900-N13

из множества любого числа переменных, а не только «истина» и «ложь» [138]. Проблема существования решений задачи УО является NP-полной.

В искусственном интеллекте парадигма УО признана в качестве удобного и эффективного способа моделирования и решения многих прикладных задач, таких как планирование [240] и календарное планирование [343], [166], [74], [49], [308], [321], [357], задачи назначения частоты [157], обработка изображений [289], при тестировании сверхбольших интегральных схем СБИС (VLSI) [211], анализ языков программирования [294] и понимания естественного языка [31]. В теории баз данных показано, что ключевая задача оценки конъюнктивных запросов может рассматриваться как задача УО [196], [247]. Кроме того, некоторые центральные задачи комбинаторной оптимизации могут быть представлены в виде задач УО [125], [161], [225], [241]. Опыт применения недоопределенных моделей к исследованиям экономики России является вполне обнадеживающим [9]. Помимо этого, задачи УО привлекают большое внимание в теории сложности, так как различные версии задач УО находятся в середине многих стандартных классов сложности, и так как у них имеется тенденция не иметь промежуточной сложности, то есть они бывают либо легко разрешимыми, либо полными для стандартных классов сложности [76], [77], [89], [94], [95], [125], [161], [234], [244], [314]. Программирование в ограничениях является парадигмой программирования для декларативного описания и эффективного решения комбинаторных задач, тесно связанной с теорией УО (см. [19], [266], [322], [377]).

УО является мощной парадигмой, позволяющей адекватно моделировать многие комбинаторные задачи [125], [263], [266], [364].

Многие классические комбинаторные задачи, такие как известная теорема Ферма, задача выполнимости (SAT) из пропозициональной логики, задача раскраски графа и задача изоморфизма графов из теории графов, задача BANDWIDTH из исследования операций могут формулироваться в виде задач УО (ЗУО). Остановимся подробнее на одной из давно поставленных задач в математике — задаче о раскраске графа (раскраска карты — частный случай этой задачи). Согласно данным, приведенным в (*Biggs et al.*, 1986) [67], гипотезу о четырех цветах (в соответствии с которой любой планарный граф можно раскрасить с помощью четырех или меньшего количества цветов) впервые выдвинул Френсис Гутри в 1852 году. Эта задача не поддавалась решению до тех пор, пока доказательство не было получено с помощью компьютера в (*Appel & Haken*, 1977) [33]. Формулировка задачи о раскраске в виде задачи УО ставит в соответствие вершинам раскрашиваемого графа переменные, возможные цвета представляют собой домены (области определения) переменных, а ограничения-неравенства между смежными вершинами являются ограничениями задачи.

Более сложным, но и более реалистическим примером распределения ресурсов является задача размещения выходов в аэропорту. Обычно необходимо рассматривать как физические ограничения (например, телетрапы определенного вида могут подходить только для определенных типов самолетов), так и пользовательские предпочтения (например, различные авиалинии предпочитают парковку в

определенных частях аэропорта). Описанная задача может быть представлена в виде задачи УО.

Специальные классы задач УО рассматривались на протяжении всей истории развития информатики. Одним из самых первых примеров решения таких задач была система Sketchpad (*Sutherland*, 1963) [356], которая решала геометрические ограничения на чертежах и была предшественницей современных программ рисования и инструментальных средств САПР.

Список приложений УО и программирования в ограничениях достаточно обширен. Здесь и вероятностные сети (belief maintenance) ((*de Kleer, 1989*) [130]; (*Dechter*, 1987) [131]; (*Dechter & Dechter*, 1988) [132]; (*Doyle*, 1979) [152]; (*McDermott*, 1991) [272]), кроссворды ((*Nadel*, 1989) [293]), темпоральный вывод ((*Allen*, 1984) [30]; (*Rit*, 1986) [319]; (*Tsang*, 1987) [363]), графовые задачи ((*Bruynooghe*, 1985) [87]; (*Fowler & Haralick*, 1983) [165]; (*McGregor*, 1979) [273]; (*Ullman*, 1976) [366]), задачи пропозициональной выполнимости ((*Ginsberg & McAllester*, 1994) [190]; (*Zabih & McAllester*, 1988) [387]), логическое программирование ((*Borning et al.*, 1989) [79]; (*van Hentenryck*, 1989) [368]), имитационное моделирование систем ((*Kuipers*, 1994) [254]; (*Miguel & Shen*, 1998) [282]; (*Shen & Leitch*, 1993) [341]), анализ электронных схем ((*Stallman & Sussman*, 1977) [355]), обработка естественного языка [277], проектирование экспертных систем (*Bartak*, 1997) [51].

В последнее время методы графовой декомпозиции и алгоритмы распространения ограничений были также использованы и для задач УО с непрерывными переменными [73], [117], [331] и для задач с ограничениями в виде дифференциальных уравнений [148] .

В литературе можно найти несколько хороших обзоров с описанием методов решения задач УО, включая (*Kumar*, 1992) [255], (*Dechter & Frost*, 1999) [138], (*Bartak*, 2001) [46], [278], [284], а также статьи в энциклопедических сборниках (*Dechter*, 1992) [135], [214] и (*Mackworth*, 1992) [263].

К сожалению обзоров на русском языке, посвященных проблемам теории УО, в настоящее время нет, хотя имеются публикации, посвященные отдельным аспектам УО [1], [5], [6], [8], [7], [10], [11], [14], [15], [16], [17], [18], [20], [22], [24].

Разумеется, в рамках данной обзорной статьи невозможно подробно описать все аспекты и направления теории удовлетворения ограничений и программирования в ограничениях, поэтому более полную информацию можно найти в монографиях [34], [142], [176], [266], [324], [368], [371], [374], [376].

К числу переводных монографий, в которых освещаются вопросы УО можно отнести книгу Рассел С., Норвиг П. Искусственный интеллект: современный подход. 2е изд.: Пер. с англ. М.: Вильямс, 2006. 1408 с. [12]. Обзор основных направлений программирования в ограничениях до 2000 г. сделан в работе [19].

Настоящая статья призвана заполнить указанный пробел и дать нашему читателю представление об основных направлениях теории УО и программирования в ограничениях.

Коснемся вначале терминологии и истории возникновения методов УО. *Montanari* [289] предложил использовать модели УО для описания ряда комбинатор-

ных задач, возникающих при компьютерной обработке изображений, и назвал эти задачи УО «*сетями ограничений*» (networks of constraints). Это связано с тем, что систему ограничений можно представить в виде неориентированного графа с вершинами-переменными и ребрами, соответствующими ограничениям между двумя переменными. По мнению *Dechter* [142], сети ограничений являются графовым представлением, используемым для поиска стратегий решения задач УО.

Достаточно быстро этот подход был использован для решения гораздо более широкого класса задач. В литературе используются оба этих термина — *сети ограничений* и *задачи удовлетворения ограничений* (*Mackworth* [263]).

1.2. О связи задач УО с другими математическими методами

Некоторые из прикладных задач, решаемых с помощью методов УО, могут быть также решены с помощью методов исследования операций: УО предлагает общность, гибкость, среду моделирования высокого уровня, управление поиском, компактность представления задачи, распространение ограничений, быстрые методы нахождения решения, в то время как исследование операций иногда является единственным способом поиска оптимального решения [194]. Следовательно, разумно попытаться взять лучшее у каждой из указанных технологий, путем создания гибридных систем, использующих обе технологии для решения задачи [212], [213]. Этот подход уже использован в некоторых системах, таких как **2LP** [270], [271], а также в работе [127], в которой этот гибридный подход использован для решения конкретных телекоммуникационных задач.

Заметим, что решение оптимизационной задачи может быть сведено к решению последовательности задач УО следующим образом. Находится допустимое решение, после чего добавляется ограничение, соответствующее целевой функции, которое выражает условие, что значение целевой функции должно быть лучше, чем для этого решения. Последовательные корректировки этого порогового значения, производимые до тех пор, пока задача не станет неразрешимой, позволяют найти оптимальное решение.

1.2.1. Задачи выполнимости и удовлетворение ограничений

Задача УО может быть преобразована в задачу выполнимости SAT несколькими способами (*Walsh*, 2000) [382] . Одним из них является прямое отображение. Для данной ЗУО $\mathcal{P} =< V, D, C >$[2]построим задачу выполнимости SAT следующим образом. Введем $|V| \cdot |D|$ переменных $x_{i,j}$. Переменные $x_{i,j}$ принимают значение «истина» тогда и только тогда, когда переменной V_i присвоено значение j. Для каждой переменной добавляем клаузы (дизъюнкты) $\bar{x}_{i,j} \vee \bar{x}_{i,k}$ для всех пар значений одной и той же переменной, чтобы гарантировать невозможность одновременного принятия переменной двух различных значений. Добавляем дизъюнкт

[2] $V = \{x_1, \ldots, x_n\}$ — множество переменных, $D = \{D_1, \ldots, D_n\}$ — множество доменов переменных, $C = \{C_1, \ldots, C_m\}$ — множество ограничений.

$(x_{i,0} \vee x_{i,1} \vee \dots x_{i,d})$, чтобы гарантировать, что переменной присвоено хотя бы одно значение.

1.2.2. Примеры

Приведем ряд примеров, иллюстрирующих постановки задач УО в других областях математики [253].

Пример 1. Наиболее тривиальным алгебраическим примером задачи УО является задача решения системы уравнений. Дана система линейных уравнений над конечным полем F. Имеет ли она решение? Ясно, что в этом примере каждое отдельное уравнение является ограничением, причем переменные уравнения образуют диапазон, а множество всех кортежей, соответствующих решениям уравнения, образует отношение ограничения.

Пример 2. Задача стандартной пропозициональной 3-выполнимости (3-SAT) [3], [300] определяется заданием формулы пропозициональной логики, состоящей из конъюнкции дизъюнктов, причем каждый дизъюнкт содержит 3 литерала (литерал — это переменная или ее отрицание), и ответом на вопрос, имеются ли значения переменных, которые делают формулу истинной.

Пусть $\Phi = \varphi_1 \wedge \dots \wedge \varphi_n$ — такая формула, где φ_i — дизъюнкты. Задача выполнимости для Φ может быть выражена в виде задачи УО $(V, \{0,1\}, \mathcal{C})$, где V — множество всех переменных в формуле, а $\mathcal{C}$ — множество ограничений $\{(s_1, \rho_1), \dots, (s_n, \rho_n)\}$, где каждое ограничение (s_k, ρ_k) построено следующим образом: s_k — список переменных, входящих в φ_k, а ρ_k состоит из всех кортежей, которые делают дизъюнкт φ_k истинным $(k = 1, \dots, n)$. Решения этой задачи УО — присвоения значений переменным, которые делают формулу Φ истинной. Значит, любая задача 3-выполнимости может быть выражена в виде задачи УО.

Пример 3. Любая конкретная задача УО может быть выражена в логической форме. Действительно, используя стандартное соответствие между отношениями и предикатами, можно переписать задачу УО в виде формулы первого порядка $\rho_1(s_1) \wedge \dots \wedge \rho_q(s_q)$, где ρ_i — предикаты на D и $\rho_i(s_i)$ означает предикат ρ_i, примененный к кортежу s_i переменных $(1 \leq i \leq q)$. Вопрос состоит в том, является ли эта формула выполнимой [332].

Эта задача обычно используется в теории баз данных (БД), поскольку она соответствует оценке конъюнктивного запроса [247], как показано в следующем примере.

Пример 4. Реляционная БД может быть рассмотрена как конечное множество таблиц. Таблица состоит из схемы и конкретных данных, где схема — конечное множество *атрибутов*, причем каждый атрибут имеет соответствующее ему множество возможных значений, называемое *доменом*. Конкретные данные — это конечное множество строк, где каждая строка — отображение, ставящее в соответствие каждому атрибуту схемы значение из ее домена. Стандартной задачей для реляционных БД является задача оценки конъюнктивного запроса [196], [247], в которой спрашивается, имеет ли решение конъюнктивный запрос, т.е. запрос вида $\rho_1 \wedge \dots \wedge \rho_q$, где $\rho_1, \dots, \rho_q$ — атомарные формулы. Конъюнктивный запрос над

реляционной БД соответствует конкретному примеру задачи УО, что достигается простой заменой терминов: «атрибуты» заменяются на «переменные», «таблицы» — на «ограничения», «схема» — на «диапазон», «конкретные данные» — на «отношение ограничения», а «строки» — на «кортежи». Значит, конъюнктивный запрос эквивалентен конкретному примеру задачи УО, переменные которой — это атрибуты запроса. Для каждой атомарной формулы ρ_i в запросе найдется ограничение C такое, что диапазон ограничения C — это список переменных формулы ρ_i и отношение ограничения C — это множество моделей ρ_i.

Пример 5. Другой важной переформулировкой в виде задачи УО является *задача о гомоморфизме*, состоящая в ответе на вопрос, существует ли гомоморфизм между двумя структурами отношений (см. [161], [197], [247]). Пусть $\tau = (R_1, \ldots, R_k)$ — сигнатура, т.е. список имен отношений с фиксированной арностью, приписанной каждому имени. Пусть $\mathcal{A} = (A;\ R_1^A, \ldots, R_k^A)$ и $\mathcal{B} = (B;\ R_1^B, \ldots, R_k^B)$ — структуры отношений из сигнатуры τ. Отображение $h : A \to B$ называется гомоморфизмом из A в B, если для всех $1 \leq i \leq k$, $(h(a_1), \ldots, h(a_m)) \in R_i^B$ всякий раз, когда $(a_1, \ldots, a_m) \in R_i^A$. В этом случае мы записываем $h : \mathcal{A} \to \mathcal{B}$. Чтобы убедиться, что задача о гомоморфизме сводится к задаче УО, представим элементы A в виде переменных, элементы B в виде значений, кортежи в отношениях A в виде диапазонов ограничений, а отношения из B — в виде отношений ограничений. Очевидно, что решения этой задачи УО в точности соответствуют гомоморфизмам из $\mathcal{A}$ в $\mathcal{B}$.

Пример 6. Интервальные задачи УО. Одним из видом задач УО с бесконечным числом значений, интенсивно изучаемых в ИИ, являются задачи УО, в которых значения, принимаемые переменными, являются интервалами действительной прямой. Эти задачи используются для моделирования поведения во времени систем, где интервалы представляют интервалы времени, в течение которых происходят события. Наиболее популярной формальной структурой такого рода является интервальная алгебра Аллена (ИАА), введенная в [29], описывающая бинарные качественные отношения между интервалами. ИАА содержит 13 базовых отношений, соответствующих 13 различным способам, согласно которым могут соотноситься два интервала. Полная система отношений в ИАА состоит из $2^{13} = 8192$ возможных объединений базовых отношений. Интервальная алгебра Аллена имеет три операции над отношениями: композиция, пересечение и инверсия.

Пример 7. Модель удовлетворения ограничений для задачи «SEND MORE MONEY»

Студент посылает домой закодированную телеграмму:

$$\begin{array}{r} \text{SEND} \\ +\ \text{MORE} \\ \hline \text{MONEY} \end{array}$$

Задача состоит в нахождении 8 различных цифр, закодированных 8 буквами, ис-

пользованными в телеграмме, так, чтобы получался арифметически правильный результат.

Заметим, что эта задача может быть записана с помощью модели целочисленного программирования (ЦП), но она содержит около сотни целочисленных переменных, причем эта модель ЦП трудна для решения с помощью классических подходов ЦП, таких, как алгоритмы ветвей и границ.

В то же время логический анализ задачи позволяет найти ее решение. Так, замечая, что ни S, ни M не могут быть нулями, получим: $S \neq 0,\ M \neq 0$. Далее, единственной возможностью для M является значение 1, т.е. $M = 1$. Затем, из равенства $M = 1$ получаем, что S может быть равно только 9. Можно показать, что символу O соответствует цифра 0, т.е. $O = 0$. Далее, переходя к сотням, из $O = 0$ и условия, что E и N должны быть разичными, получим $N = E + 1$. Продолжая аналогично логический анализ ограничений, найдем решение задачи: $E = 5,\ N = 6,\ D = 7,\ R = 8,\ Y = 2$. Основная сложность этой задачи состоит в необходимости учета ограничения «все цифры различны» (так называемое глобальное ограничение **all-different**).

Запишем ограничения задачи.

Первая формулировка.

Переменные: $\{S, E, N, D, M, O, R, Y\}$
Домены: $\{0, 1, \ldots, 9\}$
Ограничения:
$C1:\ \forall x, y \in \{S, E, N, D, M, O, R, Y\},\ x \neq y$
$C2:\ M = 0$ or $M = 1$
$C3:\ (1000 \times S + 100 \times E + 10 \times N + D) + (1000 \times M + 100 \times O + 10 \times R + E) =$
$= (10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y)$

Вторая формулировка.

Переменные: $\{S, E, N, D, M, O, R, Y\} + \{C1, C2, C3\}$
Домены: $\forall x \in S, E, N, D, M, O, R, Y, D_x = 0, \ldots, 9$
$\forall y \in \{C1, C2, C3\}, D_y = 0, 1$
Ограничения:
$C1 : \forall x, y \in \{S, E, N, D, M, O, R, Y\}, x \neq y$
$C2 : M = 0$ or $M = 1$
$C3 : D + E = 10 \times C1 + Y$
$C4 : N + R + C1 = 10 \times C2 + E$
$C5 : E + O + C2 = 10 \times C3 + N$
$C6 : S + M + C3 = 10 \times M + O$

Ниже, в разделе 5, приведена программа решения этой задачи на языке PROLOG.

Пример 8. Задача о N ферзях

Задача о N ферзях состоит в размещении N ферзей на шахматной доске $N \times N$ так, чтобы они не угрожали друг другу (см. рис. 1).

Модель удовлетворения ограничений:
ферзи в вертикальных столбцах: $\forall i\ r(i) \in \{1, \ldots, N\}$

отсутствие конфликтов:
$\forall i \neq j,\ r(i) \neq r(j)\ \&\ |i-j| \neq |r(i)-r(j)|$.

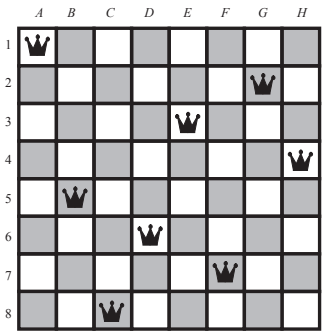


Рис. 1. Одно из решений задачи о 8 ферзях

Одним из примеров использования УО является решение кроссвордов.

Пример 9. Пример составления кроссворда

Рассмотрим пример составления кроссворда из [142]. Задача состоит в вертикальной или горизонтальной записи слов из заданного множества слов (словаря) в таблицу с учетом некоторых ограничений. Если разрешено вставлять каждое слово на пустое место соответствующей длины, возможная формулировка задачи о кроссворде в виде задачи УО выглядит следующим образом. Для каждого квадрата кроссворда вводится переменная, принимающая буквенные значения из алфавита, причем могут быть заданы возможные значения для групп смежных переменных. Приведем формальную формулировку задачи о кроссворде в терминах

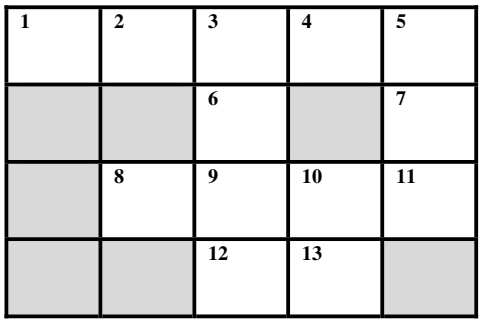


Рис. 2. Кроссворд.

ограничений. Каждому квадрату i кроссворда, который должен быть заполнен

(см. рис. 2) поставим в соответствие переменную x_i $(i = 1, \ldots, 13)$. Переменные $x_1, \ldots, x_{13}$ в качестве доменов имеют буквы алфавита, в качестве ограничений служат допустимые слова. Например, ограничения могут быть заданы следующим образом [142]:

$$R_{1,2,3,4,5} = \{(H, O, S, E, S),\ (L, A, S, E, S),\ (S, H, E, E, T),\ (S, N, A, I, L),\ (S, T, E, E, R)\}$$

$$R_{3,6,9,12} = \{(A, L, S, O),\ (E, A, R, N),\ (H, I, K, E),\ (I, R, O, N),\ (S, A, M, E)\},$$

$$R_{5,7,11} = \{(E, A, T),\ (L, E, T),\ (R, U, N),\ (S, U, N),\ (T, E, N),\ (Y, E, S)\},$$

$$R_{8,9,10,11} = R_{3,6,9,12},$$

$$R_{10,13} = \{(B, E),\ (I, T),\ (N, O),\ (U, S)\},$$

$$R_{12,13} = R_{10,13}.$$

Пример 10. Задача составления расписания

Важной для практических приложений является задача составления расписания, состоящая в упорядочении набора заданий (работ). В этой задаче заданы список заданий и ограничения, какие задачи могут быть выполнены одновременно, выполнение каких заданий должно предшествовать выполнению других и т.д.

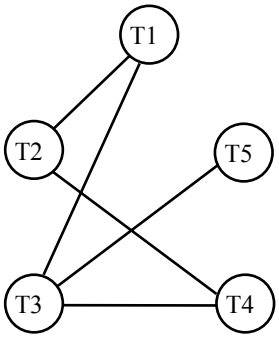
Для решения этой задачи нужно найти назначение времен начал работ заданиям так, чтобы все ограничения удовлетворялись [367].

Рассмотрим задачу составления расписания для пяти заданий $T1$, $T2$, $T3$, $T4$, $T5$, каждое из которых может быть выполнено за один час. Задания могут начинаться в 1:00, 2:00 или 3:00. Любые работы могут выполняться одновременно, учитывая ограничения на то, что $T1$ может начинаться после $T3$, $T3$ может начинаться до $T4$ и после $T5$, $T2$ не может начинаться в то же время, что $T1$ или $T4$, $T4$ не может начинаться в 2:00.

Можно построить модель составления графика, введя пять переменных, соответствующих заданиям с доменами $\{1:00,\ 2:00,\ 3:00\}$. Соответствующий граф ограничений показан на рис. 3.

Пример 11. Разметка изображений

Задача разметки изображений [384] является, по-видимому, одной из первых формализованных задач УО. Цель этой задачи состоит в распознавании объектов 3-мерного изображения с помощью интерпретации линий на 2-мерных рисунках. Линии или ребра должны быть помечены, т.е. они разбиты на несколько типов, а именно на выпуклые (+), вогнутые (-) и замыкающие ребра (<). В более сложных системах распознается также теневой край. Имеется много способов разметки изображения (в точности 3^n, где n — число ребер), но только немногие из них имеют смысл для трехмерного изображения. Идея решения этой комбинаторной задачи состоит в нахождении допустимых меток для точек пересечения, удовлетворяющих ограничениям, что ребро имеет ту же метку на обоих концах. Это намного снижает перебор, так как имеется лишь ограниченное число допустимых меток для точек пересечения.



Унарное ограничение $\mathbf{D_{T4}=\{(1:00, 3:00)\}}$.

Бинарные ограничения

$\mathbf{R_{T2,T1}=\{(1:00, 2:00), (1:00, 3:00), (2:00, 1:00), (2:00, 3:00), (3:00, 1:00), (3:00, 2:00)\}}$,

$\mathbf{R_{T2,T4}=\{(1:00, 2:00), (1:00, 3:00), (2:00, 1:00), (2:00, 3:00), (3:00, 1:00), (3:00, 2:00)\}}$,

$\mathbf{R_{T1,T3}=\{(2:00, 1:00), (3:00, 1:00), (3:00, 2:00)\}}$,

$\mathbf{R_{T3,T4}=\{(1:00, 2:00), (1:00, 3:00), (3:00, 2:00)\}}$,

$\mathbf{R_{T3,T5}=\{(2:00, 1:00), (3:00, 1:00), (3:00, 2:00)\}}$.

Рис. 3. Граф ограничений и отношения задачи составления графика.

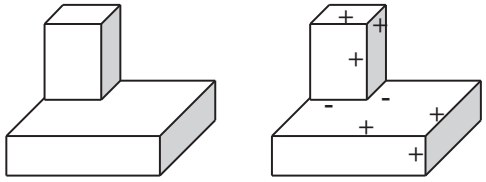


Рис. 4. Разметка изображений.

Рис. 5. Допустимые варианты разметки при соединении в вершине.

Основные алгоритмы решения описанной задачи (подобно алгоритму разметки *Waltz* [384]) относятся к достижению некоторых форм совместимости.

Другие практические приложения УО и программирования в ограничениях приведены ниже, в разделе 5.

2. Основные понятия теории удовлетворения ограничений

2.1. Базовые понятия

2.1.1. Задача удовлетворения ограничений

В данной статье рассматриваются задачи УО с дискретными переменными. Задачи УО простейшего вида характеризуются тем, что в них используются *дискретные* переменные, которые имеют *конечные домены* (области определения). К такому виду относится задача раскраски карты. Задача с восемью ферзями, описанная в главе 1, также может рассматриваться как задача УО с конечной областью определения, где переменные $Q_1, \ldots, Q_8$ представляют собой позиции каждого ферзя на столбцах $1, \ldots, 8$, а каждая переменная имеет область определения $\{1, 2, 3, 4, 5, 6, 7, 8\}$. По оценке *Bartak* [47], вероятно, более 95% приложений УО используют конечные домены.

Если максимальный размер доменов переменных в задаче УО равен d, то количество возможных полных присваиваний значений переменным измеряется величиной $O(D^n)$, т.е. зависит экспоненциально от количества переменных. К категории задач УО с конечной областью определения относятся *булевы задачи* УО, в которых переменные могут иметь значения либо *истина*, либо *ложь*. Булевы задачи УО включают в качестве частных случаев некоторые NP-полные задачи, такие как задачи 3-выполнимости $3SAT$.

Дискретные переменные могут иметь бесконечные области определения, например, такие, как множество всех целых чисел или множество всех строк. В исследовании операций часто встречаются задачи УО с непрерывными областями определения. В частности, при календарном планировании строительных работ дата начала каждой работы представляет собой переменную, а ее возможными значениями являются целочисленные интервалы времени в сутках, отсчитываемые от текущей даты. При решении задач с бесконечными областями определения больше нет возможности описывать ограничения, перечисляя все допустимые комбинации значений. Вместо этого должен использоваться специальный *язык ограничений*. Например, если работа Job_1 которая занимает пять дней, должна предшествовать работе Job_3, то для описания этого условия потребуется язык ограничений, представляющий эти условия в виде $StartJob_1 + 5 \leq StartJob_3$. Кроме того, уже нет возможности решать такие ограничения, просто перечисляя все

возможные присваивания, поскольку имеется бесконечно много возможных присваиваний. Для *линейных ограничений* с целочисленными переменными существуют специальные алгоритмы поиска решений. В некоторых случаях задачи с целочисленными ограничениями можно свести к задачам с конечной областью определения, устанавливая пределы значений этих переменных. Например, в задаче планирования можно установить верхний предел, равный общей продолжительности всех запланированных работ.

Сеть ограничений или задача удовлетворения ограничений (УО) состоит из множества переменных $V = \{x_1, \ldots, x_n\}$, множества доменов значений D_i для каждой переменной x_i, и множества ограничений и отношений. Каждый домен значений является конечным множеством значений, которые может принимать сответствующая переменная. Ограничение R_S над $S \subseteq X$ — это подмножество декартова произведения доменов переменных в S. Если $S = \{x_{i_1}, \ldots, x_{i_r}\}$, то $R_S \subseteq D_{i_1} \times \cdots \times D_{i_r}$. S называется *диапазоном* (scope) ограничения R_S.

В бинарной сети ограничений все ограничения заданы над парами переменных. Граф ограничений ставит в соответствие каждой переменной вершину, причем две вершины соединяются ребром, если соответствующие переменные имеются в диапазоне какого-либо ограничения.

При задании ограничений используются *отношения*.

Определение. Для данных множества переменных $V = \{x_1, \ldots, x_n\}$ и соответствующих их областей значений $D_1, \ldots, D_n$ *отношением* R на множестве переменных называется любое подмножество декартова произведения их областей значений. Множество переменных, на котором определено отношение R, называется *диапазоном* отношения и обозначается $scope(R)$.

Если $R = D_1 \times \ldots \times D_n$, то R называется *универсальным отношением*.

Отношения могут задаваться с помощью таблиц, описывающих допустимые сочетания значений переменных.

Определение. Задача УО (ЗУО) определяется множеством дискретных переменных $V = \{x_1, \ldots, x_n\}$, для каждой из которых задана область определения или домен $D_j = \{d_j^{(1)}, \ldots, d_j^{(p_j)}\}$ $(j = 1, \ldots, n)$, и множеством ограничений. *Ограничением* называется пара (R, S), где R — отношение, определенное на диапазоне S. ЗУО может рассматриваться как тройка (V, D, C), где $V = \{x_1, \ldots, x_n\}$ — множество переменных, $D = \{D_1, \ldots, D_n\}$ — множество доменов переменных, $C = \{C_1, \ldots, C_m\}$ — множество ограничений. *Решением* ЗУО называется присвоение значений всем переменным, которое удовлетворяет всем ограничениям. *Целью* решения ЗУО может быть нахождение одного или всех решений.

2.1.2. Действия над отношениями

Пусть R_1, R_2 – два отношения с одинаковым диапазоном. Тогда *пересечением* $R_1 \cap R_2$ называется отношение, содержащее все наборы значений переменных, которые имеются одновременно в обоих отношениях R_1 и R_2.

Объединение $R_1 \cup R_2$ – отношение, содержащее все наборы значений переменных, которые имеются либо в R_1, либо в R_2, или в обоих отношениях.

Разностью $R_1 - R_2$ называется отношение, содержащее наборы значений переменных, содержащиеся в R_1, но не содержащиеся в R_2.

Проекция $\Pi_Y(R)$ отношения R на множество переменных $Y \subseteq scope(R)$ является отношением, содержащим наборы значений только переменных, содержащихся в Y. В табличном представлении отношений проекция выбирает подмножество столбцов, соответствующих множеству Y.

Соединение отношений. Оператор соединения $R_S \bowtie R_T$ из двух отношений R_S с диапазоном S и R_T с диапазоном T строит новое отношение, состоящее из их общих переменных в S и T. Набор r из соединения $R_S \bowtie R_T$ отношений R_S и R_T можно построить, используя следующие шаги: (1) взять набор s из R_S; (2) выбрать набор t из R_T такой, что компоненты s и t согласуются по переменным из $S \cap T$, общим для R_S и R_T; (3) образовать новый набор r, комбинируя компоненты s и t, сохраняя лишь один из полученных одинаковых наборов. Диапазон получающегося отношения – $S \cup T$. Соединение двух отношений с одинаковыми диапазонами эквивалентно пересечению этих отношений.

2.2. Виды ограничений

Рассмотрим различные виды ограничений. Простейшим типом ограничения является *унарное* ограничение, которое ограничивает значение одной переменной. Любое унарное ограничение можно устранить, выполняя предварительную обработку области определения соответствующей переменной, чтобы удалить значения, нарушающие это ограничение. *Бинарное* ограничение связывает между собой две переменные. Например, бинарным ограничением является $x \neq y$. Бинарной задачей УО называется задача, в которой имеются только бинарные ограничения; она может быть представлена в виде графа ограничений. В ограничениях высокого порядка участвуют три или больше переменных. Одним из известных примеров таких задач являются *криптоарифметические* головоломки, называемые также числовыми ребусами (см. раздел 1). Обычно накладывается требование, чтобы каждая буква в криптоарифметической головоломке представляла отдельную цифру. В случае задачи SEND+MORE=MONEY из примера 7 раздела 1, такое требование может быть выражено с помощью ограничения с восемью переменными $all - different(S, E, N, D, M, O, R, Y)$. Иным образом это требование может быть представлено в виде набора бинарных ограничений, таких как $S \neq E$.

2.2.1. Глобальные ограничения

Ряд исследований по удовлетворению ограничений с конечными доменами использует *глобальные ограничения* [325]. Понятие глобального ограничения состоит в следующем: имеются случаи, когда наличие одного большего ограничения (по числу переменных в нем) может быть лучше, чем наличие множества меньших ограничений. Типичным примером служит множество ограничений-неравенств, которое редко позволяет применить успешное распространение ограничений. В подобных случаях необходимо использовать полный поиск по доменам всех переменных для нахождения одного решения. Можно избежать этого бесполезного поиска, используя ограничение "**all-different**", в которое входят все переменные. Действительно, используя одно такое ограничение, задача поиска решения может быть переформулирована в виде задачи паросочетания в двудольном графе (bipartite matching) между переменными и значениями, для решения которой могут быть использованы классические графовые алгоритмы, что может снизить перебор. В общем, введение глобальных переменных работает следующим образом: вместо записи задачи с помощью многих исходных бинарных ограничений, мы описываем ее с помощью небольшого числа глобальных ограничений для множеств переменных. Таким образом, мы можем проще моделировать задачу с меньшим числом ограничений, при этом могут быть использованы более совершенные алгоритмы. Типичными примерами служат ограничения "**all-different**" и "**atmost**", устанавливающее лимит на число переменных с определенным значением, а также *кумулятивное* ограничение.

Несмотря на то, что *глобальные ограничения* являются важным аспектом теории УО, достаточно четкого их определения нет. Глобальное ограничение — это ограничение над подмножеством переменных. У глобальных ограничений есть два преимущества. Во-первых, глобальные ограничения облегчают задачу моделирования прикладной проблемы в виде ЗУО. Во-вторых, можно разработать алгоритм распространения ограничений специального вида, учитывающий особенности ограничения и поэтому намного более эффективный. Напомним, что вынуждение дуговой совместимости на произвольной задаче имеет временную сложность $O(rd^r)$ в наихудшем случае, где r — арность ограничения, а d — размер доменов переменных. В то же время **all-different**-ограничение может быть сделано вершинно-совместным за время $O(r^2d)$ в худшем случае [311].

Классическим примером глобального ограничения служит **all-different**-ограничение, которое определяет, что переменные должны быть попарно различными. **all-different**-ограничение часто используется на практике и в связи с этим предлагается как встроенное ограничение во многих основных коммерческих и исследовательских системах программирования в ограничениях. Начиная с первых глобальных ограничений в системе программирования в ограничениях **CHIP** [27], позднее были предложены и внедрены сотни глобальных ограничений (см. [58]).

Другими примерами глобальных ограничений являются глобальное *кардинальное* ограничение (global cardinality constraint (gcc)) [312] и *кумулятивное* ограни-

чение [27]. Глобальное кардинальное ограничение над множеством переменных и значений определяет условие, что число переменных, которым присвоены значения, должно быть между заданными верхней и нижней границами, причем эти границы могут быть различны для каждого значения. Кумулятивное ограничение над множеством переменных, представляющих времена выполнения различных работ, гарантирует, что работы упорядочены так, что не превышаются имеющиеся в наличии величины ресурсов, используемые в любые периоды времени. Ограничения обоих указанных типов обычно используются при составлении расписаний, упорядочении работ и в календарном планировании.

2.2.2. Мягкие ограничения

Следует заметить, что ограничений с конечными доменами недостаточно для точного моделирования реальных задач. Проблема состоит в том, что реальные задачи обычно не могут быть описаны только с помощью классических ограничений, которые могут быть истинны либо ложны, поскольку они имеют такие характеристики, как предпочтения, вероятности, стоимости, а также степень неопределенности. Многие прикладные задачи УО обычно являются переограниченными (overconstrained) [223]: пользователь задает слишком много ограничений, так что все их одновременно удовлетворить невозможно. В таких случаях, вначале нужно выяснить, имеются ли решения задачи УО вообще (для этого может потребоваться много времени!), а затем нужно решить, какие из ограничений ослабить, чтобы задача стала разрешимой. В связи с этим было введено понятие мягкого ограничения (soft constraint) [50]: *мягкое ограничение* — это обычное ограничение плюс его дополнительная характеристика (критерий), которая может интерпретироваться как стоимость, уровень важности, степень неопределенности и т.п. Нахождение решения ЗУО с мягкими ограничениями не означает нахождение такового, удовлетворяющего всем ограничениям, но получение значений переменных, для которых достигается «наилучшее» значение для выбранного критерия. Исследования в этой области началось с иерархической системы программирования в ограничениях [79], языка программирования в ограничениях, в котором каждое ограничение имеет уровень важности и решение задачи УО находится с учетом иерахии ограничений. Для формализации этой новой модели и исследования эффективности алгоритмов ее решения в [154], [328], [160], [333] были разработаны нечеткие ограничения, у которых каждому ограничению присвоено значение между 0 и 1. В [235] рассматривается комбинация иерархии ограничений и нечетких ограничений. Переограниченные задачи решались также в [172] с использованием понятия частичного УО, при этом в ЗУО вводилась метрика, что позволило находить решение, удовлетворяющее возможно большему числу ограничений.

Были проведены теоретические исследования по разработке более общих схем для мягких ограничений, используя аппарат полуколец [68], [69], [70] (где кортежу каждого ограничения поставлен в соответствие элемент полукольца), а также аппарат ограничений с оценками (valued constraint), в котором каждому ограничению поставлен в соответствие элемент из вполне упорядоченного множества

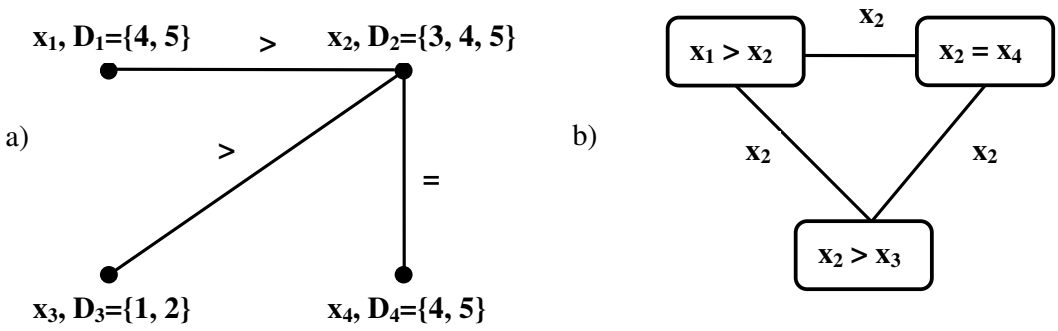


Рис. 6. a) Граф ограничений и b) двойственный граф.

[98], [143], [334]. Для этих общих схем получены обобщения и произведена адаптация распространения ограничений и методов решения, обычно используемых для ограничений с конечными доменами, и изучены их свойства.

2.2.3. Приведение задач УО высокого порядка к бинарным задачам УО

Каждое ограничение высокого порядка с конечными доменами можно свести к множеству бинарных ограничений, введя достаточное количество вспомогательных переменных. Нетрудно заметить, что ограничение **all-different** может быть разбито на бинарные ограничения — $S \neq E$, $S \neq N$ и т.д.

Двойственное преобразование было перенесено из теории БД в теорию УО *Dechter & Pearl* [141]. Первые попытки приведения задач УО высокого порядка к чисто бинарным задачам УО с помощью вспомогательных переменных были впервые предприняты в XIX веке логиком Чарльзом Сандерсом Пирсом (Peirce). Основу метода скрытого преобразования составляет метод Peirce [302] представления небинарных отношений с помощью системы бинарных отношений. *Rossi et al.* [323] на основе метода *Peirce* [302] впервые показали, что бинарные отношения имеют те же выразительные возможности, что и небинарные отношения. Соответствующие методы были введены в тематику УО в работе *Dechter* (1990) [134] и доработаны *Bacchus & van Beek* [39].

2.3. Графы ограничений

2.3.1. Граф ограничений, двойственный граф ограничений, гиперграф ограничений

Структура или топология задач УО может описываться с помощью различных графовых структур: *(первичного) графа ограничений, гиперграфа ограничений, двойственного графа ограничений.*

Определение. Первичный *граф ограничений* ЗУО (V, D, C) — это неориентированный граф $G = (V, E)$, вершины V которого соответствуют переменным ЗУО, причем две вершины соединяются ребром в графе G, если соответствующие переменные имеются в одном и том же ограничении (т.е. в одном и том же диапазоне какого-то отношения).

Граф ограничений определен как для бинарных, так и для небинарных ограничений. Однако для небинарного случая структура ограничений может быть более точно передана с помощью *гиперграфа ограничений*.

Гиперграф ограничений ЗУО — это гиперграф G_P, вершины которого соответствуют переменным задачи УО, а гипер-ребра — ограничениям C, вернее, подмножествам переменных из диапазонов ограничений $scope(C)$.

Если арность каждого ограничения в C не более 2, то задача УО называется бинарной задачей УО. Гиперграф, соответствующий бинарной ЗУО, является обычным графом. Гиперграф, соответствующий небинарной ЗУО, может быть преобразован к обычному графу ограничений путем замены каждого гиперребра кликой, состоящей из вершин гиперграфа и соединяющих их ребер [239].

В ряде случаев удобно использовать *двойственный граф ограничений*[3], вершины которого соответствуют диапазонам ограничений, а ребра соединяют вершины при наличии в них общих переменных. При этом ребра помечаются множествами общих переменных. Заметим, что двойственный граф ограничений тесно связан с двойственной ЗУО, в которой переменными (так называемыми *c-переменными*) являются ограничения исходной задачи. Ограничения двойственной задачи вынуждают общие переменные из ограничений принимать одни и те же значения, т.е. двойственные ограничения бинарны. Отсюда видно, что с помощью двойственных задач УО можно любую небинарную ЗУО свести к бинарной и решить ее с помощью методов решения бинарных УО.

2.3.2. Индуцированный граф и индуцированная ширина задачи УО

Введем понятия *индуцированного графа* и *индуцированной ширины* (induced width) задачи УО, важные для применения структурных методов решения задач УО. Индуцированная ширина определяется в терминах графа ограничений задачи УО.

Попутно введем понятие упорядочения графа или задачи УО.

Определение. Рассмотрим задачу УО (V, D, C), где $|V| = n$, и соответствующий граф ограничений $G = (V, E)$. *Упорядочением* для задачи УО и графа $G = (V, E)$ называется биекция $h : V \leftrightarrow \{1, 2, \ldots, n\}$, которая соответствует вершинам или переменным, просматриваемым в порядке $(h^{-1}(1), h^{-1}(2), \ldots, h^{-1}(n))$. Если две переменные соединены в графе ребром, иногда полезно представить первую

[3]Двойственный граф ограничений по сути совпадает с *реберным графом гиперграфа*: реберным графом гиперграфа $H = (V, \mathcal{E})$ согласно [4] называется граф $L(H) = (\mathcal{E}, E)$ множество вершин которого совпадает с множеством ребер $\mathcal{E}$ гиперграфа H, при этом две вершины графа $L(H)$ смежны тогда и только тогда, когда смежны соответствующие им ребра гиперграфа H. Таким образом, $L(H)$ — граф пересечений ребер гиперграфа H.

переменную, которая стоит раньше в упорядочении, в виде «родителя», а вторую — как «сына» первой вершины.

Перед определением индуцированной ширины введем более простое понятие *ширины* графа.

Определение. Пусть $G = (V, E)$ — граф, а h — некоторое упорядочение вершин этого графа. *Шириной* вершины v для этого упорядочения h называется число вершин, соединенных с v и предшествующих ей в этом упорядочении (т.е. число родителей вершины v). *Шириной графа* для упорядочения h называется максимальная ширина всех вершин для этого упорядочения. И наконец, *шириной графа* называется минимальная ширина для всех упорядочений. Например, дерево имеет ширину 1, а полный граф с n вершинами имеет ширину $n - 1$. Введем далее следующие определения:

Определение. *Шириной задачи УО* для данного упорядочения называется ширина его графа ограничений для этого упорядочения, а ширина ЗУО определяется как ширина ее графа ограничений.

Введем согласно *Dechter & Pearl* [140] понятие индуцированного графа по отношению к данному упорядочению.

Определение. Для данного графа $G = (V, E)$ и упорядочения h, индуцированный граф $G^* = (V, E^*)$ определяется как граф с минимальным множеством ребер E^*, таких что $E^* \supseteq E$, и если $\{x, v\} \in E^*$, $\{y, v\} \in E^*$, $h(x) < h(v)$, $h(y) < h(v)$, и $x \neq y$, то $\{x, y\} \in E^*$.

Индуцированный граф может быть построен за один проход, обрабатывая вершины исходного графа в обратном порядке; то есть вначале соединяем родителей последней вершины, затем родителей предпоследней вершины и т.д.

Определение. *Индуцированная ширина* $w(h)$ графа $G = (V, E)$ по отношению к упорядочению h — это ширина индуцированного графа (G^*, h) по отношению к этому упорядочению. Индуцированная ширина w^* графа — это его минимальная индуцированная ширина по всем упорядочениям. Индуцированная ширина задачи УО по отношению к упорядочению h — это индуцированная ширина ее графа ограничений по отношению к этому упорядочению, а индуцированная ширина задачи УО — это индуцированная ширина ее графа ограничений.

Синоним индуцированной ширины — *древовидная ширина* [23]. Родственным является понятие также частичного k-дерева[4]. *Dechter & Pearl* [140] показали, что данная задача УО при заданном упорядочении может быть решена за время, экспоненциальное от индуцированной ширины по отношению к этому упорядочению.

2.3.3. Хордальные графы

Вычисление индуцированной ширины достаточно несложно для хордальных графов. Напомним, что граф называется *хордальным*, если в нем каждый цикл длины 4 имеет хорду.

[4] Частичное k-дерево — это граф, индуцированная ширина которого не превышает k [38].

Многие трудные графовые задачи легко решаются на хордальных графах. Например, нахождение всех максимальных клик в графе, что является NP-полной задачей на графах общего вида, легко произвести для хордальных графов. Эта задача (поиск максимальных клик в хордальных графах) облегчается за счет использования процедуры упорядочения *max-cardinality*[5].

Упорядочение *max-cardinality* графа упорядочивает вершины, начиная от первой, согласно следующему правилу: первая вершина выбирается произвольно. После этого, следующей выбирается вершина, соединенная с максимальным числом уже упорядоченных вершин и т.д. Упорядочение max-cardinality может быть использовано для распознавания хордальных графов. Именно, граф — хордальный, тогда и только тогда, когда при упорядочении max-cardinality каждая вершина вместе с ее родителями образует клику. Можно таким образом перечислить все максимальные клики, соответствующие каждой вершине (записывая множества, состоящие из каждой вершины и ее родителей, и определяя затем максимальный размер клики). Заметим, что имеется не более n клик, состоящих из вершин и их родителей. Кроме того, при использовании упорядочения max-cardinality хордального графа, упорядоченный граф идентичен его индуцированному графу и, значит, его ширина равна его индуцированной ширине. Справедливо

Предложение. Если G^* — индуцированный граф графа G для некоторого упорядочения, тогда граф G^* — хордальный.

3. Методы поиска решения ЗУО

3.1. Основные методы решения

Методы построения решения задачи УО могут быть разбиты на три класса [329]: 1) *Первый класс содержит варианты поиска с возвратами.* Эти алгоритмы строят решение с помощью расширения частичного решения шаг за шагом, используя различные эвристики и применяя разумные стратегии возврата из тупиковых вершин. Снижение размера задачи позволяет уменьшить размеры пространства поиска. 2) *Алгоритмы распространения ограничений* исключают некоторые элементы, не входящие в решения, из пространства поиска. В общем, эти алгоритмы не исключают *все* элементы, не входящие в решения и, следовательно, они не строят сами по себе решения, а используются либо для препроцессинга задачи до использования алгоритма другого типа, или перемежаются с шагами алгоритма другого типа — например, поиска с возвратами — для повышения производительности последнего. 3) Наконец, *структурные алгоритмы* используют информацию о структуре первичного или двойственного графа ограничений задачи. Имеются различные алгоритмы этого класса, при этом некоторые производят декомпозицию исходной задачи УО на слабо связанные подзадачи, которые могут быть решены с помощью методов из предыдущих двух классов. Таким образом,

[5]max-cardinality — максимальное количество элементов.

структурные алгоритмы могут также использоваться в сочетании с алгоритмами других типов. Все алгоритмы из указанных выше трех классов *систематически* исследуют пространство решений. Эти алгоритмы:

- *корректны*, то есть они заканчивают работу с присвоением значений всем переменным, которое является решением;
- *полны*, т.е. они способны исследовать все пространство поиска и найти все решения.

Обсудим методы поиска с возвратами.

3.2. Поиск с возвратами (Backtrack Search) и его производные

3.2.1. Поиск с возвратами

Формулировка задачи УО в виде задач поиска позволяет решать задачи УО с помощью алгоритмов поиска.

Поиск в глубину, в котором каждый раз выбираются значения для одной переменной и выполняется возврат, если больше не остается допустимых значений, которые можно было бы присвоить переменной, называется *поиском с возвратами* (backtracking)[6].

Дерево присвоений значений переменным — это дерево, в котором каждая вершина соответствует множеству присвоений. Корень дерева отвечает пустому множеству присвоений. В каждой вершине v выбирается переменная, которой в v не было присвоено значение. Алгоритмы поиска с возвратами выполняют поиск в глубину в этих деревьях присвоений значений.

Простейшим методом решения задач УО является метод "Generate and Test", который порождает каждое возможное решение путем присвоения всех возможных значений для каждой переменной и проверяет, удовлетворяет ли это решение всем ограничениям. В наихудшем случае число всех возможных проверенных решений равно размеру декартова произведения доменов всех переменных[7]. Понятно, что метод полного перебора может требовать больших затрат времени при решении задач.

Алгоритмы поиска с возвратами — это алгоритмы систематического поиска для задач УО, которые используют частичные решения, которые строятся путем поочередного присваивания значений переменным. Если алгоритм обнаруживает тупиковую вершину, в котором частичное решение не имеет совместного расширения, то выбор последнего присвоения отменяется и делается попытка присвоения другого значения. Этот процесс повторяется до тех пор, пока не будут исчерпаны все возможности и/или не будет найдено решение. Алгоритмы поиска с возвратами

[6] Использование поиска с возвратами для удовлетворения ограничений было предложено *Bitner & Reingold* (1975) в [71].

[7] Если домены всех n переменных имеют размер d, то эта величина составляет d^n.

```
Procedure BT(i)
  Foreach Val In D[i]
    Assignments[i]:= Val
    Consistent:=True
    For h:=1 To i-1 While Consistent
      Consistent:=Test(i,h)
    If Consistent
      If i=n
        Show - Solution()
      Else
        BT(i+1)
    Return False
End Procedure
```

Рис. 7. Поиск с возвратами.

являются полными в том смысле, что они находят решение, если оно существует. Различные алгоритмы поиска с возвратами различаются по скорости обнаружения тупиковых вершин, а также по тому, насколько далеко они способны делать возврат. Алгоритм поиска с возвратами характеризуется экспоненциальной временной сложностью, но является линейным по используемой памяти (*Mackworth*, 1977) [262].

В *хронологическом поиске с возвратами* (chronological backtracking) процесс поиска рассматривается как постепенное расширение частичных решений (*Bitner & Reingold*, 1975) [71]. Порядок задания значений — это порядок, в котором присваиваются значения переменным во время поиска. На уровне i порядка присваивания прошлые переменные имеют индексы, меньшие чем i, причем им уже присвоены значения. Будущие переменные имеют индексы, большие чем i и им еще не присвоены значения. На рис. 7 описан алгоритм поиска с возвратами. Для простоты, записанный здесь псевдокод не различает случаев, когда не найдено решений или когда не найдено больше решений, значение *ложь* возвращается в обоих указанных случаях. Алгоритм описан для случая бинарных ограничений $c(x_i, x_j)$. Используются следующие обозначения:

- n — число переменных задачи.
- Assignments [ ]. Этот массив размерности n запоминает значение, присвоенное в настоящий момент каждой переменной.
- $D[\,]$. i-й элемент этого массива размерности n содержит последовательность элементов домена, соответствующего переменной x_i .

Функция $Test(i, j)$ принимает значение *истина*, если ограничение $c(x_i, x_j)$ удовлетворяется текущими присвоениями значений переменным x_i и x_j. Если ограниче-

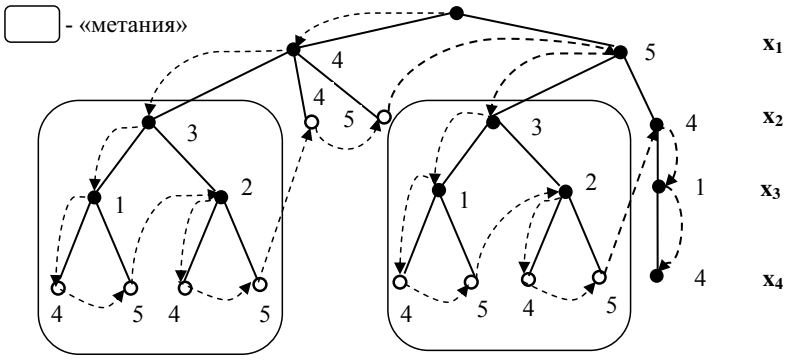


Рис. 8. Поиск с возвратами как обход дерева.

ния $c(x_i, x_j)$ не существует, Test( ) принимает *истина* (это не считается проверкой ограничения). Каждый элемент домена D_i поочередно присваивается переменной x_i, расширяя присвоение значений переменным $x_1, \ldots, x_{i-1}$. Если проверка ограничения для предыдущей переменной не проходит, это частичное присвоение больше не расширяется. Если домен D_i исчерпан (тупик или полное уничтожение домена), алгоритм делает возврат до уровня $i-1$ с целью нахождения другого совместного присвоения для x_{i-1}. Если же такового нет, делается возврат к x_{i-2}, и т.д. пока не будет найдена переменная с другим совместным присвоением или будет показано, что ЗУО не имеет решения. Для данного совместного присвоения (все проверки ограничений были успешны), делается рекурсивный вызов BT$(i+1)$ или, если не остается свободных переменных, получено решение. Поиск с возвратами может рассматриваться как обход вершин дерева. На рис. 8 показано, как поиск с возвратами решает задачу, описанную на на рис. 6. Каждая ветвь соответствует частичному присвоению (которое становится полным, когда ветвь доходит до n-й переменной) и уровень i дерева представляет выборы значений, сделанные для i-й (согласно упорядочению) переменной. Черные и белые вершины представляют соответственно успешные и неудачные присвоения в данной точке поиска. Вершины снабжены описанием значений доменов, приписанных данной переменной в рассматриваемой точке поиска. Путь поиска показан пунктирной направленной линией.

Поиск с возвратами неэффективен в том смысле, что требует выполнения бóльшего числа операций, чем это нужно для нахождения решения.

3.2.2. Проблема мечущегося поведения алгоритма с возвратами

Метания (*Gaschnig*, 1979) [181] являются главным фактором плохого поведения поиска с возвратами: поиск бывает безуспешен повторно по той же причине. Например, пусть $c(x_g, x_i)$ — условие того, что конкретное присвоение переменной x_g запрещает все потенциальные значения для x_i. Поиск с возвратами безуспешен

на уровне i для каждого элемента D_i , причем эта неудача повторяется для каждой комбинации присвоений на промежуточных уровнях переменной x_h, где $g < h < i$. Метание наблюдается в попытке поиска с возвратами решить задачу на рис. 6. Рис. 8 показывает повторяющиеся неудачные попытки поиска в связи с условием $c(x_2, x_4)$. Метаний становится больше при увеличении числа промежуточных переменных (и величин их доменов).

3.2.3. Интеллектуальный поиск с возвратами

Интеллектуальный поиск с возвратами (intelligent backtracking) [42], [86] — это собирательный термин, объединяющий алгоритмы поиска с возвратами, позволяющие решателю обнаружить, что конкретная тупиковая вершина дерева присвоения значений не связана с некоторыми присвоениями. А именно, если тупиковая вершина была обнаружена на уровне l и алгоритм может выяснить, что присвоения, сделанные на уровнях $j \in \{k+1, \ldots, l\}$ не относятся к этой тупиковой вершине, он может перейти обратно к уровню k, поскольку попытка различных присвоений этим переменным постоянно приводила бы к тупиковой вершине. Таким образом, вместо хронологического поиска с возвратами на уровне $l-1$, этот алгоритм позволяет избежать лишней работы, связанной с попыткой присваивания различных присвоений переменным на этих уровнях, и вместо этого при возврате прямо пытается присвоить другое значение на более раннем уровне k.

К различным методам интеллектуального поиска с возвратами относятся *обратный переход* (backjumping), *обратный переход, управляемый конфликтами* (conflict-directed backjumping), *обратная проверка* (backchecking), *проставление обратных отметок* (backmarking).

Основной метод обратного перехода (backjumping) был разработан в работах (*Gaschnig*, 1977, 1979) [180], [181]. Методы обратного перехода являются одним из главных средств для преодоления тенденций поиска с возвратами к повторному нахождению одних и тех же тупиковых вершин. Тупиковая вершина встречается, когда у x_i не остается совместных значений, в этом случае алгоритм поиска с возвратами возвращается в вершину x_{i-1}. Пусть существует новое значение для x_{i-1}, но нет ограничений между x_i и x_{i-1}. Тупиковая вершина будет в x_i для любого значения переменной x_{i-1}, пока не будут исчерпаны все значения переменной x_{i-1}. Мы можем обойти эту ситуацию путем определения переменной — виновницы за тупик и последующим немедленным обратным переходом к присвоению значения переменной-виновницы тупика (конфликта), вместо повторного фиксирования значения хронологически предшествующей переменной. Определение переменной-виновницы тупика в алгоритме поиска с возвратами основано на понятии *конфликтного множества*. Для простоты изложения будем предполагать, что переменные упорядочены следующим образом: $h = \{x_1, \ldots, x_n\}$.

Вместо того, чтобы дожидаться тупиковой вершины $\overrightarrow{a}_i$[8], метод обратного перехода Gaschnig'a [181] запоминает некоторую информацию в процессе построения

[8] $\overrightarrow{a}_i$ — тупиковый кортеж, представляющий собой набор присвоений значений переменным.

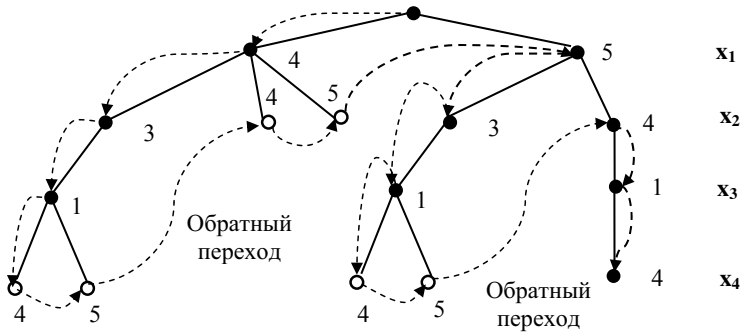


Рис. 9. Метод обратного перехода.

$\overrightarrow{a}_i$, и использует эту информацию для определения переменной x_b — виновницы тупиковой вершины (см. рис. 9). Алгоритм использует метод проставления отметок, посредством которого для каждой переменной используется указатель на последнего предшественника, который оказался несовместным с любым из значений переменной. При построении кортежей $\overrightarrow{a}$ на прямом шаге алгоритм сохраняет указатель $high_i$ для каждой переменной x_i. Этот указатель определяет последнюю переменную, проверенную на совместность с x_i, если было обнаружено, что она является самой первой переменной, конфликтующей со значением в D_i'. Например, в случае, когда нет сравнимых значений для x_i и если $high_i = 3$, это показывает, что $\overrightarrow{a}_3$ — конфликтное множество переменной x_i. Если $\overrightarrow{a}_i$ имеет совместное значение, то $high_i$ приписывается значение $i-1$. Алгоритм переходит из тупикового листа $\overrightarrow{a}_i$, который несовместен с x_{i+1}, обратно к переменной $x_{high_{i+1}}$ — виновнице конфликта, так как тупиковая переменная — x_{i+1}.

Две идеи: *обратный переход* к переменной, которая, будучи зафиксирована, находится в конфликте с текущей переменной, и *обратный переход* к внутренним тупикам, интегрированы в одном алгоритме обратного перехода, управляемого конфликтами (*Prosser*, 1993) [309]. Этот алгоритм использует схему обратного перехода, используя информацию, собранную в процессе поиска. Для каждой переменной алгоритм поддерживает индуцированное множество обратных переходов. Для данного тупикового кортежа $\overrightarrow{a}_i$, определим множество обратных переходов для $\overrightarrow{a}_i$ (или для x_{i+1}) как множество переменных, участвующих в самом раннем минимальном конфликтном множестве $\overrightarrow{a}_i$. Обратный переход, управляемый конфликтами, включает во множество переходов переменную, если ее текущее значение конфликтует со значением текущей переменной, которая не была в конфликте ни с одним более ранним присвоением переменной.

Предложенный Сталманом и Зюссманом (*Stallman & Sussman*, 1977) [355] метод *поиска с возвратами управляемого зависимостями*, являющийся общей и мощной формой интеллектуального поиска с возвратами, привел к разработке

систем обеспечения истинности (truth maintenance systems) (*Doyle*, 1979) [152]. Связь между этими двумя направлениями проанализирована в (*Kleer*, 1989) [130]. В указанной работе [355] была также предложена идея *регистрации ограничения* (constraint recording), в соответствии с которой частичные результаты, полученные в ходе поиска, можно сохранять и повторно использовать на последующих этапах этого поиска. Данная идея была введена формально в поиск с возвратами в работе (*Dechter*, 1990 a) [133].

Особенно простым методом является *проставление обратных отметок* (backmarking) (*Gaschnig*, 1979) [181], в котором для предотвращения повторной проверки ограничений сохраняются и используются совместные и несовместные попарные присваивания. Проставление обратных отметок [181] — это алгоритм типа кэширования, снижающий число проверок на совместность, необходимых для построения каждой вершины, не пытаясь отсечь часть пространства поиска самого по себе. Сохраняя информацию о том, где проверка на совместность не прошла ранее, проставление обратных отметок может исключить необходимость излишне повторять выполненные ранее проверки.

Опишем этот алгоритм как усовершенствование наивного алгоритма поиска с возвратами. Напомним, что алгоритм поиска с возвратами движется либо вперед, либо назад в пространстве поиска. Допустим, что текущая переменная — x_i, а x_p — самая первая переменная (согласно упорядочению), которая изменила свое значение после последнего захода в x_i. Понятно, что любая проверка значений x_i по отношению к переменным, предшествующим x_p даст те же результаты: если она была неудачной по отношению к более ранним присвоениям, то она будет неудачна снова; если она была успешной, то она опять будет успешной. Следовательно, при сохранении правильной информации из более ранних частей поиска, значения в домене переменной x_i могут быть либо немедленно распознаны, как несовместные, либо еще проверены только по отношению к предшествующим присвоениям, начиная с x_p. Проставление обратных отметок реализует эту идею с помощью использования двух новых таблиц. Вначале для каждой переменной x_i и для каждого из ее значений a_v, проставление обратных отметок запоминает самую первую предшествующую переменную x_p, такую, что текущее частичное присваивание $\overrightarrow{a}_p$ конфликтует с $x_i = a_v$. Эта информация содержится в таблице с элементами $M_{i,v}$. (Заметим, что это предполагает, что ограничения, включающие более ранние переменные, протестированы до ограничений, включающих более поздние переменные). Если $x_i = a_v$ совместно со всеми ранее выполненными частичными присвоениями $\overrightarrow{a}_j$, $j < i$, то $M_{i,v} = i$. Например, $M_{10,2} = 4$ означает, что $\overrightarrow{a}_4$ оказалось несовместным с $x_{10} = a_2$ после задания значений и что $\overrightarrow{a}_1$, $\overrightarrow{a}_2$, $\overrightarrow{a}_3$ не конфликтовали с $x_{10} = a_2$. Вторая таблица с элементами low_i запоминает самый первый элемент, изменивший значение после того, как переменной x_i в последний раз было присвоено значение. Эта информация включена в использование на каждом шаге порождения вершины. Если $M_{i,v}$ меньше low_i, то алгоритм знает, что переменная, на которую указывает $M_{i,v}$, не изменялась и что $x_i = a_v$ будет приводить к тупику снова при проверке с $\overrightarrow{a}_{M_{i,v}}$, так что никакой дальней-

шей проверки в этой вершине на совместность не требуется. Если $M_{i,v}$ не меньше low_i, то $x_i = a_v$ совместно с $\overrightarrow{a}_j$ для всех $j < low_i$, и эти проверки могут быть пропущены.

Проставление обратных отметок можно комбинировать с обратным переходом, управляемым конфликтами; в [248] предложен гибридный алгоритм, который превосходит любой из этих отдельно взятых методов (что подтверждается формальным доказательством). Применение локального поиска при решении задач удовлетворения ограничений было показано (*Kirkpatrick et al.*, 1983) в [243], где описан метод эмуляции отжига.

Эвристика с минимальными конфликтами (minimal conflicts) была впервые предложена (*Gu*, 1989) в [200] и независимо разработана в [285] (*Minton et al.*, 1992). В (*Sosic & Gu*, 1994) [354] задача с 3 000 000 ферзей была решена быстрее, чем за минуту. Это привело к переоценке характера и распространенности «легких» и «трудных» задач. В (*Cheeseman et al.*, 1991) [104] исследовалась сложность задач УО, сформированных случайным образом, и было обнаружено, что почти все такие задачи являются либо тривиально легкими, либо не имеют решений. «Трудные» экземпляры задач встречаются, только если параметры генератора задач устанавливаются в некотором узком диапазоне, в пределах которого лишь примерно половина задач является разрешимой. Для решения задач с ограничениями весьма успешно может быть использован локальный поиск на основе эвристики с *минимальными конфликтами*.

В работе [88] показано, как может быть усовершенствована программа хронологического поиска с возвратами на языке **Prolog**, для работы с интеллектуальным поиском с возвратами.

3.2.4. Запоминание конфликтов

В тупиковой вершине присвоения переменных $x_1, \ldots, x_{i-1}$ образуют конфликтное множество (КМ) (conflict set) с x_i. Эта информация используется для запоминания одного или нескольких ограничений (или множеств неперспективных наборов значений (МННЗ) (nogoods) — см. ниже параграф 3.2.5), которые при дальнейшем поиске не допускают несовместностей, приводящих к этой тупиковой вершине. Рассмотрим рис. 6 и дерево поиска, построенное поиском с возвратами (рис. 8). В первой тупиковой вершине КМ $\{x_1 = 4,\ x_2 = 3,\ x_3 = 1\}$ конфликтует с x_4. Запоминание КМ полезно лишь в случае, если задача УО будет повторно решаться: точно такое же множество присвоений не встретится во время этого поиска. Однако, КМ возможно содержит подмножества в конфликте с x_i. Минимальные КМ (*Bruynooghe*, 1985) [87] не содержат никаких других КМ и отвечают за текущий конфликт. Процесс поиска выигрывает, если такие минимальные КМ больше не встречаются.

Запоминание конфликтов может рассматриваться как обучение (глубокое или поверхностное) (*Dechter*, 1990) [133]: полезная информация, вскрытая решателем задачи, запоминается для дальнейшего использования. Глубокое обучение находит все минимальные конфликты в тупиковой вершине, реализованные в методе поис-

ка с возвратами, управляемого зависимостями (Dependency Directed Backtracking) (*Stallman & Sussman*, 1977 [355]) и обычно используется в системах обеспечения истинности (truth-maintenance systems) (*Doyle*, 1979 [152]). Это требует затрат памяти: каждая тупиковая вершина представляет новую возможность добавления ограничений. В связи с этим, в памяти сохраняется большая часть пространства поиска.

Поверхностное обучение [133] ограничивает объем работы, выполняемой в тупиковой вершине x_i с помощью удаления всех присвоений переменных из конфликтного множества, которые не релевантны (т.е. совместны со всеми возможными присвоениями) к x_i. Графовое поверхностное обучение использует граф ограничений для (неполной) проверки на нерелевантность: x_h не релевантен к x_i, если переменные не смежны в графе. В вышеописанном примере, это выводит $\{x_1 = 4, x_3 = 1\}$, оставляя $\{x_2 = 3\}$. Этот конфликт запоминается с удалением знавчения 3 из D_2. Добавление новых ограничений, однако, затрудняет поиск из-за увеличения числа проверок ограничений. Запомненные конфликты могут весьма вероятно оказаться полезными в процессе поиска. *Dechter* (1990) [133] вводит ограничение на число запоминаемых конфликтов. Обучение первого порядка запоминает лишь одну переменную. Обучение второго порядка добавляет ограничения, содержащие до двух переменных.

Dechter (1990) [133] произвела сравнение поверхностного и глубокого обучения первого и второго порядка. Для легких задач обучение дает небольшой эффект, но для более сложных задач наблюдается больший эффект на производительность алгоритма. Не всегда верно то, что наиболее сильная форма обучения дает максимальный эффект, так как в некоторых случаях затраты могут превышать выгоду.

Bitner & Reingold впервые предложили использовать эвристику MRV, названную ими эвристикой с наиболее ограниченной переменной (most-constrained-variable). *Brelaz* (1979) [82] использовал степенную эвристику (degree heuristic) для устранения неопределенности, возникающей после применения эвристики MRV. Полученный в итоге алгоритм, несмотря на его простоту, до настоящего времени остается наилучшим методом раскраски произвольных графов в k цветов. *Haralick & Elliot* (1980) [203] предложили эвристику с наименее ограничительным значением. В статье [248] (*Kondrak & van Beek*, 1997) приведен аналитический обзор алгоритмов поиска с возвратами.

3.2.5. Множества неперспективных наборов значений

Определение (*Dechter*, 1990) [133]. *Множеством неперспективных наборов значений* (МННЗ) (nogood) [238] N_g или конфликтным множеством (conflict set) для задачи УО P называется множество наборов значений, которые не могут быть частью никакого решения задачи P.

По-другому, МННЗ — частичное присвоение значений переменным, которое не приводит к решению [66]. Таким образом, МННЗ не может быть дополнено до решения задачи УО. Например, любое присвоение значений, нарушающее ограничение, является МННЗ. ЗУО не имеет решения тогда и только тогда, когда

пустое множество присвоений является МННЗ. Множество присвоений $\mathcal{A}$ является МННЗ, если имеется не просмотренное решение, содержащее $\mathcal{A}$. Любое супермножество МННЗ является также МННЗ.

Изучение МННЗ — стандартный метод усовершенствования поиска с возвратами (Dechter 1990) [133]. Заметим, что метод обучающих МННЗ был перенесен в решатели задачи выполнимости SAT в (Bayardo & Schrag 1997) [54].

В контексте задач выполнимости SAT МННЗ соответствует клаузе (дизъюнкту), причем использование обучения и использования большого числа клауз во время поиска (*Moskewicz et al.* 2001) [291] оказалось весьма эффективным для решения задач выполнимости. В то же время, МННЗ не имели столь большого влияния на решатели ЗУО. Так, большинство коммерческих решателей задач УО не используют обучения с МННЗ. Под обучением (learning), как и выше, здесь понимается запоминание потенциально полезной информации, которая становится известной решателю задачи во время процесса решения [138].

Алгоритмы поиска с возвратами имеют различные возможности для нахождения МННЗ в процессе поиска, что и используется при разработке различных алгоритмов поиска с возвратами. В [309] МННЗ, называемые *конфликтами*, используются для более мощных обратных переходов в алгоритме обратного перехода, управляемого конфликтами.

В работе [4] МННЗ, названные там *исключающими объяснениями* (eliminating explanations), используются для управления динамическим переупорядочением ветвей дерева поиска в алгоритме динамического поиска с возвратами. В [5] МННЗ изучаются в тупиках и используются для улучшения последующего поиска. Работы [42], [130], [286], [355], также содержат полезные идеи о связи между МННЗ и поиском с возвратами. Отмечено, что по сути дела любой алгоритм поиска с возвратами использует МННЗ для управления поиском, явно или неявно.

В работе [238] предложено понятие обобщенного МННЗ, которое существенно повышает результативность поиска с возвратами.

3.2.6. Динамический поиск с возвратами

Обратный переход на уровень выше удаляет результаты проделанной решателем работы по определению совместного присвоения на предыдущем уровне.

Во многих случаях уровни, с которых делается возврат, не имеют ничего общего с несовместностью, которую алгоритм пытается разрешить (примером может служить мечущееся поведение, описанное выше). В результате, потенциально полезная информация бессмысленно выбрасывается при поиске реальной причины несовместности.

В методе *динамического поиска с возвратами* (ДПВ) (dynamic backtracking) (*Ginsberg*, 1993) [188], [189] сохраняются успешные частичные присваивания из полученных позднее подмножеств переменных, причем ДПВ обходится без фиксированного порядка присвоения значений, вместо этого выбирая переменные для присвоения значений динамическим образом. ДПВ эффективным образом меняет порядок присваивания значений так, что уровень, ответственный за несовмест-

ность, становится самым глубоким уровнем, оставляя не затронутыми все нерелевантные присвоения переменных.

Для каждого $d_{i_a} \in D_i$ ДПВ сохраняет множество объяснений причин элиминации, в котором запоминается причина того, почему d_{i_a} в настоящее время исключено из рассмотрения. Эти объяснения определены в терминах текущих присвоений ранее рассмотренным переменным. Когда необходим переход от x_i к x_h, ДПВ сохраняет текущее присвоение x_i и все его множества объяснений причин элиминации, кроме тех, которые включают x_h (присвоение которого должно меняться).

Использование множеств объяснений причин элиминации аналогично МННЗ, запоминающих причины конфликтов. Объяснения причин элиминации хранятся лишь для текущих присвоений другим переменным; как только присваивание изменилось, все объяснения, включающие его, исключаются. Итак, ДПВ представляет собой компромисс между емкостной и временной сложностями.

ДПВ, однако,не работает всегда лучше существующих алгоритмов. Сравнивая эффективность ДПВ в сравнении с другими алгоритмами, используя стандартные эвристики (*Baker*, 1994) [42] показал, что ДПВ работает хуже, чем алгоритм обратного перехода на множитель, экспоненциально зависящий от размера задачи. Проблема здесь состоит в том, что присвоения переменных, которые ДПВ не отбросил, часто не имеют на основе эвристик подтверждения того, что найдена несовместность. Хранение таких присвоений переменных ухудшает результат применения эвристик.

3.3. Распространение ограничений

3.3.1. О распространении ограничений

Среди различных видов методов для преодоления присущей задачам УО трудной разрешимости выделяются алгоритмы *распространения ограничений* [35].

При удалении избыточных значений из доменов переменных размер пространства решений снижается. Алгоритмы уменьшения пространства решений исключают значения с помощью распространения ограничений, называемого также *сужением* [116]. Уменьшение пространства решений задачи может быть выполнено либо однократно — в виде этапа препроцессинга перед другим алгоритмом, либо — многократно, шаг за шагом, перемежаясь с исследованием пространства решений с помощью алгоритма поиска. В последнем случае отсекаются подмножества пространства решений, экономя время, которое алгоритм поиска затратил бы на систематическое исследование отброшенных элементов. Если в результате уменьшения пространства решений какой-либо домен стал пустым, отсюда следует, что рассматриваемая задача УО не имеет решений.

Объем распространения ограничений характеризуется уровнем совместности задачи, поэтому эти алгоритмы называются также *алгоритмами совместности.*

Распространение ограничений [48] имеет давнюю традицию в УО и является очень общим понятием, которое появлялось под разными названиями: релаксация

ограничения (constraint relaxation), фильтрующие алгоритмы (filtering algorithms), сужающие алгоритмы (narrowing algorithms), вывод ограничений (constraint inference), алгоритмы упрощения (simplification algorithms) и др. Ниже будут рассмотрены наиболее хорошо известные и часто используемые алгоритмы. Распространение ограничений, предложенное вначале для разметки изображений в работах [289], [384], было затем распространено и приспособлено к решению многих других задач, породив большое число алгоритмов распространения ограничений, при этом в числе наиболее значимых следует отметить работы [136], [140], [255], [262], [263], [265], [289].

С помощью распространения (информации) воздействие предшествующих присвоений может уменьшить размер пространства поиска. Основным методом здесь является усечение домена, которое удаляет некоторые значения из доменов переменных, которые будут рассмотрены позднее. Как только присвоение будет осуществлено, может быть использован алгоритм распространения ограничений для нахождения последствий предшествующих присвоений на домены переменных, просматриваемых позднее.

Исследования по распространению ограничений и алгоритмам совместности были инициированы в работе *Waltz* [383], в которой был предложен алгоритм распространения ограничений (позднее получивший название дуговой совместности) для задач полиэдральной линейной разметки для систем компьютерного видения. Он показал, что для некоторых многогранных фигур достаточно применить основные алгоритмы распространения для решения этих задач. *Препроцессинг* или *предварительная обработка* в задачах УО создает эквивалентную, с тем же множеством решений, но более простую задачу. Поиск на дереве после препроцессинга может решить полученную задачу с гораздо меньшими затратами. Исходная задача УО преобразуется следующим образом. Вначале домены переменных могут фильтроваться для удаления элементов, которые не могут быть частью ни одного решения. Далее множество ограничений может быть изменено с целью недопущения несовместных комбинаций присвоений. Изменения производятся с помощью распространения ограничений: используется локальная группа ограничений для вывода информации, которая запоминается в виде измененного домена ограничения или переменной. Это локальное изменение является основой для дальнейших выводов, поэтому результат любого изменения постепенно распространяется по всей задаче.

Хотя с целью простоты изложения представленные здесь методы описаны для бинарных задач УО, они применимы к n-арной задаче УО, используя, например, двойственную графовую интерпретацию.

Рассмотрим далее понятие локальной совместности.

3.3.2. Локальная совместность

Одним из наиболее важных понятий в УО является понятие локальной совместности. Локальная несовместность — это присвоение значений некоторым переменным, удовлетворяющее соответствующим ограничениям, но которое не может

быть расширено на одну или более переменных и поэтому не может быть частью никакого решения. Если для поиска решения ЗУО используется поиск с возвратами, то подобная несовместность может быть причиной многих тупиковых вершин и может стать причиной многих бесполезных попыток поиска. Это приводит к (a) определению условий, характеризующих уровень локальной совместности задачи УО ([167], [262], [289]), (b) разработке алгоритмов распространения ограничений — алгоритмов, которые вынуждают эти уровни к локальной совместности с помощью удаления несовместностей из ЗУО ([262], [289]), и (c) разработке эффективных алгоритмов поиска с возвратами для поиска решений ЗУО, поддерживающих уровень локальной совместности во время поиска ([128], [179], [203]).

В общем, имеется компромисс между затратами на распространение ограничений, выполняемое в каждой вершине дерева поиска и величиной усечения доменов. Один из способов уменьшения затрат на распространение ограничений — это рассмотрение более ограниченных локальных совместностей, одним из примеров которых служит совместность границ. Предположим, что домены переменных велики и упорядочены, причем они могут быть представлены интервалами (задаваемыми минимальным и максимальным значением домена). Используя совместность границ, вместо выяснения того, что каждое значение домена имеет поддержку в ограничении, выясняется лишь то, имеют ли минимальное и максимальное значение поддержку в ограничении. Хотя совместность границ слабее, чем дуговая совместность, она полезна для арифметических ограничений и глобальных ограничений, так как она может быть вынуждена более эффективно. Задача, включающая бинарные ограничения, путе-совместна, если каждая совместная пара значений двух переменных может быть расширена до любой третьей переменной. Для некоторых видов задач, подобных темпоральным ограничениям, возможно имеет смысл вынуждение более сильных уровней совместности, чем путевая совместность [262].

3.3.3. Вершинная и дуговая совместность

Простейшим методом препроцессинга является *вершинная совместность*, которая удаляет те значения из домена переменной, которые несовместны с унарными ограничениями на соответствующую переменную: если $c(x_i)$ — унарное ограничение для вершины x_i, тогда величины, не удовлетворяющие $c(x_i)$, исключаются из домена D_i. Прямой алгоритм вершинной совместности (node-consistency algorithm (NC)), который удаляет излишние элементы доменов переменных с помощью проверки доменов одного за другим, имеет временную сложность $O(dn)$, где d — максимальный размер доменов, а n — число переменных.

Дуговая совместность состоит в рассмотрении каждой дуги[9], соответствующей бинарному ограничению $c(x_i, x_j)$, и выполнении сужения доменов. Величины, не удовлетворяющие $c(x_i, x_j)$, исключаются из доменов D_i и D_j.

Проверку совместности дуг можно использовать либо в качестве этапа предва-

[9]В данном случае термин «дуга» обозначает ориентированное ребро в графе ограничений.

рительной обработки перед началом процесса поиска, либо в качестве этапа распространения ограничения (аналогично предварительной проверке) после каждого присваивания во время поиска.

Поиск компромисса между затратами на обеспечение совместности и достигаемыми за счет этого преимуществами с точки зрения сокращения объема поиска привел к новым алгоритмам. Так, (*Haralick & Elliot*, 1980) [203] предложили алгоритм предварительной проверки (forward checking), описанный в (*McGregor*, 1979) [273], а (*Gaschnig*, 1979) [181] предложил применять полную проверку совместности дуг после присваивания значения каждой переменной; в дальнейшем соответствующий алгоритм в работе (*Sabin & Freuder*, 1994) [330] получил название MAC (**M**aintaining **A**rc **C**onsistency). В последней статье показано, что при решении более трудных задач УО полная проверка совместности дуг вполне окупается.

В работе *Mackworth & Freuder* [265] было замечено, что дуговая совместность может решать древовидные задачи УО.

Для данного ограничения говорят, что значение для переменной в ограничении имеет *поддержку* (support), если существуют такие значения для других переменных в этом ограничении, что это ограничение удовлетворяется.

Ограничение дуго-совместно, если каждое значение из доменов переменных имеет поддержку.

Для достижения дуговой совместности на $arc(i, j)$, для каждого $d_{i_a} \in D_i$, ищется элемент $d_{j_b} \in D_j$, такой что выполняется условие дуговой совместности; d_{j_b} поддерживает d_{i_a}. Все элементы домена без поддержки удаляются. Удаление значение без поддержки называется усечением (сужением) доменов.

Для ограничений, содержащих более двух переменных, дуговая совместность часто называется гипер-дуговой совместностью или обобщенной дуговой совместностью. Например, пусть домены переменных x и y — $\{0, 1, 2\}$ и рассмотрим ограничение $x+y = 1$. Вынуждение дуговой совместности на этом ограничении сузило бы домены обеих переменных до $\{0, 1\}$. Величины, удаленные из доменов обеих переменных, локально несовместны — они не принадлежат ни одному набору присвоений переменных, удовлетворяющих ограничению, и поэтому не могут быть частью какого-либо решения всей задачи УО. Для вынуждения дуговой совместности в задаче УО требуется повторно удалять величины из доменов, пока не достигнем устойчивой точки.

Основываясь на работах [383], [289], *Mackworth* [262] выделил три вида локальной совместности: вершинную, дуговую и путевую совместность. Им были предложены алгоритмы (AC-1, AC-2, AC-3) для вынуждения дуговой совместности, а также алгоритмы PC-1, PC-2, PC-3 для вынуждения путевой совместности. Вычислительная сложность указанных алгоритмов была исследована далее в работе *Mackworth & Freuder* [265], причем там же было также замечено, что дуговая совместность может решать древовидные задачи УО.

Алгоритмы вынуждения дуговой совместности интенсивно исследовались и совершенствовались (см. [65], [262] и ссылки на литературу в этих работах). Оптимальный алгоритм для произвольного ограничения характеризуется временно́й

сложностью в худшем случае, равной $O(r \cdot d^r)$, где r — арность ограничения, а d — размер доменов переменных [288].

Простейшим методом достижения глобальной дуговой совместности является повторение проходов по проверке каждой дуги задачи до тех пор, пока не будет никаких изменений графа ограничений после очередного прохода. Этот подход реализован в виде алгоритма AC-1 (из (*Machworth*, 1977) [262].

Вслед за исторически первым алгоритмом AC-1 были разработаны алгоритмы AC-2, ..., AC-7, каждый из которых улучшает хранение информации об измененных доменах и управление итерациями.

Вычислительная сложность алгоритмов AC-1, AC-2, AC-3 была исследована далее в работе *Mackworth & Freuder* [265]. Временна́я сложность алгоритма AC-3 составляет $O(d^3e)$, емкостная сложность — $O(e + nd)$, где e — число бинарных ограничений. Алгоритм AC-4 [287] улучшает AC-3 и имеет временную и емкостную сложность $O(d^2e)$. Другим улучшением алгоритма AC-3 является алгоритм AC-5 [369], использующий семантику специальных видов ограничений. Алгоритм AC-7 использует симметрию бинарных ограничений.

Сложность проверки совместности дуг можно проанализировать следующим образом: любая бинарная задача УО имеет самое большее $O(n^2)$ дуг; каждая дуга (X_i, X_j) может быть «внесена в повестку дня» только d раз, поскольку область определения X_i имеет самое большее d значений, доступных для удаления; проверка совместности любой дуги может быть выполнена за время $O(d^2)$, поэтому в наихудшем случае затраты времени составляют $O(n^2 \cdot d^3)$.

3.3.4. Направленная дуговая совместность

Определение. Задача УО является *направленно-дуго-совместной* по отношению к упорядочению $h = \{x_1, \ldots, x_n\}$, тогда и только тогда, когда каждая переменная x_i дуго-совместна по отношению к каждой переменной x_j, такой что $i \leq j$.

Процедура распространения ограничений **DAC** работает с задачей УО (V, D, C) и упорядочением $h = \{x_1, \ldots, x_n\}$ для задачи УО. Процедура обрабатывает переменные в обратном порядке $(x_n, \ldots, x_1)$. Для каждой переменной x она проверяет родителей x[10]. При обработке x_i, рассматриваются все бинарные ограничения, инцидентные x_i, R_{ki}, при $k \leq i$, и сужаются соответствующие домены D_k переменных x_k.

После выполнения этой предварительной обработки выясняется направленная дуговая совместность [140]. Это означает, что для любого значения домена переменной и для любого сына этой переменной найдется по крайней мере одно значение из домена сына, совместное со значением родителя.

Глобально совместные ЗУО обладают свойством, что любое совместное присвоение значений подмножеству переменных может быть расширено до совместного присвоения значений всем переменным без получения тупиков.

[10] переменные, соединенные с x, которые предшествуют x в упорядочении.

3.3.5. Путевая совместность

Для бинарных задач УО имеется еще один вид совместности — путевая совместность, которую *Montanari* предложил в [289].

Определение Бинарная задача УО является *путе-совместной* (path-consistent), если для любого пути в ее графе ограничений справедливо следующее: если присвоения значений начальной и конечной переменной пути совместны, то это может быть расширено на совместное частичное присвоение значений оставшимся переменных на пути.

Аналогично семейству алгоритмов дуговой совместности, имеется семейство алгоритмов достижения путевой совместности бинарной задачи УО. Эти алгоритмы также гарантируют вершинную и дуговую совместность. Наилучший из этих алгоритмов — PC-4, аналогичный AC-4, характеризуется временно́й и емкостной сложностью $O(d^3n^3)$.

Локальная путевая совместность выполняется, если для данного пути длины l через вершины $(x_1, \ldots, x_l)$, для всех значений $d_{1_a} \in D_1$ и $d_{l_b} \in D_l$ таких, что выполняются унарные ограничения $c(x_1), c(x_l)$, и бинарные ограничения $c(x_1, x_l)$, имеется последовательность значений: $d_{2_c} \in D_2, \ldots, d_{l-1_c} \in D_l$, такая, что

$$c(x_2), \ldots, c(x_{l-1}), c(x_1, x_2), c(x_2, x_3), \ldots, c(x_{l-1}, x_l)$$

выполняются. На рис. 10 показан пример для пути (x_1, x_2, x_3). Начальное состо-

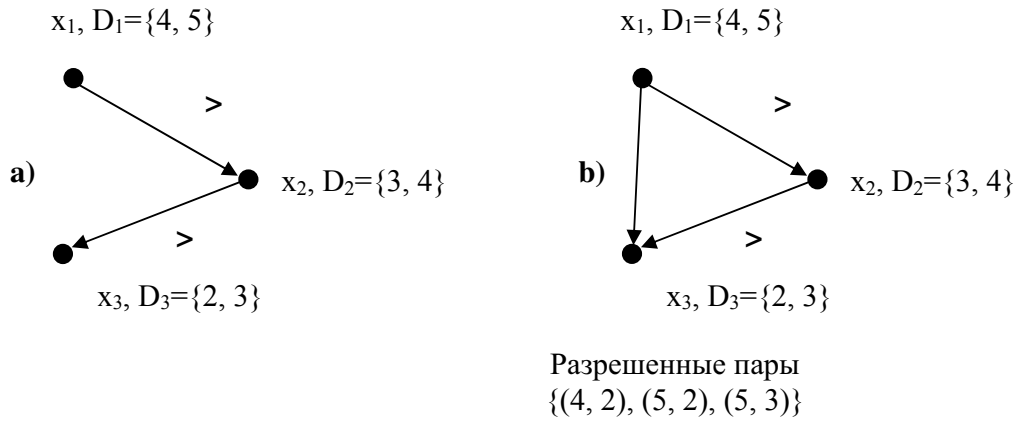


Рис. 10. Эффект вынуждения путевой совместности [283].

яние показано на рис. 10 a. Добавлено новое ограничение (рис. 10 b), которое не разрешает пару присвоений $x_1 = 4,\ x_3 = 3$: для этих присвоений не существует присвоения для x_2, удовлетворяющего бинарным ограничениям $c(x_1, x_2),\ c(x_2, x_3)$. Вынуждение дуговой совместности не разрешает этой пары присвоений.

Глобальная путевая совместность выполняется, если любая пара присвоений значений, которые совместны с прямым ограничением между двумя переменными, также совместна со всеми путями между двумя вершинами. Если любой путь длины 2 в полном графе ограничений является путе-совместным, то путевая совместность выполняется глобально (*Montanari*, 1974) [289].

Путевая совместность вынуждается с помощью изменения прямого ограничения между начальной и конечной точкой пути для того, чтобы запретить все пары назначений, которые нарушают требуемое условие. Ограничения могут быть обновлены путем добавления нового явного ограничения к задаче (см. рис. 10).

Первый алгоритм путевой совместности был предложен в (*Montanari*, 1974) [289], причем он эквивалентен алгоритму PC-1 (*Mackworth*, 1977) [262]. *Mackworth* [262] усовершенствовал эффективность PC-1 и предложил PC-2. PC-3 (*Mohr & Henderson*, 1986) [287] улучшает эффективность как в AC-4. PC-3 не полностью корректен. PC-4 (*Han & Lee*, 1988) [202] — это корректная версия. Дальнейшие усовершенствования алгоритма путевой совместности могут быть найдены в (*Chmeiss & Jegou*, 1996) [106]. *Mohr & Henderson* [287] предложили оптимальные алгоритмы для дуговой и путевой совместности, соответственно AC-4 и PC-4, которые были основаны на работе *Lauriere* [258].

3.3.6. k-совместность

Дуговая совместность может также пониматься как информация о том, насколько далеко частичное решение может всегда быть расширено. А именно, любое частичное решение, содержащее лишь одну зафиксированную переменную, может быть расширено с помощью фиксации любой другой переменной на соответствующим образом выбранном значении. Применение того же принципа для большего числа переменных приводит к понятию k-совместности, которое ввел *Freuder* [167], [168], [169], исследовавший и его связь со сложностью решения задач УО.

Можно определить k-совместность, рассматривая ограничения с k переменными (для $k = 3$ — путевая совместность, для $k > 3$ — гипер-дуговая совместность).

Определение (k-совместность) Задача УО *k-совместна*, если любой совместный кортеж значений любых $k - 1$ переменных может быть расширен путем задания значения любой одной из оставшихся переменных.

Задача УО *строго k-совместна* тогда и только тогда, когда она j-совместна для всех $j \leq k$. Строго n-совместная сеть, где n — число переменных ЗУО, называется глобально совместной.

Дуговая и путевая совместность соответствуют 2- и 3-совместности.

Заметим, что k-совместная ЗУО не обязательно разрешима, и наоборот, разрешимость задачи не влечет за собой совместности любого уровня, а не только 1-совместность.

3.4. Вычислительная сложность и поиск легко разрешимых задач удовлетворения ограничений

Одной из наиболее фундаментальных сложных проблем УО и программирования в ограничениях является изучение вычислительной сложности задач УО. В [262] было показано, что класс всех задач УО является NP-трудным, так что вряд ли существуют эффективные (полиномиальные) алгоритмы общего назначения для решения всех видов задач УО.

Целью многих исследований была идентификация классов легко разрешимых задач УО, которые могут быть решены за полиномиальное время (см. обзоры [255], [263], [364]). Подобные классы задач обычно описываются либо деревьями, древовидными графовыми структурами ([141], [170], [326]) либо определенной комбинацией алгебраических операторов [227].

Для некоторых классов задач УО было показано, что использование распространения ограничений гарантирует поиск без возвратов (backtrack-free search) [136]. В результате сравнения многих методов графовой декомпозиции показано, что наилучшими являются методы, основанные на использовании понятия *гиперграфовой декомпозиции* (hypertree decomposition), разработанного в теории баз данных [195].

Формализация понятия *языка ограничений* (constraint language)[11] может дать информацию, является ли задача УО, выраженная с помощью определенного языка ограничений, легко разрешимой [111], [218], [226], [227], [229], [230], [231], [232].

Хотя общая задача УО является NP-полной, во многих приложениях возникающие задачи УО имеют специальный вид, что позволяет разработать более эффективные алгоритмы их решения [90], [112], [260], [310]. Известно, что бинарные задачи УО с древовидной структурой легко разрешимы ([324], глава 7). Подкласс общей задачи УО, которые могут решены за полиномиальное время, а также распознан за полиномиальное время, называется *легко разрешимым подклассом*.

Существуют два основных направления поиска специальной структуры задач УО, которые могут быть легко разрешимыми: выделение структурных свойств задач УО (описываемых топологией графа ограничений), гарантирующих легкую разрешимость независимо от того, какие типы ограничений включены в рассмотрение [115], [140], либо выделение типов ограничений, достаточно ограничительных для гарантирования легкой разрешимости независимо от того, как они сочетаются [90], [161].

Благодаря исследованиям в этой области были выделены многие классы легко разрешимых задач УО, основываясь на структурных свойствах (см. [324]). Подобные классы могут встретиться, в частности, в случае, когда имеется некоторая специальная структура взаимосвязи ограничений (их пересечений). Наиболее естественным для изучения взаимосвязей ограничений является аппарат гиперграфов. Для класса задач УО, у которых арность ограничений ограничена некоторой фиксированной константой (таких, как бинарные задачи УО) было показано, что

[11] Язык ограничений ЗУО — это множество отношений в ограничениях ЗУО [113]

(при выполнении некоторых дополнительных ограничений) единственным классом структур, гарантирующим легкую разрешимость, является класс структур с ограниченной *древовидной шириной* [198], [209], [23].

Второе направление исследований состоит в рассмотрении задач УО с более ограниченными типами ограничений. Естественным подходом изучения типов ограничений является математическая теория отношений и соответствующие им алгебры, применение которых позволило получить характеризации того, какие виды ограничений обеспечивают легкую разрешимость независимо от их взаимосвязи. Множество типов ограничений с этим свойством называется *легко разрешимым языком ограничений*. В общем было показано, что любой легко разрешимый язык ограничений должен иметь определенные алгебраические свойства, называемые *полиморфизмами* [225].

Фундаментальной открытой проблемой в этой области является четкое определение типов ограничений в задачах УО, которые могут быть решены за полиномиальное время. Эта проблема важна с теоретической точки зрения, так как она помогает выяснить границу между легкой и трудной разрешимостью для широкого круга задач комбинаторного поиска [125], [161], [225], [252]. Кроме того, она важна и с прикладной точки зрения, так как она позволяет разрабатывать языки программирования в ограничениях, использующие существование различных семейств легко разрешимых ограничений для поиска более эффективных методов решения [260], [310]. В ряде публикаций [228], [230], [227], [233] было показано, что сложность языков ограничений над конечными доменами может быть охарактеризована, используя алгебраические свойства отношений. Согласно [114], первый этап алгебраического подхода к языкам ограничений использует известную идею, состоящую в том, что к данному начальному множеству отношений ограничений могут добавляться дополнительные ограничения без изменения сложности соответствующего класса задач. На самом деле, было показано, что можно добавить все ограничения, полученные из исходных ограничений, используя некоторые простые правила. Большие множества отношений, полученные при использовании этих правил, называются реляционными клонами [146], [306]. Первый этап алгебраического подхода состоит в том, что достаточно анализировать сложность лишь для тех множеств отношений, которые являются реляционными клонами. Второй этап состоит в том, что реляционные клоны могут быть описаны с помощью их полиморфизмов, являющихся алгебраическими операциями над тем же основным множеством [225],[227]. Помимо того, что полиморфизмы дают точный и удобный метод описания больших семейств отношений, они отражают также определенные аспекты структуры отношений, которые могут быть использованы для создания эффективных алгоритмов. Указанная связь между реляционными клонами и полиморфизмами играет сейчас ключевую роль при определении многих легко разрешимых классов ограничений и разработке эффективных алгоритмов их решения соответствующих задач УО [96],[92],[93], [91],[126],[228]. Полная характеризация всевозможных легко разрешимых языков ограничений была установлена в следующих случаях: консервативные языки ограничений (т.е. языки ограни-

чений, содержащие все унарные ограничения) [94], и языки ограничений над 2-элементным доменом [332] или над 3-элементным доменом [96]. Кроме того, были разработаны новые эффективные алгоритмы для решения задач УО специального вида как для конечных, так и для бесконечных доменов [75], [92], [94], [112], [121], [126], [251]. Последний этап алгебраического подхода связывает язык ограничений и конечные универсальные алгебры (см. раздел 6.4.3 работы [114]). Язык конечных универсальных алгебр предоставляет множество очень удобных новых средств для анализа сложности ограничений, в том числе глубокие структурные результаты, разработанные для классификации структуры конечных алгебр [210], [274], [358].

В случае, когда множество разрешённых отношений инвариантно относительно некоторой мальцевской операции[12], показывается, что соответствующая задача УО может быть решена за полиномиальное время [2].

Однако многие задачи УО не обладают достаточно ограниченной структурой или используют недостаточно ограниченный язык ограничений, чтобы попасть в какой-либо из классов легко разрешимых задач. Они могут, тем не менее, иметь свойства, обеспечивающие эффективность их решения, но эти свойства относятся как к структуре, так и к типам ограничений. Эти свойства иногда называются гибридными причинами легкой разрешимости [301], они гораздо менее изучены, чем свойства языка и описанные выше структурные свойства.

В [122] описан новые гибридные легко разрешимые классы бинарных ЗУО, обобщающие задачи с древовидной структурой, а также гибридные легко разрешимые классы, основанные на языке. Эти новые классы основаны на локальных свойствах упорядоченных троек переменных. При этом задача определения упорядочения переменных, для которого выполняются эти свойства, решается за полиномиальное время.

4. Структура и методы декомпозиции задач УО

Исследования, касающиеся структуры и сложности задач УО, начались с работы (*Freuder*, 1985) [169], где было показано, что поиск на деревьях с совместными дугами происходит без каких-либо возвратов. Аналогичные результаты применительно к ациклическим гиперграфам были получены в области теории баз данных (*Beeri et al*, 1983) [55]. Со времени публикации этих работ достигнут значительный прогресс в части получения более общих результатов, касающихся связи между сложностью решения задачи УО и структурой ее графа ограничений.

В данном разделе рассматриваются способы, позволяющие использовать для быстрого поиска решений структуру самой задачи, представленную в виде графа ограничений. Большинство описанных здесь подходов является очень общими и применимыми для решения других дискретных задач, кроме УО.

[12] Мальцевская операция — это тернарная операция φ, удовлетворяющая $\varphi(x, y, y) = \varphi(y, y, x) = x$.

4.1. Графовые методы

Графовые методы используют структуру графа ограничений при решении задачи УО. Ниже рассмотрены графовые методы, использующие то, что задача УО с древовидным графом ограничений может быть решена за линейное время (см. (*Miguel & Shen* [283]), раздел 4.4), причем древовидная структура находится или строится из графа ограничений данной задачи УО.

Для разбиения задачи УО на независимые подзадачи, можно рассмотреть связные компоненты графа ограничений. Каждый компонент соответствует одной подзадаче УО_i. Если присваивание S_i является решением УО_i, то $\cup_i S_i$ является решением $\cup_i \text{УО}_i$. Предположим, что каждая подзадача УО_i имеет c переменных из общего количества n переменных, где c — константа. В таком случае должно быть n/c подзадач, и для решения каждой из них требуется, самое большее, объем работы d^c[13]. Поэтому общий объем работы измеряется величиной $O(d^c n/c)$, которая линейно зависит от n; без такой декомпозиции общий объем работы измерялся бы величиной $O(d^n)$, которая экспоненциально зависит от n. Приведем более конкретный пример из [12]: декомпозиция булевой задачи УО с $n = 80$ на четыре независимые подзадачи с $c = 20$ сокращает продолжительность поиска решения в наихудшем случае от величины, равной времени существования всей Вселенной, до величины, меньшей одной секунды. Поэтому полностью независимые подзадачи являются очень привлекательными, но встречаются редко. В большинстве случаев подзадачи любой задачи УО связаны друг с другом. Простейшим случаем является тот, в котором граф ограничений является деревом: любые две переменные связаны не больше чем одним путем. Алгоритм решения блочной ЗУО имеет следующие этапы.

- Выбрать в качестве корня дерева любую переменную и упорядочить переменные от корня до листьев таким образом, чтобы родительская вершина каждой вершины в дереве предшествовала этой вершине в таком упорядочении. Обозначить эти переменные по порядку как $X_1, \ldots, X_n$. Теперь каждая переменная, кроме корня, имеет только одну родительскую переменную.

- В цикле по j от n до 2 применять проверку совместности к дугам (X_i, X_j), где X_i — родительская вершина вершины X_j, удаляя значения из области определения D_i по мере необходимости.

- В цикле по j от 1 до n присваивать X_j любое значение, совместное со значением, присвоенным X_i, где X_i — родительская вершина вершины X_j.

Теперь, после создания эффективного алгоритма для деревьев, следует рассмотреть вопрос о том, можно ли каким-то образом приводить к древовидным структурам более общие графы ограничений. Существуют два основных способа выполнения этой задачи; один из них основан на удалении вершин, а другой — на слиянии

[13] d — размер доменов переменных.

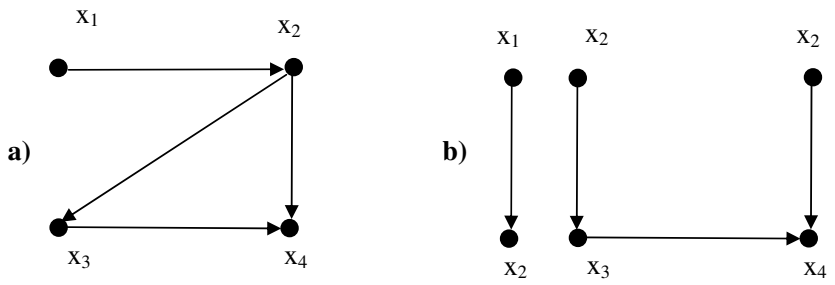


Рис. 11. Эффект фиксации множества разрыва цикла.

вершин друг с другом и образовании супер-вершин. Первый подход предусматривает присваивание значений некоторым переменным так, чтобы оставшиеся переменные образовывали дерево.

4.2. Метод множеств разрыва цикла

Метод множеств разрыва цикла (cycle cutset) (*Dechter*, 1990) [133] основан на том, что присвоение значения переменной устраняет ее из дальнейшего рассмотрения в этой ветви дерева поиска. Это сильно меняет связность оставшейся части графа ограничений.

Множество разрыва цикла графа ограничений — это множество вершин, при исключении которых (с помощью фиксации соответствующих переменных) остается дерево. Рассмотрим пример на рис. 11 a. Фиксирование значения переменной x_2 «блокирует» путь, проходящий через эту вершину, так что имеется только один путь между любыми двумя переменными (вершинами). В результате получается граф ограничений (рис. 11 b), в котором зафиксированная сейчас переменная x_2 повторяется для каждой смежной переменной. Если множество фиксированных переменных совпадает с множеством разрыва цикла, то оставшийся граф имеет древовидную структуру и соответствующая ему задача может быть решена очень эффективно. Этот метод находит локально совместное присвоение для переменных из множества разрыва цикла и решает оставшуюся (древовидную) задачу. Если не найдено ни одного решения, то должно быть найдено другое совместное присвоение для переменных из множества разрыва цикла и делается попытка решения задачи, соответствующей оставшейся части графа. Нахождение минимального множества разрыва цикла является NP-полной задачей. В связи с этим, целесообразно встраивать метод множества разрыва цикла в метод дерева поиска, такой, как поиск с возвратами. В случае, когда множество зафиксированных переменных образует множество разрыва цикла, оставшаяся часть задачи может быть быстро решена. По сравнению с поиском с возвратами на случайно сгенерированных задачах, метод поиска с возвратами, использующий множества разрыва цикла, в среднем на 20% эффективнее [284]. По сравнению с наивной версией

алгоритма обратного перехода, его комбинация с множествами разрыва цикла в среднем на 25% эффективнее [284].

Общий алгоритм решения указанным способом описан ниже [12].

- Выбрать подмножество S из множества переменных задачи УО, такое, что граф ограничений после удаления S становится деревом. Подмножество S называется *множеством разрыва цикла.*
- Для каждого возможного присваивания переменным в S, которое удовлетворяет всем ограничениям в S, выполнить следующее:
  - удалить из областей определения оставшихся переменных любые значения, несовместные с данным присваиванием для S;
  - если оставшаяся задача УО имеет решение, вернуть это решение вместе с присваиванием для S.

Если множество разрыва цикла имеет размер c, то общее время работы алгоритма составляет $O(d^c \cdot (n-c)d^2)$. В том случае, если граф по своей форме «очень близок к дереву», то множество с будет небольшим, а экономия времени по сравнению с прямым поиском с возвратами окажется огромной. Но в наихудшем случае количество элементов с может достигать $(n-2)$. Хотя задача поиска наименьшего множества разрыва цикла является NP-трудной, но известно несколько эффективных алгоритмов решения этой задачи. В целом данный алгоритмический подход носит название *определения условий выбора множества разрыва цикла* (cutset conditioning).

4.3. Связные компоненты

Выделение связных компонентов графа (*Dechter & Pearl* [140]; *Freuder* [168]) может быть полезно при решении задач УО. Точка сочленения x_i в графе ограничений — эта такая вершина, что все пути между остальными вершинами должны проходить через нее. Разделимый граф содержит точку сочленения, а связный граф — нет. Связный компонент — это подграф без точек сочленения. *Even* [159] предложил алгоритм со сложностью $O(e)$ для нахождения всех связных компонентов и точек сочленения (здесь e — число ограничений). После нахождения всех связных компонентов, необходимо найти все возможные решения для каждого из них. Решение всей задачи УО может быть получено с помощью склеивания решений для компонентов (с учетом совместности значения для точки сочленения смежных компонентов). *Dechter & Pearl* [140] предложили алгоритм со сложностью $O(n \cdot d^r)$, использующий описанный подход, где n — число переменных, d — размер максимального домена переменных, а r — размер наибольшего компонента.

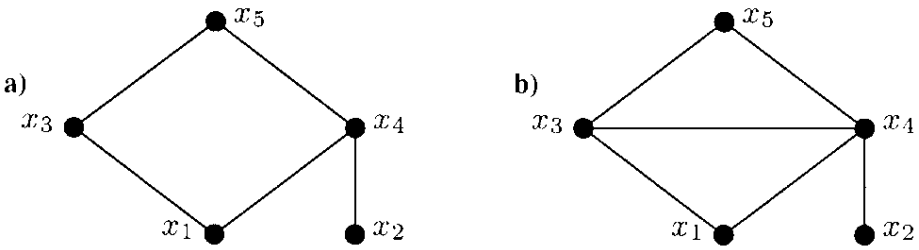


Рис. 12. Построение хордального первичного графа.

4.4. Древовидная кластеризация

Древовидная кластеризация (tree-clustering) (*Dechter & Pearl* [141]) преобразует произвольную n-арную задачу УО в ациклическую форму в представлении в виде двойственного графа путем формирования кластеров из переменных, причем граф взаимосвязей имеет структуру дерева. n-арная задача УО называется ациклической (см. (*Miguel & Shen* [283])), если ее первичный граф ограничений обладает следующими свойствами:

- *хордальность*: каждый цикл, содержащий не менее 4 вершин, содержит хорду (ребро, соединяющее 2 непоследовательные вершины цикла).
- *конформность*: каждая из максимальных клик графа соответствует одному ограничению в задаче УО.

Для достижения указанных свойств может быть использован *алгоритм триангуляции* (*Tarjan & Yannakakis* [361]), используя два шага: а) Найти упорядочение с максимальной степенью (maximum cardinality) и б) рекурсивно добавить ребра между вершинами, соединяющими вершины, с большим порядком согласно указанному упорядочению. На рисунках 12 a и b (из (*Dechter & Pearl* [141])) показан первичный граф ограничений задачи УО соответственно до и после процесса триангуляции. Максимальными кликами этого графа являются кластеры, которые составляют ограничения преобразованной задачи УО (см. рис. 13 a). Для построения дерева из этого представления достаточно удалить лишние ребра из графа, т.е. те ребра, удаление которых не оказывает влияния на множество решений. В двойственном графе дуга избыточна, если множество переменных, которым она помечена, является общим для всех дуг на отдельном пути между конечными вершинами дуги.

На рис. 13 b) показан один способ построения дерева для задачи УО. Сложность этой процедуры $O(n \cdot r \cdot d^r)$, где r — размер наибольшей максимальной клики.

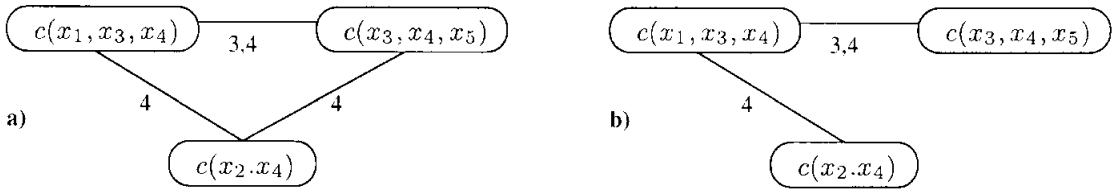


Рис. 13. Построение дерева из двойственного графа.

4.5. Элиминация переменных

4.5.1. Сегментная элиминация

Сегментная элиминация (Bucket elimination) [137], [142] — по сути является аналогом несериального динамического программирования [64], [21], [22], для решения задач УО. Этот алгоритм может найти все возможные решения задачи. Алгоритм сегментной элиминации имеет экспоненциальные оценки временно́й и емкостной сложности по индуцированной ширине графа ограничений. Большая арность промежуточных ограничений, которые хранятся в памяти в виде таблиц, является главным препятствием для применения метода сегментной элиминации на практике. При достаточно небольшой арности ограничений алгоритм сегментной элиминации достаточно эффективен [257].

4.5.2. Метод Зейделя — процедура инвазии

Еще одним известным декомпозиционным алгоритмом является *алгоритм инвазии* Зейделя (*Seidel*, 1981) [337], который находит все решения бинарной задачи УО. Метод разбивает данную задачу УО на множество подзадач УО, связанных линейным графом (путем — частным случаем дерева).

Процедура инвазии[14] (*Seidel*, 1981) [337] предназначена для нахождения всех решений задачи УО и основана на последовательности подграфов $G_1, \ldots, G_n$, называемой *инвазией* графа ограничений G с n вершинами. Число вершин в подграфе G_i должно быть i, причем G_i должен быть подграфом G_{i+1}. *Фронт* F_i — это множество вершин подграфа G_i, смежных с вершинами, не принадлежащих G_i, а длина фронта f_i — это число вершин в F_i . Длина фронта инвазии — это максимум длин ее отдельных фронтов. Рассмотрим пример на рис. 6а). Инвазия может быть построена следующим образом: $G_1 = \{x_1\}$, $G_2 = \{x_1, x_2\}$, $G_3 = \{x_1, x_2, x_3\}$, $G_4 = \{x_1, x_2, x_3, x_4\}$. Найдем фронты, соответствующие подграфам: $F_1 = \{x_1\}$, $F_2 = \{x_2\}$, $F_3 = \{x_2\}$, $F_4 = \{\ \}$. Заметим, что F_n всегда пустое множество, так как G_n совпадает с исходным графом ограничений. Для данной инвазии алгоритм Зейделя строит *граф решений* S, представляющий множество решений исходной задачи УО. Вершины графа S сгруппированы в виде множеств

[14] инвазия — нашествие.

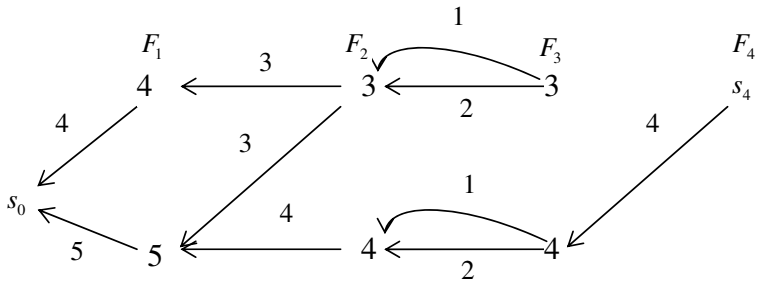


Рис. 14. Граф решений для алгоритма Зейделя.

$S_0, \ldots, S_n$, таких, что S_i $(i \neq 0,\ i \neq n)$ является множеством кортежей присвоенных значений для фронта F_i . S_n и S_0 содержат по одной вершине, s_n и s_0, соответственно (это — начало и конец решения). Направленное ребро из кортежа из S_i к кортежу из S_{i-1} помечается присвоением значения переменной x_i (где x_i — переменная в G_i, которая не принадлежит G_{i-1}). Это присвоение должно удовлетворять всем ограничениям, содержащим x_i и все переменные из фронта F_{i-1} для данных присвоений, представленных конкретному кортежу из S_{i-1}. Путь из s_n в s_0 с фиксированием x_i на значениях, соответствующих каждому пройденному ребру, соответствует одному решению. Граф решений задачи с рис. 6 а) показан на рис. 14. При построении S_i каждый кортеж предыдущего множества кортежей S_{i-1} рассматривается по очереди для всех возможных присвоений x_i. Кортеж добавляется к S_i для каждого присвоения переменной x_i, которое удовлетворяет всем ограничениям с конкретным присвоением кортежа в S_{i-1}. Добавляется ребро, соединяющее каждый вновь построенный кортеж из S_i с кортежем из S_{i-1}, и пометим его величиной x_i. Временна́я сложность этого алгоритма — $O(e \cdot d^{f+1})$, где e — число ограничений, d — максимальный размер домена, а f — длина фронта инвазии. Как и для других декомпозиционных методов, эффективность алгоритма инвазии зависит от максимального размера подзадач УО.

Таким образом, эффективность этого алгоритма зависит от нахождения хорошей инвазии, т.е. инвазии с небольшой длиной фронта. К сожалению, возможность существования хороших инвазий для любых графов не гарантируется — полный граф с n вершинами имеет инвазии только с длиной фронта $n-1$. Для некоторых задач, таких как задача о n ферзях, максимальный размер подзадач УО будет равен n.

4.6. Древовидная декомпозиция

Понятие ширины дерева было введено в теории графов Робертсон и Сеймур (*Robertson & Seymour*, 1986) [320]. *Dechter & Pearl*, 1987, 1989) [140], [141], основываясь на работе Фрейдера (*Freuder*, 1985) [169], применяли то же понятие (названное ими *индуцированной шириной*) к задачам УО и разработали подход, ис-

пользующий *древовидную декомпозицию* (tree decomposition) графа ограничений и создании множества связанных подзадач. Каждая подзадача решается независимо, а затем итоговые решения комбинируются. Как и большинство алгоритмов, действующих по принципу «разделяй и властвуй», этот алгоритм работает успешно, если подзадачи не слишком велики. Любая древовидная декомпозиция должна удовлетворять трем приведенным ниже требованиям.

- Каждая переменная из первоначальной задачи появляется по меньшей мере в одной из подзадач.
- Если две переменные первоначальной задачи связаны ограничением, то должны появиться вместе (наряду с этим ограничением) по меньшей мере в одной из подзадач.
- Если какая-то переменная появляется в двух подзадачах в дереве, то должна появляться в каждой подзадаче вдоль пути, соединяющего эти подзадачи.

Первые два условия гарантируют, что в декомпозиции будут представлены все переменные и ограничения. Третье условие на первый взгляд кажется довольно формальным, хотя оно просто отражает то ограничение, что любая конкретная переменная должна иметь одно и то же значение в каждой подзадаче, в которой появляется; соблюдение этого ограничения гарантируют связи между подзадачами в дереве.

Каждая подзадача решается независимо; если какая-либо из них не имеет решения, то известно, что вся задача также не имеет решения. Если удается решить все подзадачи, то может быть предпринята попытка составить глобальное решение следующим образом. Прежде всего, каждая подзадача рассматривается как «супер-переменная», областью определения которой является множество всех решений этой подзадачи. Затем решается задача с ограничениями, связывающими подзадачи; для этого используется эффективный алгоритм для деревьев, приведенный выше. Ограничения, связывающие подзадачи, указывают на то, что решения подзадач должны быть согласованными по их общим переменным. Любой конкретный граф ограничений допускает большое количество древовидных декомпозиций; при выборе декомпозиции нужно стремиться к тому, чтобы подзадачи были как можно меньше. *Ширина дерева* древовидной декомпозиции графа на единицу меньше размера наибольшей подзадачи; ширина дерева самого графа определяется как минимальная ширина дерева среди всех его древовидных декомпозиций. Если граф имеет ширину дерева w и дана соответствующая древовидная декомпозиция, то соответствующая задача может быть решена за время $O(nd^{w+1})$. Это означает, что *задачи УО с графами ограничений, характеризующимися конечной шириной дерева, могут быть решены за полиномиальное время.* К сожалению, задача поиска декомпозиции с минимальной шириной дерева является NP-трудной, но существуют эвристические методы, которые хорошо работают на практике.

4.7. Другие методы декомпозиции

Опираясь на работы (*Robertson & Seymour*, 1986) [320], (*Dechter & Pearl*, 1987, 1989) [140], [141], и на результаты из области теории баз данных, Готтлоб и др. (*Gottlob et al.*, 1999) [195], [196] разработали понятие *ширины гипердерева*, которое основано на методе характеризации задачи УО с помощью гиперграфа. Они не только показали, что любую задачу УО с шириной гипердерева w можно решить за время $O(n^{w+1} \log n)$, но и обосновали утверждение, что критерий ширины гипердерева превосходит все ранее предложенные критерии «ширины» в том смысле, что в некоторых случаях ширина гипердерева является конечной, тогда как ширина, определяемая другими критериями, — неограниченной. В (*Pearson & Jeavons*, 1997) [301] приведен обзор легко разрешимых классов задач УО и описаны как методы структурной декомпозиции, так и методы, основанные на свойствах областей определения или свойствах самих ограничений.

В работе [115] предложен общий класс методов структурной декомпозиции для задач УО, названный *осторожной декомпозицией* (guarded decomposition). Показано, что многие существующие методы декомпозиции могут характеризоваться в терминах нахождения осторожной декомпозиции, удовлетворяющей некоторым специальным дополнительным условиям.

5. Программирование в ограничениях

5.1. Основные подходы и состояние дел

Программирование в ограничениях (constraint programming) — программная технология для декларативного описания и эффективного решения больших комбинаторных задач в областях планирования и календарного планирования.

Цель программирования в ограничениях (см. (*Colmerauer*, 1990) [120], (*Jaffar & Lassez*, 1987) [220], (*Van Hentenryck*, 1989) [368]) состоит в разработке языков программирования для задания ограничений и процедур поиска задач УО. Процедура поиска в УО неявно описывает обход дерева поиска, не описывая при этом, *как* обходить это дерево, это уже — задача стратегий поиска. Обычно дерево поиска обходится с помощью поиска в глубину. Тем не менее, имеются и другие стратегии поиска (см. (*Harvey & Ginsberg*, 1995) [206], (*Korf*, 1996) [249], (*Meseguer*, 1997) [279], (*Walsh*, 1997) [381]), представляющие большой интерес. В частности, было показано, что поиск с ограниченной несовместностью (limited discrepancy search (LDS)) и его вариации достигают на некоторых прикладных задачах существенно лучших результатов по сравнению с поиском в глубину. Благодаря успешному решению многих прикладных задач (см. (*Harvey & Ginsberg*, 1995) [206], (*Laburthe & Caseau*, 1998) [256], (*Perron*, 1999) [304]), стратегии поиска стали неотъемлемой частью языков программирования в ограничениях (см. (*Laburthe & Caseau*, 1998) [256], (*Perron*, 1999) [304], (*Schulte*, 1997) [336], (*Van Hentenryck et al.*, 2000) [375]). В частности, программная система **Oz** впервые ввела спецификацию общих

стратегий поиска, которые могут быть заданы независимо от процедуры поиска (*Schulte*, 1997) [335]. В результате, стратегия поиска может быть применена к многим процедурам поиска и процедура поиска может быть оснащена несколькими стратегиями поиска. Поисковый язык **SALSA**[15] (*Laburthe & Caseau*, 1998) [256] также содержит ряд заданных общих стратегий поиска. В работе (*Perron*, 1999) [304] показано, что много стратегий поиска может быть задано с помощью всего двух функций: функции оценки, которая ставит оценку в соответствие каждой вершине дерева поиска и функции подвешивания, которая неявно задает, какая вершина будет следующей.

Основные идеи, лежащие в основе программирования в ограничениях, просты [97]: декларативное представление ограничений задачи, совмещенное с общими методами решения типа хронологического поиска с возвратами или локального поиска. Программирование в ограничениях имеет множество сильных сторон: мощные языки моделирования, позволяющие представить сложные и динамические прикладные задачи; быстрые методы вывода общего назначения типа вынуждения дуговой совместности, для усечения части пространства поиска; быстрые методы вывода специального назначения, связанные с глобальными ограничениями; гибридные методы, сочетающие преимущества методов программирования в ограничениях и подходов исследования операций; методы локального поиска, позволяющие быстро находить решения, близкие к оптимальным; большой диапазон расширений типа мягких ограничений и распределенного решения ограничений, с помощью которых возможно более адекватное моделирование практических задач.

5.2. Логическое программирование в ограничениях

За последние 10-15 лет программирование в ограничениях [266] прошло путь от научной идеи до мощной парадигмы программирования, которая все больше используется для моделирования и решения многих трудных практически важных задач. Под программированием в ограничениях (constraint programming) понимается программирование алгоритмов решения задач УО [81]. Программирование в ограничениях считается в настоящее время одним из стратегических направлений информатики [372].

Логическое программирование в ограничениях (ЛПО) (Constraint Logic Programming (CLP)) [97] было предложено в работах *Jaffar & Lassez* [220], *Marriott & Stuckey* [266], [37] как расширение логического программирования. Логическое программирование — это парадигма программирования, основанная на логике [116]. Его конструкты — булевы импликации (т.е. $q \Rightarrow p$ или $p :- q$), композиций, использующих булевы операции конъюнкции и дизъюнкции. Логическое программирование может рассматриваться как процедурный язык, в котором процедуры — это булевы функции, результат работы программы всегда *истина* или *ложь*.

Программирование в ограничениях является декларативной парадигмой, поз-

[15] **SALSA**: **A** **L**anguage for **S**earch **A**lgorithms.

воляющей выразить многие прикладные задачи с помощью ограничений, т.е. отношений между переменными задачи. Подобный путь задания задач может производиться для многих традиционных сред программирования, но лучше всего подходят те среды, которые используют декларативную парадигму программирования, подобные парадигме логического программирования [261], которая также основана на использовании отношений.

Программирование в ограничениях (*Marriott & Stuckey*, 1998) [266] состоит из двух необходимых компонентов: решателя ограничений и машины поиска.

Хотя ранее и были разработаны некоторые языки для поддержания ограничений специального вида (**ThingLab** [78], **Bertrand** [259], и **ALICE** [258]), но не было никакой общей лингвистической поддержки для решения задач УО и распространения ограничений до разработки схемы ЛПО. Эта схема может рассматриваться как оболочка логического программирования, поддерживаемая системой ограничений, с помощью которой может записывать и решать любые ограничения. По этой причине схема ЛПО может обозначаться как ЛПО(X), где X определяет класс ограничений, с которыми мы хотим работать.

Понятно, что описание задачи в виде системы ограничений является декларативным, причем описывается, «что» должно выполняться, но не указывается, «как» это сделать. Разработано много методов решения, позволяющих (точно или приближенно) находить одно или несколько решений задачи УО. Наиболее известным является описанный выше поиск с возвратами, который может в общем случае обойти все дерево возможных значений для нахождения решения, или установления факта отсутствия решений.

Тот факт, что системы удовлетворения ограничений и логическое программирование [261] — декларативные парадигмы, использующие отношения и поиск с возвратами, сделал их интеграцию легкой и естественной. Чрезвычайно важным явился факт объединения ЛПО с областью искусственного интеллекта, называемого *удовлетворение ограничений*. ЛПО — расширение логического программирования, в котором унификация может быть заменена или дополнена другими видами ограничений, определенных на доменах (областях определения) используемых переменных. ЛПО и удовлетворение ограничений имеют общие цели, а также используют компактное декларативное формулирование комбинаторных задач и эффективные стратегии поиска.

Так, распространение ограничений используется как для решения ограничений с конечными доменами, так и в программировании в ограничениях.

Общепринятый подход к решению задач УО сочетает дерево поиска с возвратами с распространением ограничений. Это реализовано в системах программирования в ограничениях с конечными доменами, таких как **SICStus Prolog** [99], **ILOG Solver** [216], и **Gecode** [183], успешно использованных для решения многих практических задач.

Когда ЗУО встроена в логическое программирование, то ограничение может быть определено в программе как множество фактов или даже множество правил.

Основная идея ЛПО состоит в расширении логического программирования

так, что подстановки заменяются ограничениями, унификация заменяется решением задачи УО, а все семантические свойства логического программирования (в основном, существование декларативных, денотациональных и процедуральных семантик и эквивалентность этих трех объектов) все еще выполняются.

Jaffar & Lassez (1987) [220] показали, что теоретические основы языков логического программирования остаются справедливыми для языков ЛПО. Некоторые языки ЛПО до сих пор используются специалистами по логическому программированию: **PROLOG III** и **IV**, **CLP(R)**[16], **CHIP**[17] **CLP(BNR)**[18].

5.3. Программирование в ограничениях с конечными доменами

Многие языки ЛПО были созданы для работы с ограничениями с конечными доменами. ЛПО с конечными доменами впервые было реализовано в конце 1980-х годов *Van Hentenryck* [368] в языке **CHIP** [149]. Другими примерами могут служить языки **ECLiPSe**[19] [158], [378] и **CLP(FD)**[20] [109].

Одним из главных преимуществ использования системы ограничений в рамках языка ЛПО является контроль за процессом поиска. Использование методов частичного поиска (partial search) существенно улучшило методы поиска. Методы частичного поиска ограничивают число альтернатив, изучаемых в различных вершинах дерева поиска. Таким образом, обычный поиск в глубину с возвратами, используемый в ЛПО, который иногда проверяет лишь небольшую часть дерева поиска, может быть заменен на другие поисковые схемы (такие как LDS [206] или поиск [57]), которые могут рассматривать много различных путей в дереве поиска. Выше, в разделе 2.2 описаны глобальные ограничения, позволяющие снизить перебор при решении задач УО.

Языки, подобные **CHIP** [149], имеют сложную морфологию глобальных ограничений [56]. Другие языки, такие как **SICStus Prolog** [99], **ECLiPSe** [158], [378], **IF Prolog** `http://www.ifcomputer.de/Products/Prolog/home_en.html`, а также **Oz** [207], [353], [204], [292], включают некоторые из глобальных ограничений, используемых в **CHIP**. Работы по использованию распространения ограничений для глобальных ограничений (в частности, ограничения **all-different**) показывают практическую ценность использования ограничений этого вида [313], [338]. Глобальные ограничения были использованы также для частичных задач УО, в которых некоторые ограничения могут нарушаться при нахождении решения [43].

Задачи с динамическими ограничениями — это задачи УО, изменяющиеся во времени, с помощью добавления или исключения некоторых ограничений, либо комбинации этих двух операций. В то время как добавление ограничений часто используется на каждом шаге вычислений, исключение ограничений рассматривается редко, хотя это было бы весьма полезно для интерактивных или динами-

[16] **CLP(R)**: **C**onstraint **L**ogic **P**rogramming (**R**eals).
[17] **CHIP**: **C**onstraint **H**andling **i**n **P**rolog.
[18] **CLP(BNR)**: **C**onstraint **L**ogic **P**rogramming (**B**ooleans, **N**aturals and **R**eals).
[19] **ECLiPSe**: **C**onstraint **L**ogic **P**rogramming **S**ystem.
[20] **CLP(FD)**: **C**onstraint **L**ogic **P**rogramming over **F**inite **D**omains.

ческих систем, в том числе при необходимости пересчета решения при изменении условий задачи УО, а также поиска устойчивых решений [380]. По этой причине, в ряде работ введено понятие исключения (или отмены) ограничений, и предложены эффективные алгоритмы для этого в рамках классических языков ЛПО. Например, программная схема **CLP(FD)** была расширена как в плане синтаксиса, семантики и возможности отмены ограничений в [184].

5.4. Нелинейные ограничения в языках программирования в ограничениях.

Возможность использовать и решать нелинейные ограничения имеет важное значение во многих областях приложений (экономика, механика, химическое производство, технические приложения и др.). Обычно решение задач такого типа достаточно сложно и требует привлечения больших вычислительных ресурсов. Помимо классических методов, в настоящее время здесь используются методы интервального анализа [295]. По этой же причине, интервальные ограничения [62] используются в таких языках ЛПО, как **ECLiPSe** [158], [378],[385] и **CLP(BNR)**, а языки **Helios** [370] и **Numerica** [371] были разработаны для решения систем нелинейных ограничений, используя алгоритмы интервального анализа [295].

5.5. Моделирование.

Чрезвычайно важно иметь хороший язык моделирования для записи задач УО. В настоящее время большинство систем программирования в ограничениях являются либо расширением декларативного языка программирования (типа **Prolog**), либо библиотеками, используемыми совместно с обычными языками программирования (такими как C/C++). Сейчас имеется много исследований по разработке языков моделирования ограничений, которые делают фазу моделирования более легкой и естественной (примеры: **Helios** [370], **Numerica** [371] и **OPL** [374]). Некоторые из этих языков используют средства визуализации для описания и генерирования программ в ограничениях. Например, система **FORWARD** календарного планирования нефтеперегонного завода [191] использует средство графической спецификации для **CHIP** [344], при этом модель нефтеперегонного завода вводится с помощью графического интерфейса. **CML**[21] — язык моделирования ограничений, который был разработан для облегчения постановки и модификации комбинаторных задач [32]: **CML**-спецификации задачи позволяют формулировать ограничения, отражающие выражения естественного языка, которые затем переводятся в эффективные **CHIP**-программы.

Другим языком моделирования является **VISUAL SOLVER** [373], который позволяет задать как постановку задачи, так и эвристики, используемые для поиска решений.

Другие подходы к моделированию реальных задач в виде задачи УО используют интерактивные методы: решатель задачи УО общается с пользователем, пред-

[21] **CML**: **C**onstraint **M**odelling **L**anguage.

лагающим частичные решения, и получает дальнейшую информацию о задаче от человека на основании оценки решателем этих частичных решений [173]. В числе других методов стоит упомянуть методы машинного обучения: при использовании мягких ограничений пользователь оценивает решения, а система изучает задачу УО [327], что весьма упрощает решение задачи с мягкими ограничениями.

Другой интересной схемой моделирования является схема, основанная на абдукционном ЛПО (Abductive Constraint Logic Programming (ACLP) (см. http://www.cs.ucy.ac.cy/aclp/), построенная на основе **ECLiPSe** [158].

5.6. Отладка.

Одной из распространенных проблем, возникающих у пользователей систем программирования в ограничениях является иногда непредсказуемое поведение модели с ограничениями: даже небольшие изменения программы или данных может приводить к существенному изменению производительности программы.

В то время, как простые приложения могут быть созданы достаточно быстро, ознакомление со всеми возможностями системы в ограничениях может занять много времени. Одной из попыток решения этих проблем является визуализация дерева поиска [100]. Программные средства типа **Oz Explorer** [335], поискового дерева **CHIP** [345] (обзор ранних приложений **CHIP** приведен в работе [150]), система, недавно разработанная в рамках системы **CIAO** [208], которая обеспечивает 3D-визуализацию с помощью **VRML** [352], позволяют намного глубже понять процесс поиска, поскольку они предоставляют достаточную информацию для анализа распространения ограничений и состояния доменов переменных в каждой вершине дерева.

5.7. Мягкие ограничения.

В работе [109] язык ЛПО **CLP(FD)**, работающий с конечными доменами, был обобщен для работы с ограничениями из полукольца, в результате получен язык **CLP(FD,S)** [22] [185], S — какое-либо полукольцо (semiring), выбираемое пользователем. Выбирая конкретное полукольцо, пользователь решает использовать конкретный класс мягких ограничений: нечетких, оптимизируемых, вероятностных, либо классических жестких ограничений. Для ограничений с оценками пока нет полных языков программирования в ограничениях для работы с ними, но имеется много методов нахождения хороших нижних оценок для их оптимальных решений [98], [143].

5.8. Языки запросов с ограничениями (Constraint query languages).

Поскольку ограничения используют отношения (как и базы данных (БД)), многие исследователи изучали взаимосвязь между ЛПО и БД. В этой области исследований имеются два главных направления. С одной стороны, результаты

[22] **CLP(FD,S)**: **S**emiring-based **C**onstraint **L**ogic **P**rogramming **L**anguage over **F**inite **D**omains.

из теории БД (например, по декомпозиции и выводу (получению) информации) были обобщены и применены для задач УО, что позволило разработать полезные инструменты для проверки некоторых классов задач УО на легкую разрешимость и эффективного решения этих задач [201], [110]. С другой стороны, классические языки обработки запросов в БД были расширены для работы с ограничениями, которые используются не только для установления целостности информации, но также как средство для представления множества кортежей БД компактным образом. Действительно, кортеж реляционной БД может рассматриваться как особый вид ограничения, а именно, конъюнкции ограничений-равенств между атрибутами кортежа и значений из данных доменов. Введение новых логических формул для выражения взаимоотношений (т.е. ограничений) между атрибутами БД приводит к определению БД с ограничениями (Constraint Databases) как новую интересную область исследований [83], [237].

Ограничения могут быть добавлены к реляционным БД на разных уровнях. На уровне данных ограничения могут выражать обобщенные кортежи, способные задавать бесконечные множества кортежей. Ограничения представляют собой мощный механизм для моделирования пространственных и темпоральных понятий, возникающих в теории БД, где имеется необходимость в компактном представлении континуальной информации [60], [199]. Для работы с ограничениями были расширены реляционное исчисление и реляционная алгебра классических БД [192], [236], [237], что привело к новым более выразительным языкам обработки запросов. Подобный язык был получен путем добавления линейных ограничений к **Datalog** в [317]. Другими примерами могут служить безопасный стратифицированный **Datalog**, рассмотренный в [316], а также интеграция линейных ограничений с БД, оптимизация запросов для которых была исследована в [84]. Оптимизация запросов была также рассмотрена в [362] в контексте дедуктивных запросов в БД с ограничениями.

5.9. Конкурирующие и распределенные системы.

Использование ограничений при моделировании поведения конкурирующих систем началось с развитием программирования в конкурирующих ограничениях (concurrent constraint programming), в которой конкурирующие агенты делят между собой хранилище ограничений, причем они могут либо сообщать о наличии дополнительных ограничений (tell operation), обмениваться сообщениями с другими агентами, или спрашивать, что имеется ли в наличии (ask operation), синхронизироваться с другими агентами. Языки программирования в ограничениях, такие как **Oz** [207], [353], [204], [292], **AKL**[23] [205], [224], [290], **CHR**[24] [175], **CIAO** [208] использовали и внедрили эту модель, обеспечив тем самым программную основу для конкурирующих систем.

Темпоральное конкурентное программирование в ограничениях (Temporal con-

[23] **AKL**: **A**ndorra **K**ernel **L**anguage.
[24] **CHR**: **C**onstraint **H**andling **R**ules.

current constraint programming) и недетерминистское темпоральное конкурентное программирование в ограничениях (non-deterministic temporal concurrent constraint programming (NTCC)) являются вариантами программирования в ограничениях, работающими со временем.

Расширениями этой модели являются распределенные системы [386] и системы реального времени [129].

5.10. Современные пакеты программного обеспечения

CHIP [149] был первым языком ЛПО, использующим распространение ограничений [368]. Другими примерами систем ЛПО могут служить библиотеки поддержки ограничений ILOG [216] и COSYTEC [123], а также языки логического программирования в ограничениях **Prolog III** [119], [120], **Prolog IV** [307], **CLP(R)** [222] , **ECLiPSe** [158], [378], **CIAO** [208], **CLP(FD)** [109]. Используя эти системы, многие сложные прикладные задачи были успешно решены с помощью технологий УО. В числе примеров упомянем проверку электронных схем, календарное планирование, распределение ресурсов, составление расписаний, управляющие системы, графические интерфейсы, а также множество комбинаторных задач [368], [377].

Синтаксис для выражения ограничений над конечными доменами зависит от языка программирования в ограничениях. Ниже приведена программа на языке **Prolog**, решающая классический кроссворд из главы 1 SEND+MORE=MONEY:

```
sendmore(Digits) :-
   Digits = [S,E,N,D,M,O,R,Y],  % Создание переменных
   Digits :: [0..9],            % Домены переменных
   S #\= 0,                     % Ограничение: S должно отличаться от 0
   M #\= 0,                     % Ограничение: M должно отличаться от 0
   alldifferent(Digits),        % все переменные должны принимать
                                % различные значения
    1000*S + 100*E + 10*N + D   % Другие ограничения
    + 1000*M + 100*O + 10*R + E
   #= 10000*M + 1000*O + 100*N + 10*E + Y,
   labeling(Digits).            % Начать поиск
```

Интерпретатор создает переменную для каждой буквы кроссворда. Символ :: используется для задания доменов этих переменных, так что они принимают множество значений $\{0, 1, 2, 3, \ldots, 9\}$. Ограничения $S\# = 0$ и $M\# = 0$ означают, что эти переменные не могут принимать значение 0. При оценке интерпретатором этих ограничений он уменьшает домены этих двух переменных, удаляя значение 0 из них. Затем рассматривается и запоминается ограничение alldifferent(Digits). Последнее алгебраическое ограничение выражает тот факт, что цифры, присвоенные буквам, должны быть такими, чтобы «SEND+MORE=MONEY» выполнялось при замене каждой буквы на соответствующую ей цифру.

Перечислим популярные языки ЛПО:

- **B-Prolog** (на базе **Prolog**, коммерческий) http://www.probp.com/
- **CHIP V5** (на базе **Prolog**, включает библиотеки C++ и C, коммерческий) http://www.cosytec.com/production_scheduling/chip/optimization_product_chip.htm
- **Ciao Prolog** (на базе **Prolog**, свободно распространяемое ПО: GPL/LGPL) http://clip.dia.fi.upm.es/Software/Ciao/
- **ECLiPSe** (на базе **Prolog**, open source) http://eclipse-clp.org/
- **SICStus** (на базе **Prolog**, коммерческий) http://www.sics.se/isl/sicstuswww/site/index.html
- **GNU Prolog** (свободно распространяемое ПО) http://www.gprolog.org/
- **Oz** http://www.dmoz.org/Computers/Programming/Languages/Oz/
- **YAP Prolog** http://www.dcc.fc.up.pt/~vsc/Yap/documentation.html
- **SWI Prolog** — свободная система **Prolog**, содержащая несколько библиотек для решения ограничений http://www.swi-prolog.org/
- **Claire** http://www.claire-language.com/cgi-bin/trac.cgi
- **Curry** (на базе **Haskell**, свободно распространяемое) http://www.curry-language.org/
- **HAL**: [147], [145]

Программирование в ограничениях реализуется в императивном программировании с помощью создания отдельных библиотек. Такими популярными библиотеками для программирования в ограничениях являются:
Choco (библиотека Java, свободно распространяемое ПО)
http://www.emn.fr/x-info/choco-solver/doku.php
Comet (язык C-стиля для программирования в ограничениях, локальный поиск, основанный на ограничениях, и математическое программирование, свободно распространяемые исполнимые файлы для академических целей) http://dynadec.com/
Disolver (библиотека C++, коммерческий)
Gecode (C++ library, свободное ПО: X11 style) http://www.gecode.org/
ILOG CP Optimizer (C++, Java, библиотеки .NET libraries, коммерческий)
http://www.ilog.com/products/cpoptimizer/
ILOG CP (библиотека C++, коммерческий) http://www.ilog.com/products/cp/
JaCoP(библиотека Java, open source) http://jacop.osolpro.com/
JOpt (библиотека Java, свободно распространяемое ПО) http://jopt.sourceforge.net/
Koalog Constraint Solver (библиотека Java, коммерческий)
http://linux.softpedia.com/get/Programming/Libraries/

Koalog-Constraint-Solver-8073.shtml
Minion (C++ program, GPL) http://minion.sourceforge.net/
Python-constraint (библиотека Python, GPL) http://labix.org/python-constraint
Cream (библиотека Java, свободно распространяемое ПО: LGPL)
http://bach.istc.kobe-u.ac.jp/cream/
Emma (библиотека Python, коммерческий) http://www.eveutilities.com/products/emma

Языки с поддержкой программирования в ограничениях

Common Lisp via Screamer (свободно распространяемое ПО, которое поддерживает поиск с возвратами и возможности CLP(R), CHiP)
http://www.cl-user.net/asp/d86x/sdataQkvrOGEiQe5uDQ3jQRpX8yBX8yBXnMq=
/sdataQu3F$sSHnB==

Bertrand — язык для построения систем программирования в ограничениях [259].

5.11. Сравнение с другими языками моделирования

Алгебраические языки моделирования, появившиеся в 1970-х годах, используемые для решения задач математического программирования (например, **GAMS** (*Brooke et al.*, 1988) [85] и **MGG** (*Simons*, 1987) [351]), были существенным продвижением вперед. Во-первых, они были разработаны для упрощения роли пользователя в решении задач математического программирования, обеспечивая близкий к математической записи синтаксис. Во-вторых, они имеют декларативную природу, описывая задачу, а не то, как решения могут быть найдены. Это снимает нагрузку с пользователя, и позволяет просто применять различные решатели для решения одной и той же задачи. Другие полезные языки моделирования были созданы в других областях, например, **ACE** (*Fuchs & Schwitter*, 1996) [178], — язык рассуждений о базах знаний, подобный английскому.

Успех языков моделирования показал, что подобный подход мог бы быть плодотворным для решения задач УО. Ранние попытки в этом направлении, язык **ALICE** (*Lauriere*, 1978) [258] имеет много аспектов, подобных алгебраическим языкам моделирования того времени, в том числе декларативную природу. Языки программирования в ограничениях постепенно расширяли диапазон типов решающих переменных. Например, **Eclipse** (*Gervet*, 1994) [186] поддерживает решающие переменные, элементы доменов которых являются множествами; аналогично **F** поддерживает функции, **ESRA** поддерживает отношения и функции, а **NP-Spec** поддерживает множества, перестановки, разбиения и целые функции. Усиление абстракции позволяет пользователю избежать дополнительных усилий при моделировании, передавая это все компилятору. **ESSENCE** сделал большой скачок в этом направлении, обеспечивая более широкий диапазон типов, чем это было в предшествующих языках и, что уникально, конструкторы типов, которые могут быть как угодно вложенными. Именно эти факторы делают **ESSENCE** скорее языком спецификаций, а не просто языком моделирования.

При решении сложных комбинаторных задач обычно используется двухшаговая процедура. На первом шаге разрабатывается концептуальная модель, описывающая, в чем состоит задача, без описания, как фактически ее решать. Второй шаг состоит в отображении концептуальной модели на проектную модель, которая уже может быть непосредственно решена. В идеале, одна и та же концептуальная модель может быть преобразована в различные проектные модели, позволяя тем самым специалистам по моделированию легко «подключать и работать» (plug and play) с различными решателями. Новый язык моделирования **Zinc**, разработанный в рамках этой методологии, описан в [268]. **Zinc** (*Marriott et al.*, 2006), [267]; (*de la Banda et al.*, 2006) [144] — это новый язык спецификаций, использующий, как и **ESSENCE**, вложенные конструкторы типов, но, кроме того, в отличие от **ESSENCE**, он имеет возможность определять предикаты. Кроме **ESSENCE** известны лишь три языка программирования в ограничениях с подобными возможностями: **ESRA**, **F** и **LOCALIZER** (*Michel & Van Hentenryck*, 2000) [281].

Другой подход к спецификации задачи, который использует более мощный и общий язык спецификаций, такой как **Z**, был исследован в работе (*Renker & Ahriz*, 2004) [315], создавших инструментальные средства схемы **Z** для поддержки общих глобальных ограничений и использовавших это для построения обширного каталога спецификаций [318]. Недостаток этого подхода состоит в том, что **Z** является чересчур общим для задач УО, так как позволяет спецификации задач, которые естественным образом к ЗУО не сводятся. К недостаткам спецификаций языка **Z** относится гораздо меньшая их естественность по сравнению со спецификациями **ESSENCE**.

Язык **Alloy** (*Jackson*, 2006) [219] позволяет избежать некоторых недостатков языка **Z** путем ограничивания его до логики первого порядка. **Alloy** дает естественный и выразительный способ спецификации задач в терминах отношений и атомов и отображает эти спецификации в эффективные модели выполнимости SAT.

6. Приложения программирования в ограничениях и УО

Программирование в ограничениях в настоящее время имеет широкий круг приложений. Такие компании, как American Express, BMW, Coors, Danone, eBay, France Telecom, General Electric, HP, JB Hunt, LL Bean, Mitsubishi Chemical, Nippon Steel, Orange, Porsche, QAD, Royal Bank of Scotland, Shell, Travelocity, US Postal Service, Visa,Wal-Mart, Xerox, Yves Rocher, Zurich Insurance используют программирование в ограничениях для оптимизации своих бизнес-процессов [325].

Рассмотрим главные области приложений, в которых была использована технология программирования в ограничениях и оказалась конкурентноспособной как по гибкости моделирования, так и по эффективности решения. В числе обзорных работ по приложениям программирования в ограничениях можно упомянуть [221], [348], [349], [377].

6.1. Задачи о назначениях.

Задачи о назначениях были одним из первых типов прикладных задач, которые были решены с помощь технологии ЛПО. Эти задачи обычно учитывают два типа ресурсов и ограничений, связывающих их, и пытаются приписать один ресурс первого типа одному ресурсу второго типа так, чтобы все ограничения удовлетворялись.

Одним из примером этой задачи о назначениях может служить задача размещения стоянок самолетов в аэропорту, где самолеты (ресурсы первого типа) должны парковаться на доступных стоянках (второй тип ресурсов) во время их пребывания в аэропорту [107], [108], [303], [348].

6.2. Назначение персонала

Задачи назначения персонала являются частным случаем задач о назначениях, в которых одним типом ресурсов являются люди, что делает эти задачи достаточно специальными для того, чтобы рассматривать их отдельно. В самом деле, изменяющиеся правила работы и всевозможные предписания порождают сложные ограничения, зависящие от времени. Помимо этого, пользовательские предпочтения часто ведут к переограниченным задачам, не имеющим решений, удовлетворяющих всем ограничениям. Другим важным аспектом является необходимость соблюдения баланса объема работы между разными людьми, что приводит к трудным задачам оптимизации. Различные приложения программирования в ограничениях в области назначения персонала описаны в работах [102], [103] (разработка сменных графиков медсестер), [118] (графики работы на телевидении Франции), [155], [246], [298].

6.3. Управление сетями

Система **LOCARIM** [347] была разработана компанией COSYTEC для France Telecom: начиная с архитектурного проекта здания, она предлагает прокладку кабеля телекоммуникационной сети здания. Система **PLANETS** (PLanning Activities on NETworkS) [124], разработанная Университетом Каталония (University of Catalonia) в Барселоне для испанской электрической компании, как инструмент перестройки структуры электрической сети, который позволяет строить календарный план технического обслуживания деятельности с помощью выделения участков сети без нарушения обслуживания потребителей.

В работе [36] описано использование языка **CHIP** в управлении водной системой. Планирование телекоммуникационной сети мобильной связи было осуществлено с помощью системы **POPULAR** [177], запрограммированной на **ECLiPSe**.

6.4. Календарное планирование

Наверное, наиболее успешной прикладной областью для логического программирования в ограничениях является календарное планирование [44], [101]. Для

заданного множества ресурсов с данными мощностями, множество видов деятельности с данной продолжительностью и потребностями в ресурсах, множество темпоральных ограничений между видами деятельности, задача «чистого» календарного планирования состоит в решении, когда выполнить каждую работу так, чтобы выполнялись темпоральные ограничения и ограничения по ресурсам.

Современные результаты исследований по сложности классов задач УО показывают, что такие задачи календарного планирования на самом деле легко разрешимы и поэтому имеются эффективные алгоритмы их решения [310].

Типичным примером приложения календарного планирования, основанного на УО, является **ATLAS** [346], осуществляющий календарное планирование производства гербицидов на заводе Monsanto в Антверпене.

Система **PLANE** [59] использована компанией Dassault Aviation для планирования производства военного самолета "Mirage 2000" и самолета "Falcon".

Система **MOSES** [28], [350] была разработана компанией COSYTEC для производителя питания для животных в Великобритании.

Система **FORWARDC** [191] представляет собой систему поддержки принятия решений (СППР) [13], реализованную на **CHIP**, и используется на трех нефтеперегонных заводах в Европе для решения задач календарного планирования, возникающих при привозе сырой нефти, ее обработке, смешивании и доставке.

Xerox использовал систему УО для календарного планирования различных работ на копировальных машинах [174].

Упреждающие (preemptive) задачи календарного планирования — это те задачи, в которых работы могут прерываться во времени. И для таких задач могут успешно применяться технологии логического программирования в ограничениях: так, **CLAIRE Scheduler** — это библиотека программ в ограничениях, позволяющая работать с подобными задачами, в том числе при наличии предпочтений [296], [297]. Подобные функции выполняет и **ILOG Scheduler** [217].

Если сравнивать задачи календарного планирования по методам решения, то они решаются с помощью императивных языков (например, C/C++), неспециализированных языков ЛПО (например, **CHIP**), и специальных средств программирования в ограничениях. Работа [41] показала, что подход ЛПО превосходит другие по многим параметрам: время разработки, вершины, просмотренные в дереве поиска, по числу построенных допустимых решений, даже по эффективности. Согласно этому исследованию, главной причиной успеха языков ЛПО при решении этих задач состоит в использовании распространения ограничений, которое помогает уменьшить как время разработки, так и время выполнения. Выяснилось, что для некоторых классов задач календарного планирования не существует метода, лучшего всех остальных методов, а лучше применять сочетание нескольких методов. В связи с этим были предложены гибридные алгоритмы, использующие как языки ЛПО, так и другие методы. Так, сочетание ЛПО и целочисленного программирования было использовано для решения задач календарного планирования с многими подъемниками [321].

Отметим следующие системы планирования.

TAP-AI [52] — активная система планирования, предназначенная для назначения экипажей по авиалиниям SAS. Она предназначена для ежедневного управления деятельностью (операциями).

OPTISERVICE [118] — программный пакет для назначения персонала для всех заграничных теле- и радиостанций сети RFO. Он назначает команды квалифицированных журналистов и технических работников по передачам, учитывая ограничения по времени и условиям оплаты.

Система **MOSAR** (http://www.cosytec.com/constraint_programming/cases_studies/administration.htm), разработанная компаниями Cisi и COSYTEC для министерства юстиции Франции, назначает охранников тюрем по 200 тюрьмам Франции по чередующимся сменам.

6.5. Транспортные проблемы.

Многие транспортные проблемы были решены с использованием технологии ограничений. Эти задачи часто очень сложны в связи с их большими размерами, числу и множеству видов ограничений, а также из-за наличия сложных ограничений, накладываемых на возможные маршруты. Кроме того, эти задачи часто требуют решения подзадачи назначения персонала, причем при выполнении ряда сложных условий. Рассмотрим соответствующие приложения.

Система **COBRA** [350] разрабатывает диаграммы рабочих планов машинистов поездов компании North Western Trains в Великобритании.

Проект **DAYSY Esprit** (пакет **SAS-Pilot**) [52] осуществляет переназначение летных экипажей по полетам. Эта же задача решается с помощью другой системы [164], сочетающей методы программирования в ограничениях и исследования операций.

Система, описанная в [269], использует библиотеки ограничений ILOG для разработки и оптимизации поездных оперативных планов для грузовой железнодорожной компании.

Для задач планирования транспорта (как и для задач календарного планирования), в ряде случаев наилучшим методом оказалась комбинация нескольких методов. Например, УО и локальный поиск были совместно использованы при решении задач маршрутизации транспортных средств в [340], а в работе [305] такие же задачи решались с помощью комбинации методов УО и методов исследования операций.

Маршрутизация транспортных средств (Vehicle Routing) — это задача построения маршрутов для транспортных средств для перевозок продукции пользователям с минимальной стоимостью.

6.6. Системы управления электромеханическими системами

Технология УО используется для разработки ПО для управления электромеханическими системами с конечным числом входов и выходов и внутренних состояний.

Системы управления требуются для таких приложений, как лифты, копировальные машины, сборочные линии и электростанции. В качестве примеров такой программной системы можно привести системы верификации: **SVE** (system verification environment) [163], а также система верификации параллельных систем [276], использующая сети Петри и программирование в ограничениях (язык программирования **2LP**).

6.7. Электронные таблицы с ограничениями

Электронная таблица в настоящее время — одно из наиболее распространённых программных средств поддержки принятия решений. Однако, обычные электронные таблицы имеют следующие два ограничения: они вычисляют лишь в фиксированном направлении, (от входных ячеек к ячейкам-результатам), при этом результаты могут быть определены после задания значений входных ячеек. С другой стороны, ограничения позволяют переменным воздействовать друг на друга во всех направлениях (как в ограничении $X \leq Y$), и работать естественным образом с частичной информацией Идея применения технологии ЛПО для преодоления этих ограничений существующих электронных таблиц была использована при создании нескольких программных систем.

Система краткосрочного планирования (The Short Term Planning (STP)) для компании Renault [105] решает задачу транспортировки автомобилей заказчикам с учетом множества ограничений. Были разработаны электронные таблицы на основе ограничений для помощи пользователю в решении этой задачи. Во время процесса планирования, величины в электронных таблицах — не точные числа, а интервалы [min, max]. Ограничения приписаны к различным ячейкам таблицы и используются для распространения последствий изменений, наложенных пользователем, который может либо ввести конкретное значение в ячейку, или прямо ограничить диапазон возможных значений путем сужения интервала.

Другим перспективным приложением электронных таблиц с ограничениями является финансовое планирование [215]: электронные таблицы позволяют специалисту по планированию финансов использовать потенциальные инвестиции. Финансовые приложения часто содержат нелинейные ограничения, которые могут решаться с помощью методов интервального анализа. Планирование бюджетов районов Москвы и последующий контроль за этими бюджетами были осуществлены с помощью электронных таблиц на основе ограничений, где метод решения ограничений был встроен в язык **ECLiPSe** [342].

6.8. Интерактивное решение задач

Технология ЛПО, в особенности распространение ограничений, может быть использована при организации участия пользователя в разработке решений. Фактически на каждом шаге последствия конкретного выбора явно демонстрируются пользователю с помощью механизма распространения ограничений. Пользователь в этом случае либо отказывается от этого решения, либо использует распростра-

ненную информацию для выработки следующего выбора или разрешает системе сделать этот выбор автоматически.

Например, системы ЛПО такого типа были использованы для решения задач составления расписаний в Banque Bruxelles Lambert [153] и медицинского факультета Университета им. Гумбольта (Humbolt University) [193].

Использование систем ЛПО описано также в работах [26], [63], [182], [187].

6.9. Графические интерфейсы пользвателя

Одной из обширных и старейших областей приложения для технологии УО является разработка графических интерфейсов. Роль УО состоит здесь в сохранении графических объектов в правильном отношении друг к другу после обновления графического окна (обычно с помощью мышки). Здесь цель — относительно быстрое нахождение допустимого решения. Коммерческая библиотека программирования в ограничениях для построения интерактивных графических интерфейсов пользователя предложена в [80]. Программирование в ограничениях было использовано при создании алгоритмов построения графов [360] и при создании средств визуализации и анимации [359]. Графический интерфейс, использующий технологию УО, был использован для решения задачи сборки в профессиональных системах CAD/CAM [339]. Разработан язык **EaCL** [365] для содействия пользователям в построении задач УО.

6.10. Переограниченные задачи

Многие прикладные задачи являются переограниченными: пользователь вводит настолько много ограничений, что становится невозможным удовлетворить их все одновременно. В этих случаях нужно вначале показать, что решений нет (выяснение этого может занять много времени), а затем решить, какие из ограничений ослабить, чтобы задача стала разрешимой. Для решения этой второй задачи предлагается попросить конечного пользователя найти некоторые решения, так, как это делается в интерактивном решении задач. В результате получается иерархия ограничений (где вышестоящие ограничения являются более важными) [79], или в более общем случае используются мягкие ограничения [172], [154], [328], [334], [68]. Типичными переограниченными задачами являются задачи составления расписаний, в которых ограничения из разных источников (например, доступность аудиторий и предпочтения преподавателей). Современные системы составления расписаний используют как жесткие, так и мягкие ограничения для моделирования и решения задач составления расписаний, как в University of Siegen, Germany [53], а другая система — составления расписаний по информатике в Техническом университете Флориды использует два этапа УО (первый этап — назначить факультет курсам, а второй — приписать временные интервалы курсам лекций) [72]. Еще одна система, решающая переограниченные задачи с помощью иерархической системой УО, используется в настоящее время в больнице Neuwied для решения задачи календарного планирования работы медсестер [280].

6.11. Другие области приложений

Сложная задача прогноза структуры белка (одна из наиболее важных задач вычислительной биологии) решается с помощью технологии ЛПО [40]. Эта задача состоит в нахождении структуры белка с минимальной энергией. Оказалось, что использование ограничений помогает снизить объем поиска для решения этой NP-полной задачи.

Технология ограничений была также использована Sony в нескольких приложениях, связанных с музыкой [299].

7. Заключение.

УО и программирование в ограничениях — интересные и многообещающие технологии искусственного интеллекта, позволяющие в сочетании с методами исследования операций решать сложные комбинаторные задачи.

Автор надеется, что данный обзор, посвященный чрезвычайно интересному и перспективному направлению искусственного интеллекта, будет интересен и полезен специалистам в области искусственного интеллекта, исследования операций, дискретной математики, систем поддержки принятия решений.

Список цитируемых источников

1. Братко И. Алгоритмы искусственного интеллекта на языке Prolog, 3-е издание / Пер. с англ. М.: Издательский дом «Вильямс», 2004. 640 с.
2. Булатов А.А. Полиномиальность мальцевских задач CSP // Алгебра и логика. 2006. Т. 45. С. 655–686.
3. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982. 416 с.
4. Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И. Лекции по теории графов. Изд. 2-е, испр. М.: Книжный дом «ЛИБРОКОМ», 2009. 392 с.
5. NeMo+: Объектно-ориентированная система программирования в ограничениях на основе недоопределенных моделей / Г.Б. Загорулько [и др.] // Тр. шестой нац. конф. по искусственному интеллекту (КИИ'98). Пущино. 1998. С. 524–530.
6. Обстановка для программирования в ограничениях на основе недоопределенных моделей NeMo+ (язык, архитектура, интерфейс) / Г.Б. Загорулько [и др.] // Научно-техн. отчет N 7 / Российский НИИ искусственного интеллекта, Институт систем информатики им. А. П. Ершова СО РАН. Москва, Новосибирск, 1998. 107 с.
7. Программирование в ограничениях и недоопределенные модели А.С. Нариньяни [и др.] // Информационные технологии. 1998. N 7. С. 13–22.
8. Технология решения задач в объектно-ориентированной среде NeMo+ / А.А. Кузнецов [и др.] // Пятая национальная конференция с международным участием «Искусственный интеллект — 96». Сборник научных трудов в трех томах. Том III. Казань, 1996. С. 408–414.
9. Нариньяни А.С., Напреенко В.Г., Смирнов Е.П. Компьютерная модель экономики региона на основе интеллектуальной технологии нового поколения. Концепция,

технология и инструменты формирования управленческих решений в социально-экономической и технологической сферах субъектов Российской Федерации. Издание рабочей группы по стратегии регионального развития и безопасности Государственного Совета Российской Федерации. Москва, 2005.

10. Петров Е.С. Опыт интеграции логического программирования и программирования в ограничениях // Программирование. 1998. N 3. С.40–49.
11. Петров Е.С., Яхно Т.М. Недоопределенные модели и логическое программирование: реализация ограничений // Программирование. 2001, № 2. С.60-67.
12. Рассел С., Норвиг П. Искусственный интеллект: современный подход. 2е изд. / Пер. с англ. М.: Вильямс, 2006. 1408 с. имеется копия: http://rriai.org.ru/zadachi-udovletvoreniya-ogranicheniy.html (дата обращения 07.08.2010).
13. Сараев А.Д., Щербина О.А. Системный анализ и современные информационные технологии // Труды Крымской академии наук. Симферополь: СОНАТ. 2006. С. 47-59.
14. Семенов А.Л. Методы распространения ограничений: основные концепции. Труды конференции ИМРО-03. Новосибирск, 2003. С. 19-31.
15. Сидоров В.А. Программирование в ограничениях с черными ящиками. Новосибирск, 2003. 39 с. (Препр. / ЗАО Ледас; N2).
16. Сидоров В.А. Методы и средства программирования в ограничениях для систем автоматизации проектирования: дисс... канд. физ.-мат. наук (05.13.11 — математическое моделирование, численные методы и комплексы программ). Новосибирск, Ин-т систем информатики им. А.П. Ершова СО РАН, 2007. 152 с.
17. Скворцов Е.С. Об эффективных алгоритмах для задачи CSP и их программной реализации. Автореферат дисс... канд. физ.-мат. наук (05.13.18 — математическое моделирование, численные методы и комплексы программ). Екатеринбург, Уральский государственный университет им. А.М. Горького, 2008. 24 с.
18. Телерман В.В., Ушаков Д.М. Удовлетворение ограничений в задачах математического программирования // Вычислительные технологии. 1998. Том 3, № 2. С. 45-53.
19. Ушаков Д.М., Телерман В.В. Системы программирования в ограничениях (обзор) // Системная информатика: Сб. науч. тр. Новосибирск: Наука, 2000. Вып.7: Проблемы теории и методологии создания параллельных и распределенных систем. С. 275-310.
20. Швецов И.Е., Нестеренко Т.В. Проектирование статических и динамических систем средствами программирования в ограничениях. // Труды международной конференции «Проблемы управления и моделирования в сложных системах». Самара: Самарский научный центр РАН, 1999. С. 262-267.
21. Щербина О.А. Элиминационные алгоритмы декомпозиции задач дискретной оптимизации // Таврический вестник информатики и математики. 2006. №2. С. 28-41.
22. Щербина О.А. Локальные элиминационные алгоритмы для задач удовлетворения ограничений // Таврический вестник информатики и математики. 2007. Вып. 1. С. 24-39.
23. Щербина О.А. Древовидная декомпозиция и задачи дискретной оптимизации (обзор) // Кибернетика и системный анализ. 2007. №4. С.102-118.

24. Щербина О.А. Локальные элиминационные алгоритмы для решения некоторых задач искусственного интеллекта // Труды 11 Национальной конференции по искусственному интеллекту (Дубна, 28 сентября — 3 октября 2008 г.). М.: ЛЕНАНД, 2008. Т. 2. С. 244-252. http:www.raai.org/cai-08/files/cai-08_paper_189.doc (дата обращения 07.08.2010)
25. Models and solution techniques for the frequency assignment problems / K.I. Aardal [et al.] // Annals of Operations Research. 2007. Vol. 153. P. 79-129.
26. Abdennadher S., Schlenker H. INTERDIP — an interactive constraint based nurse scheduler // Proc. PACLP99, 1999. http://www.pms.ifi.lmu.de/publikationen/PMS-FB/PMS-FB-1999-2/PMS-FB-1999-2.pdf (дата обращения 07.08.2010)
27. Aggoun A., Beldiceanu N. Extending CHIP in order to solve complex scheduling and placement problems // Mathl. Comput. Modelling. 1993. V. 17. P. 57–73.
28. Aggoun A., Gloner Y., Simonis H. Global constraints for scheduling in CHIP. Invited Industrial Presentation // JFPLC 99, 1999.
29. Allen J.F. Maintaining knowledge about temporal intervals // Comm. ACM. 1983. 26. P. 832–843.
30. Allen J.F. Toward a general theory of action and time // Artificial Intelligence. 1984. V.23(2). P.123–154.
31. Allen J.F. Natural Language Understanding. 2nd edition. Benjamin Cummings, 1994. 654 p.
32. Andersson K., Hjerpe T. Modeling Constraint Problems in CML // The Third International Conference on the Practical Application of Constraint Technology (PACT'98), London, 1998. P. 295–312.
33. Appel K., Haken W. Every planar map is four colorable: Part I: Discharging // Illinois J. Math. 1977. V. 21. P. 429-490.
34. Apt K. R. Principles of Constraint Programming. New York: Cambridge University Press, 2003. 407 p.
35. Apt K. R. The essence of constraint propagation // Theoretical Computer Science, 1999. 221(1-2). P. 179—210.
36. New Trends in Constraints / K.R. Apt [et al.], eds. Joint ERCIM/Compulog Net Workshop, October 1999, Selected papers, vol. 1865 of Lecture Notes in Computer Science, Paphos, Cyprus. Springer-Verlag, 2000. 343 p.
37. Apt K.R., Wallace M. Constraint Logic Programming using Eclipse. New York: Cambridge University Press, 2006. 329 p.
38. Arnborg S. Efficient algorithms for combinatorial problems on graphs with bounded decomposability — a survey // BIT. 1985. V. 25. P. 2-23.
39. Bacchus F., van Beek P. On the conversion between non-binary and binary constraint satisfaction problems // Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98). Madison: AAAI Press, 1998. P. 311-318.
40. Backofen R. The protein structure prediction problem: A constraint optimization approach using a new lower bound // Constraints. 2001. V. 6. P. 223-255.
41. Multi-criteria comparison between algorithic, constraint logic and specific constraint programming on a real scheduling problem / A. Bachelu [et al.] // Proc. PACT97, 1997.

42. Baker A.B. Intelligent backtracking on constraint satisfaction problems: Experimental theoretical results. Ph.D. Thesis, Graduate School of the University of Oregon, 1995.
43. Baptiste P., Le Pape C., Peridy L. Global constraints for partial CSPs: a case- study of resource and due date constraints // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520, 1998.
44. Baptiste P., Le Pape C., Nuijten W. Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Dordrecht: Kluwer Academic Publishers, 2001. 198 p.
45. Barnier N., Brisset P. Graph coloring for air traffic flow management // Annals of Operations Research. 2004. V.130. P. 163-178.
46. Bartak R. Theory and practice of constraint propagation // Proceedings of the Third Workshop on Constraint Programming for Decision and Control (CPDC-0I). Poland, Gliwice, 2001. P. 7-14.
47. 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.
48. Bartak R.Theory and Practice of Constraint Propagation // Proceedings of the 3rd Workshop on Constraint Programming for Decision and Control (CPDC2001), Poland, Gliwice: Wydavnictvo Pracovni Komputerowej, 2001. P. 7-14.
49. Bartak R. Constraint-based Scheduling: An Introduction for Newcomers. // Intelligent Manufacturing Systems 2003. A Proceedings volume from the 7th IFAC Symposium / L. Monostori, B. Kadar, G. Morel (eds.) (6-8 April, 2003, Budapest, Hungary). Elsevier Science, 2003. P. 69-74.
50. Bartak R. Modelling soft constraints: A survey // Neural Network World. 2002. Vol. 12, Number 5. P. 421-431,
51. Bartak R. Expert Systems Based on Constraints (in Czech, English abstract), Doctoral Dissertation. Prague, Charles University, April 1997.
52. Baues G., Kay P., Charlier P. Constraint based resource allocation for airline crew management // Proc. ATTIS'94, 1994.
53. Priority-driven constraints used for scheduling at universities / M. Baumgart [et al.] // Proc. PACT97, 1997.
54. Bayardo R. J., Schrag R. C. Using CSP lookback techniques to solve real-world sat instances // Proc. of the AAAI National Conference. 1997. P. 203–208.
55. On the desirability of acyclic database schemes / C. Beeri [et al.] // Journal of the Association for Computing Machinery. 1983. V. 30(3). P. 479-513.
56. Beldiceanu N., Contejean E. Introducing global constraints in CHIP // Journal of Mathematical and Computer Modeling. 1994. V. 20. P. 97-123.
57. Partial search strategy in CHIP / N. Beldiceanu [et al.] // Proc. 2nd Int. Conf. on Meta-Heuristics. France, Sophia-Antipolis, 1997.
58. Beldiceanu N. Global constraints as graph properties on structured network of elementary constraints of the same type. Technical Report T2000/01, SICS, 2000.
59. Bellone J., Chamard A., Pradelles C. Plane — an evolutive planning system for aircraft production // Proc. 1st Int. Conf. on Practical Applications of Prolog (PAP92), 1992.
60. Belussi A., Bertino E., Catania B. Manipulating spatial data in constraint databases // Proc. 5th Symp. on Spatial Databases. Springer-Verlag, LNCS 1262, 1997.

61. Benhamou F., Older W. Applying interval arithmetic to real, integer and boolean constraints // Journal of Logic Programming. 1997. V. 32(1). P. 1-24.
62. Benhamou F., Van Hentenryck P. (eds.). Special Issue on Interval Constraints // Constraints: An International Journal. 1997. V.2, N.2.
63. Berger R. Constraint-based gate allocation for airports // Proc. ILOG User- group meeting, 1995.
64. Bertele U., Brioschi F. Nonserial Dynamic Programming. New York: Academic Press, 1972. 235 p.
65. An optimal coarse-grained arc consistency algorithm / C. Bessière [et al.] // Artificial Intelligence. 2005. V. 165. P. 165–185.
66. Bessière C. Constraint propagation. Chapter 4 in Rossi F., van Beek P., Walsh T. (Eds.) Handbook of Constraint Programming. Elsevier, 2006. P. 29-83
67. Biggs N.L., Lloyd E.K., Wilson R.J. Graph Theory 1736-1936. Oxford: Oxford University Press, 1976. 239 p.
68. Bistarelli S., Montanari U., Rossi F. Semiring-based constraint solving and optimization // Journal of the ACM. 1997. 44. P. 201-236.
69. Bistarelli S., Montanari U., Rossi F. Semiring-based Constraint Logic Programming // Proc. IJCAI97. Morgan Kaufman, 1997.
70. Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison / S. Bistarelli [et al.] // Constraints. 1999. V. 4. P. 199–240.
71. Bitner J. R., Reingold E. M. Backtrack programming techniques // Communications of the Association for Computing Machinery. 1975. 18(11). P. 651-656.
72. Blanco J.J., Khatib L. Course scheduling as a constraint satisfaction problem // Proc. PACT98, 1998.
73. Bliek C., Neveu B., Trombettoni G. Using graph decomposition techniques for solving continuous CSPs // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98), Springer-Verlag, LNCS 1520, 1998. P. 102-116.
74. Blum A., Furst M. Fast planning through planning graph analysis // Artificial Intelligence. 1997. V. 90. P.281–300.
75. Bodirsky M., Neš etřil J. Constraint satisfaction with countable homogeneous templates // Journal of Logic and Computation. 2006. V. 16. P. 359–373.
76. Böhler E., Hemaspaandra E., Reith S., and Vollmer H. The complexity of Boolean constraint isomorphism // STACS 2004 (Montpellier, 2004), LNCS 2996. Berlin: Springer, 2004. P. 164-175.
77. Quantified constraints: Algorithms and complexity / F. Börner [et al.] // Computer Science Logic (Vienna, 2003), Lecture Notes in Comput. Sci. 2803, Berlin: Springer, 2003. P. 58-70.
78. Borning A. The programming language aspects of thinglab, a constraint oriented simulation laboratory // ACM Transaction on Programming Languages and Systems. 1981. V. 3(4). P. 353–387.
79. Borning A., Maher M., Martindale A., Wilson M. Constraint hierarchies and logic programming // Proc. 6th International Conference on Logic Programming / M. Martelli, G. Levi (eds.). cambridge: MIT Press, 1989. P. 149-164.

80. Borning A., Freeman-Benson B.N. The OTI constraint solver: a constraint library for constructing graphical user interfaces // Proceedings 1st Int. Conference on Principles and Practice of Constraint Programming (CP95). Springer-Verlag, LNCS 976, 1995.
81. Brailsford S.C., Potts C.N., Smith B.M. Constraint satisfaction problems: Algorithms and applications // European Journal of Operational Research. 1999. V. 119. P. 557-581.
82. Brelaz D. New methods to color the vertices of a graph // Communications of the Association for Computing Machinery. 1979. 22(4). P. 251-256.
83. Brodsky A. Constraint databases: Promising technology or just intellectual exercise? // Constraints: An International Journal. 1997. V. 2. P. 35-44.
84. Brodsky A., Jaffar J., Maher M. Toward practical query evaluation for constraint databases // Constraints: An International Journal, 1997. V. 2. P. 279-304.
85. Brooke A., Kendrick D., Meeraus A. GAMS: A Users' Guide. Danvers: The Scientific Press, 1988.
86. Bruynooghe M. Solving combinatorial search problems by intelligent backtracking // Inform. Process. Lett. 1981. V. 12. P. 36–39.
87. Bruynooghe M. Graph Coloring and Constraint Satisfaction. Technical Report CW- 44. Kathioleke Universiteit Leuven: Department Computerwtenschappen, 1985.
88. Bruynooghe M. Enhancing a search algorithm to perform intelligent backtracking // Theory Pract. Log. Program. 2004. V. 4. P. 371-380.
89. Bulatov A., Dalmau V. Mal'tsev constraints are tractable // SIAM J. on Computing. 2006. V. 36(1). P. 16-27.
90. Bulatov A., Krokhin A., Jeavons P. Classifying the complexity of constraints using finite algebras // SIAM Journal on Computing. 2005. V. 34(3). P. 720-742.
91. Bulatov A., Krokhin A., Jeavons P. The complexity of maximal constraint languages // Proceedings 33rd ACM Symposium on Theory of Computing (STOC'01), 2001. P. 667–674.
92. Bulatov A. Mal'tsev Constraints are Tractable. Technical Report PRG-02-05, Computing Laboratory, Oxford University, 2002.
93. Bulatov A., Jeavons P.G. Tractable Constraints Closed under a Binary Operation. Technical Report PRG-TR-12-00. Oxford, University of Oxford, Computing Laboratory, 2002.
94. Bulatov A. Tractable conservative constraint satisfaction problems // Proceedings of the 18th Annual IEEE Simposium on Logic in Computer Science, (LICS'03), 2003. P. 321–330.
95. Bulatov A., Dalmau V. Towards a dichotomy theorem for the counting constraint satisfaction problem // Proceedings 44rd IEEE Symposium on Foundations of Computer Science, FOCS'03, Boston, 2003, P. 562–571.
96. Bulatov A., Dalmau V. A Simple Algorithm for Mal'tsev Constraints // SIAM J. Comput. 2006. V. 36. P. 16-27.
97. Buscemi M.G., Montanari U. A survey of constraint-based programming paradigms // Computer Science Review. 2008. V. 2. P. 137-141.
98. Cabon B., de Givry S., Verfaille G. Anytime lower bounds for constraint violation minimization problems // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520, 1998.

99. Carlsson M., Widen J. SICStus Prolog User's Manual. Technical report, Swedish Institute of Computer Science (SICS), 1999. on-line version at http://www.sics.se/sicstus/ (дата обращения 07.08.2010).
100. Carro M., Hermenegildo M. Some design issues in the visualization of constraint program execution // Proc. Joint Conference on Declarative Programming (AGP'98), 1998.
101. Caseau Y., Laburthe F. Improved CLP scheduling with task intervals // Proceedings of the Eleventh International Conference on Logic Programming. Santa Margherita Ligure, 1994. P. 369–383.
102. Chan P., Heus K., Veil G. Nurse scheduling with global constraints in CHIP: Gymnaste // Proc. PACT98, 1998.
103. Chan P. La planification du personnel: acteurs, actions et termes multiples pour une planification operationelle des personnes. PhD thesis, Universite Joseph Fourier, Grenoble 1. TIMC-IMAG, October 2002.
104. Cheeseman P., Kanefsky B., Taylor W. Where the really hard problems are // Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91). Morgan Kaufmann, 1991. P. 331-337.
105. Chew T., David J.-M. A constraint-based spreadsheet for cooperative production planning // Proc. AAAI SIGMAN workshop on knowledge-based production planning, scheduling and control, 1992.
106. Chmeiss A., Jegou P. Path-consistency: when space misses time // Proceedings of the 13th National Conference on Artificial Intelligence. Portland, 1996. P. 196–201.
107. Chow K.P., Perrett M. Airport counter allocation using constraint logic programming // Proc. Third International Conference on Constraint Technology (PACT97). London, 1997.
108. Stand allocation with constraint technologies at Chek Lap Kok international airport / A.H.W. Chun [et al.] // Proc. The First International Conference and Exhibition on The Practical Application of Constraint Technologies and Logic Programming PACLP99 (19th -21 April 1999), Commonwealth Conference and Events Centre, London, 1999. P. 131-136.
109. Codognet P., Diaz D. Compiling constraints in clp(fd) // Journal of Logic Programming. 1996. V. 27. P. 185-226.
110. Cohen D.A., Gyssens M., Jeavons P.D. Derivation of constraints and database relations // Proceedings 2nd Int. Conference on Principles and Practice of Constraint Programming (CP96). Springer-Verlag, LNCS 1118, 1996. P. 134-148.
111. Cohen D.A., Jeavons P., Koubarakis M. Tractable disjunctive constraints. // Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97). Springer-Verlag, LNCS 1330, 1997. P. 478-490.
112. Cohen D.A., Jeavons P.G., Jonsson P., Koubarakis M. Building tractable disjunctive constraints // Journal of the ACM. 2000. V. 47. P. 826–853.
113. Cohen D., Jeavons P., and Gault R. New tractable constraint classes from old // Exploring Artificial intelligence in the New Millennium / G. Lakemeyer and B. Nebel (eds.). San Francisco: Morgan Kaufmann Publishers, 2003. P. 331-354.
114. Cohen D.A., Jeavons P.D. The complexity of constraint languages. // Handbook of Constraint Programming, chapter 8. / F. Rossi, P. van Beek, and T. Walsh (eds.). Elsevier, 2006.

115. Cohen D.A., Jeavons P.D., Gyssens M. A unified theory of structural tractability for constraint satisfaction problems // Journal of Computer and System Sciences. 2008. V. 74. P. 721-743.
http://www.comlab.ox.ac.uk/activities/constraints/publications/JCSSspreadcut.pdf (дата обращения 07.08.2010)

116. Cohen J. Logic Programming and Constraint Logic Programming // Computer Science Handbook / editor-in-chief A.B. Tucker. 2nd ed. Chapman & Hall/CRC, 2004. Chapter 93.

117. Collavizza H., Delobel F., Rueher M. A note on partial consistency over continuous domains // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520, 1998. P. 147-161.

118. Collignon C. Gestion optimisee de ressources humaines pour l'audiovisuel // Proc. CHIP users' club, 1996.

119. Colmerauer A. Opening the Prolog-III universe // BYTE magazine. 1987. V. 12. P.177-182.

120. Colmerauer A. An introduction to Prolog-III // Communication of the ACM. 1990. V. 33. P. 69-90.

121. Cooper M.C., Cohen D.A., Jeavons P.G. Characterising tractable constraints // Artificial Intelligence. 1994. V. 65. P. 347–361.

122. Cooper M.C., Jeavons P.D., Salamon A.Z. Hybrid tractable CSPs which generalize tree structure // Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008) / M. Ghallab [et al.] eds, (July 21—25, Patras, Greece). Vol. 178 of Frontiers in Artificial Intelligence and Applications. IOS Press, 2008. P. 530—534 http://www.gaon.net/andras/academic/cjs-ecai2008.pdf (дата обращения 07.08.2010)

123. COSYTEC, CHIP C library. COSYTEC SA, Parc Club Orsay Universite, 1995.

124. Constraint-based Maintenance Scheduling on an Electric Power-Distribution Network / T. Creemers [et al.] // Proc. of the 3rd International Conference and Exhibition on Practical Applications of Prolog. Paris: Alinmead Software Ltd. April, 1995. P. 135-144.

125. Creignou N., Khanna S., Sudan M. Complexity Classification of Boolean Constraint Satisfaction Problems, volume 7 of SIAM Monographs on Discrete Mathematics and Applications. Philadelphia: SIAM, 2001.

126. Dalmau V. A new tractable class of constraint satisfaction problems // Annals of Mathematics and Artificial Intelligence. 2005. V. 44(1-2). P. 61-85.

127. Constraint logic programming and integer programming approaches and their collaboration in solving an assignment scheduling problem / K. Darby-Dowman [et al.] // Constraints: An International Journal. 1997. V. 1. P. 245–264.

128. Davis M., Putnam H. A computing procedure for quantification theory. // Journal of the ACM. 1960. V. 7. P.201-215.

129. de Boer F., Gabbrielli M., Meo M.C. Semantics and expressive power of a timed concurrent constraint language // Proc. Third Int'l Conf. on Principles and Practice of Constraint Programming (CP 97) / G. Smolka (ed.). LNCS 1330. Berlin: Springer-Verlag, 1997. P. 47—61.

130. de Kleer J. A comparison of ATMS and CSP techniques // Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89), Vol. 1. Detroit: Morgan Kaufmann, 1989. P. 290-296.
131. Dechter R. A Constraint-Network Approach to Truth-Maintenance. Technical Report R-80. Cognitive Systems Laboratory, University of California, Computer Science Department, 1987.
132. Dechter R., Dechter A. Belief maintenance in dynamic constraint networks // Proceedings of The 7th National Conference on Artificial Intelligence. St Paul, Minnesota, 1988. P. 37–42.
133. Dechter R. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition // Artificial Intelligence. 1990. V. 41. P. 273-312.
134. Dechter R. On the expressiveness of networks with hidden variables // Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90). Boston: MIT Press, 1990. P. 379-385,.
135. Dechter R. Constraint networks // Encyclopedia of Artificial Intelligence (2nd edition) / S. Shapiro (ed.). New York: Wiley and Sons, 1992. P. 276-285
136. Dechter R. From local to global consistency // Artificial Intelligence. 1992. V. 55. P. 87-107.
137. Dechter R. Bucket elimination: a unifying framework for processing hard and soft constraints // Constraints. 1997. V. 2. P. 51-55.
138. Dechter R., Frost D. Backtracking algorithms for constraint satisfaction problems. Tech.rep., University of California, Department of Information and Computer Science, 1999.http://www.ics.uci.edu/~csp/r56-backtracking.pdf (дата обращения 07.08.2010)
139. Dechter R., Meiri I. Experimental evaluation of preprocessing techniques in constraint satisfaction problems // Artificial Intelligence. 1994. V. 68. P. 211-241.
140. Dechter R., Pearl J. Network-based heuristics for constraint-satisfaction problems // Artificial Intelligence. 1987. 34(1). P. 1-38.
141. Dechter R., Pearl J. Tree clustering for constraint networks // Artificial Intelligence. 1989. 38(3). P. 353-366.
142. Dechter R. Constraint processing. San Francisco: Morgan Kaufmann, 2003. 481 p.
143. De Givry S., Verfaille G., Schiex T. Bounding the optimum of constraint optimization problems // Proc. CP97 / G. Smolka (ed.). Springer-Verlag, LNCS 1330, 1997. P. 405-419.
144. The modelling language Zinc / M.G. de la Banda [et al.] // Principles and Practice of Constraint Programming — CP 2006. LNCS 4204. Springer, 2006. P. 700–705.
145. To the Gates of HAL: a HAL tutorial / G.M. de la Banda // 6th International Conference on Functional and Logic Programming (FLOPS), Japan, 2002.
146. Denecke K., Wismath S.L. Universal Algebra and Applications in Theoretical Computer Science. Chapman and Hall/CRC Press, 2002. 383 p.
147. An overview of HAL / B. Demoen [et al.] // International Conference on Principles and Practice of Constraint Programming, CP99. Alexandria, 1999. P. 174-188.
148. Deville Y., Jansenn M., Van Hentenryck P. Consistency techniques in ordinary differential equations // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98), LNCS 1520. Springer-Verlag, 1998. P. 162-176.

149. The constraint logic programming language CHIP / M. Dincbas [et al.] // Proc. International Conference on Fifth Generation Computer Systems (FGCS). Berlin Heidelberg New York: Springer, 1988. P. 693-702.

150. Applications of CHIP to industrial and engineering problems / M. Dincbas [et al.] // First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Tullahoma, Tennessee, 1988. P.885-892.

151. Dincbas M., Simonis H. Apache — a constraint based, automated stand allocation system // Proc. of Advanced Software technology in Air Transport (ASTAIR'91). London, UK: Royal Aeronautical Society, 1991. P. 267-282.

152. Doyle J. A truth maintenance system // Artificial Intelligence. 1979. 12(3). P. 231-272.

153. Dresse A. A constraint programming library dedicated to timetabling // Proc. ILOG User-group meeting, 1995.

154. Dubois D., Fargier H., Prade H. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction // Proc. IEEE International Conference on Fuzzy Systems. IEEE, 1993. P. 1131-1136.

155. Dubos A., Du Jeu A. Application EPPER planification des ageents roulaants // Proc. CHIP users' club, 1996.

156. Dunkin N., Allen S.M. Frequency assignment problems: Representations and solutions. Technical Report CSD-TR-97-14. Royal Holloway: University of London, 1997.

157. Towards High Order Constraint Representations for the Frequency Assignment Problem / N.W. Dunkin [et al.]. Technical Report CSD-TR-98-05, Royal Holloway, University of London, June 1998. ftp://ftp.cs.rhul.ac.uk/pub/constraints/CSD-TR-98-05.ps (дата обращения 07.08.2010)

158. The ECLiPSe Library Manual Release 5.0. http://www.win.tue.nl/~setalle/lp/eclipse-doc/libman/libman.html,2000 (дата обращения 07.08.2010)

159. Even S. Graph Algorithms. Potomac: Computer Science Press, 1979.

160. Fargier H., Lang J., Schiex T. Selecting preferred solutions in fuzzy constraint satisfaction problems // Proc. 1st European Ccongress on Fuzzy and Intelligent Technologies (EUFIT), 1993.

161. Feder T., Vardi M.Y. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory // SIAM Journal of Computing. 1998. V. 28(1). P. 57–104.

162. Fikes R. A heuristic program for solving problems stated as non-deterministic procedures. Ph.D. dissertation, Comput. Sci. Dept., Carnegie-Mellon Univ. Pittsburgh, 1968.

163. Filkhorn T., Schneider H.-A., Scholz A., Strasser A., and Warkentin P. SVE system verification environment. Technical report, Siemens AG, Technical report SVE, ZFE BT SE, 1995.

164. Constraint logic programming for the crew rostering problem / F. Focacci [et al.] // Proc. PACT97, 1997.

165. Efficient graph automorphism by vertex partitioning / G. Fowler [et al.] // Artificial Intelligence. 1983. V. 21. P. 245–269.

166. Fox M.S. Constraint-Directed Search: A Case Study of Job-Shop Scheduling. Pitman: Morgan Kaufmann, 1987. 184 p.

167. Freuder E. C. Synthesizing constraint expressions // Communications of the Association for Computing Machinery. 1978. 21(11). P. 958-966.

168. Freuder E. C. A sufficient condition for backtrack-free search // Journal of the ACM. 1982. 29(1). P. 24-32.

169. Freuder E. C. A sufficient condition for backtrack-bounded search // Journal of the ACM. 1985. 32(4). P. 755-761.

170. Freuder E. C. Backtrack-free and backtrack-bounded search // Search in Artificial Intelligence / L. Kanal , V. Kumar (eds.). Springer-Verlag, 1988. P. 343-369.

171. Freuder E. C., Mackworth A. K. Constraint satisfaction: An emerging paradigm // Foundations of Artificial Intelligence. 2006. V. 2. P. 13-27.

172. Freuder E. C., Wallace R. J. Partial constraint satisfaction // Artificial Intelligence. 1992. V. 58. P. 21-70.

173. Freuder E. C., Wallace R. J. Suggestion strategies for constraint-based matchmaker agents // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520, 1998.

174. Fromherz M., Gupta V., Saraswat V. Model-based computing: constructing constraint-based software for electro-mechanical systems // Proc. Conf. on Practical Applications of Constraint Technology (PACT95). Paris, 1995. P. 63-66.

175. Frühwirth T. Constraint simplification rules // Constraint Programming: Basics and Trends / A. Podelski, ed. Springer-Verlag, LNCS 910, 1995.

176. Frühwirth T., Abdennadher S. Essentials of Constraint Programming. Springer, 2003. 144 p.

177. Frühwirth T., Brisset P. Optimal planning of digital cordless telecommunication systems // Proc. PACT97, 1997.

178. Fuchs N., Schwitter R. Attempto controlled English (ACE) // Proc. of 1st International Workshop on Controlled Language, 1996.

179. Gaschnig J. A constraint satisfaction method for inference making // Proceedings Twelfth Annual Allerton Conference on Circuit and System Theory. Illinois, Monticello, 1974. P. 866–874.

180. Gaschnig J. A general backtrack algorithm that eliminates most redundant tests // Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77) Cambridge, 1977. P. 457.

181. Gaschnig J. Performance measurement and analysis of certain search algorithms. Technical report CMU-CS-79-124. Carnegie-Mellon University, Computer Science Department, 1979.

182. Extending CLP(FD) with interactive data acquisition for 3D visual object recognition / M. Gavanelli [et al.] // Proc. PACLP99, 1999.

183. Gecode: Generic Constraint Development Environment. http://www.gecode.org (дата обращения 07.08.2010).

184. Georget Y., Codognet P., Rossi F. Constraint retraction in clp(fd): Formal framework and performance results // Constraints: An International Journal. 1999. V. 4.

185. Georget Y., Codognet P. Compiling semiring-based constraints with clp(fd,s) // Proc. CP98 / M. Maher, J-F.Puget (eds.). Springer-Verlag, LNCS 1520, 1998.

186. Gervet C. Conjunto: Constraint logic programming with finite set domains // Logic Programming — Proc. of the 1994 International Symposium / M. Bruynooghe, ed. Cambridge: The MIT Press, 1994. P. 339–358.

187. Gervet C., Caseau Y., and Montaut D. On refining ill-defined constraint problems: A case study in iterative prototyping // Proc. PACLP99, 1999.

188. Ginsberg M. L. Essentials of Artificial Intelligence. San Mateo: Morgan Kaufmann, 1993. 430 p.

189. Ginsberg M.L. Dynamic Backtracking // Journal of Artificial Intelligence Research. 1993. v. 1. P. 25-46.

190. Ginsberg M.L., McAllester D.A. GSAT and dynamic backtracking // Proceedings of The 4th International Conference on Principles of Knowledge Representation and Reasoning, 1994.

191. Glaisner F., Richard L.-M. FORWARD-C: A refinery scheduling system // Proc. PACT97, 1997.

192. Goldin D.Q., Kanellakis P. Constraint query algebras // Constraints: An International Journal. 1996. V.1. P. 45-84.

193. Goltz H.-J., Matke D. Combined interactive and automatic timetabling // Proc. PACLP99, 1999.

194. Gomes C.P. Artificial intelligence and operations research: challenges and opportunities in planning and scheduling // Knowl. Eng. Rev. 2000. V. 15. P. 1-10.

195. Gottlob G., Leone N., Scarcello F. A comparison of structural CSP decomposition methods // Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99). Stockholm: Morgan Kaufmann, 1999. P. 394-399.

196. Gottlob G., Leone N., Scarcello F. Hypertree decompositions and tractable queries // Proceedings of the 18th ACM International Symposium on Principles of Database Systems. Philadelphia: Association for Computing Machinery, 1999. P. 21-32.

197. Gottlob G., Leone N., Scarcello F. Hypertree decomposition and tractable queries // Journal of Computer and System Sciences. 2002. V. 64 (3). P. 579–627.

198. Grohe M. The structure of tractable constraint satisfaction problems // Proceedings of the 31st Symposium on Mathematical Foundations of Computer Science, LNCS 4162. Springer-Verlag, 2006. P. 58–72.

199. Dedale: A spatial constraint database / S. Grumbach [et al.] // Proc. Intl. Workshop on Database Programming Languages (DBPL'97). Estes Park, Colorado, USA, 1994.

200. Gu J. Parallel Algorithms and Architectures for Very Fast AI Search. Ph.D. thesis, University of Utah,1989.

201. Gyssens M., Jeavons P.G., Cohen D.A. Decomposing constraint satisfaction problems using database techniques // Artificial Intelligence. 1994. V. 66. P. 57-89.

202. Han C., Lee C. Comments on Mohr and Henderson's path consistency algorithm // Artificial Intelligence. 1988. V. 36. P. 125–130.

203. Haralick R. M., Elliot G. L. Increasing tree search efficiency for constraint satisfaction problems // Artificial Intelligence. 1980. V. 14(3). P. 263-313.

204. Programming languages for distributed applications / S. Haridi [et al.] // New Generation Computing. 1998. V.16. P. 223-261.
205. Haridi S., Janson S. Kernel Andorra Prolog and its computational model // Proc. ICLP90. MIT Press, 1990.
206. Harvey W., Ginsberg M. Limited discrepancy search // Proc. IJCAI95, 1995.
207. Henz M., Smolka G., Wurtz J. Oz — a programming language for multiagent systems // 13th International Joint Conference on Artificial Intelligence / Ruzena Bajcsy, ed. vol. 1. Morgan Kaufmann Publishers, 1993. P. 404-409.
208. Hermenegildo M. and the CLIP group. Some methodological issues in the design of CIAO, a generic, parallel concurrent constraint system // Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming, May 02-04, 1994. Springer-Verlag, LNCS 874, 1994. P. 123-133.
209. Hicks I.V., Koster A.M.C.A., Kolotoglu E. Branch and tree decomposition techniques for discrete optimization // Tutorials in Operations Research. New Orleans: INFORMS, 2005. P.1-29. http://ise.tamu.edu/people/faculty/Hicks/bwtw.pdf (дата обращения 07.08.2010)
210. Hobby D., McKenzie R.N. The Structure of Finite Algebras, volume 76 of Contemporary Mathematics. Providence: American Mathematical Society, 1988. 203 p.
211. Hooker J.N., Yan H. Verifying logic circuits by Benders decomposition // Principles and Practice of Constraint Programming / V.A. Saraswat, P. van Hentenryck (eds.) Cambridge: MIT Press, 1994.
212. Hooker J.N., Ottosson G., Thorsteinsson E.S., and Kim H.-J. A scheme for unifying optimization and constraint satisfaction methods // The Knowledge Engineering Review. 2000. V.15, N.1. P. 11-30.
213. Hooker J.N. Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction. New York: Wiley, 2000. 495 p.
214. Hower W. Constraint satisfaction — Algorithms and complexity analysis // Information Processing Letters. 1995. Volume 55 (3). P. 171-178.
215. Hyvonen E. Interval constraints spreadsheets for financial planning // Proc. 1st Int. Conf. on Artificial Intelligence Applications on Wall Street. IEEE Computer Society Press, 1991.
216. ILOG SOLVER: Object oriented constraint programming. ILOG SA, 12, Av. Raspail, BP 7, 94251 Gentilly cedex, France, 1995.
217. ILOG Scheduler. http://www.ilog.fr/products/scheduler/
218. Istrate G. Counting, structure identification and maximum consistency for binary constraint satisfaction problems // Proceedings 3rd Int. Conference on Principles and Practice of Constraint Programming / G. Smolka (ed.). Springer-Verlag, LNCS 1330, 1997. P. 136-149.
219. Jackson D. Software Abstractions: Logic, Language, and Analysis. Cambridge: The MIT Press, 2006. 366 p.
220. Jaffar J., Lassez J.L. Constraint logic programming // Proc. 14th ACM Symp. Principles Programming Lang. POPL-87, Munich, ACM Press, 1987. P. 111–119.
221. Jaffar J., Maher M.J. Constraint logic programming: A survey // Journal of Logic Programming. 1994. V. 19. P. 503-581.

222. The CLP(R) Language and System / J. Jaffar // ACM Transactions on Programming Languages and Systems. 1992. V. 14. P. 339-395.

223. Jampel M. (ed.) Over-Constrained Systems. Springer-Verlag, LNCS 1106, 1996.

224. Janson S. AKL — A Multiparadigm Programming Language. PhD thesis, Uppsala Theses in Computer Science 19, Uppsala University, and SICS Dissertation Series 14, 1994.

225. Jeavons P.G. On the algebraic structure of combinatorial problems // Theoretical Computer Science. 1998. V. 200. P. 185-204. ftp://ftp.cs.rhul.ac.uk/pub/constraints/algebraic.ps (дата обращения 07.08.2010)

226. Jeavons P.G. Constructing constraints // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520, 1998. P. 2-16. ftp://ftp.cs.rhul.ac.uk/pub/constraints/constructing.ps (дата обращения 07.08.2010)

227. Jeavons P.G., Cohen D.A., Gyssens M. Closure properties of constraints // Journal of ACM. 1997. V. 44. P. 527-548.

228. Jeavons P.G., Cohen D.A., Cooper M.C. Constraints, consistency and closure // Artificial Intelligence. 1998. V. 101. P. 251–265. ftp://ftp.cs.rhul.ac.uk/pub/constraints/conconclo.ps (дата обращения 07.08.2010)

229. Jeavons P.G., Cohen D.A. An algebraic characterization of tractable constraints // Proc. of Computing and Combinatorics (COCOON'95). Springer-Verlag, LNCS 959, 1995. P. 633-642.

230. Jeavons P.G., Cohen D.A., Gyssens M. A unifying framework for tractable constraints // Proc. 1st Int. Conf. on Principles and Practice of Constraint Programming (CP95), Springer-Verlag, LNCS 976, 1995. P. 276-291.

231. Jeavons P.G., Cohen D.A., Gyssens M. How to determine the expressive power of constraints // Constraints: An international Journal 1999. V. 4. P. 113–131.

232. Jeavons P.G., Cohen D.A., Gyssens M. A test for tractability // Proc. 2nd Int. Conf. on Principles and Practice of Constraint Programming (CP96). Springer-Verlag, LNCS 1118, 1996. P. 267-281

233. Jeavons P.G., Cooper M.C. Tractable constraints on ordered domains // Artificial Intelligence. 1995. 79(2). P. 327–339,

234. Jonsson P., Krokhin A. Recognizing frozen variables in constraint satisfaction problems // Theoret. Comput. Sci. 2004. V. 329 (1–3). P. 93–113.

235. Kam R.W.L., Lee J.H.M. Fuzzifying the constraint hierarchies framework // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520, 1998.

236. Kanellakis P., Goldin D.Q. Constraint programming and database query languages // Proc. Int. Symp. on Theoretical Aspects of Computer Software. Springer-Verlag, LNCS 789, 1994. P. 96-120.

237. Kanellakis P., Kuper G., Revesz P. Constraint query languages // Journal of Computer and System Sciences. 1995. V. 51. P. 25-52.

238. Katsirelos G., Bacchus F. Generalized NoGoods in CSPs // Proc. of 20th AAAI, 2005. P. 390–396.

239. Katsirelos G. Nogood Processing in CSPs. Phd thesis. University of Toronto, 2008.

240. Kautz H., Selman B. Planning as Satisfiability // Proceedings of European Conference on Artificial Intelligence ECAI-92 (Vienna, 1992). Chichester: John Wiley and Sons, 1992. P. 359-363.

241. The approximability of constraint satisfaction problems / S. Khanna [et al.] // SIAM J. Comput. 2000. V. 30(6). P. 1863-1920.

242. Estimating search tree size / P. Kilby [et al.] // Proceedings of AAAI-2006. 2006. P. 1014-1019.

243. Kirkpatrick S., Gelatt C. D., Vecchi M. P. Optimization by simulated annealing // Science. 1983. V. 220. P. 671-680.

244. Kirousis L., Kolaitis Ph. A dichotomy in the complexity of propositional circumscription // Logic in Computer Science (Boston, MA, 2001). IEEE Comput. Soc., 2001. P. 71-80.

245. Knuth D. Estimating the efficiency of backtrack programs // Mathematics of Computation. 1975. V. 29(129). P. 121-136.

246. Kokkoras F., Gregory S. D-WMS: Distributed workforce management using CLP // Proc. 4th Int. Conference on the Practical Applications of Constraint Technology (PACT'98). London, UK, 25-27 March, 1998. P. 129-146.

247. Kolaitis P., Vardi M. Conjunctive-query containment and constraint satisfaction // Journal of Computer and System Sciences. 2000. V. 61. P. 302–332.

248. Kondrak G., van Beek P. A theoretical evaluation of selected backtracking algorithms // Artificial Intelligence. 1997. V. 89. P. 365-387.

249. Korf R. Improved limited discrepancy search // Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96) (Portland). 1996. P. 209–215.

250. Koubarakis M. Querying Temporal Constraint Networks: A Unifying Approach // Applied Intelligence. 2002. V.17. P. 297-311.

251. Krokhin A., Jeavons P., and Jonsson P. Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra // Journal of the ACM. 2003. V. 50. P. 591–640.

252. Krokhin A., Bulatov A., Jeavons P. Functions of multiple-valued logic and the complexity of constraint satisfaction: A short survey // Proceedings 33rd IEEE International Symposium on Multiple-Valued Logic (ISMVL 2003). IEEE Computer Society, 2003. P. 343–351.

253. Krokhin A., Bulatov A., Jeavons P. The complexity of constraint satisfaction: an algebraic approach // Structural Theory of Automata, Semigroups, and Universal Algebra, volume 207 of NATO Science Series II: Math., Phys., Chem., Springer Verlag, 2005. P. 181-213.

254. Kuipers B. Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge. MIT Press, 1994. 418 p.

255. Kumar V. Algorithms for constraint satisfaction problems: A survey // AI Magazine. 1992. 13(1). P. 32-44.

256. Laburthe F., Caseau Y. SALSA: A language for search algorithms // Proceedings of the 4h International Conference on the Principles and Practice of Constraint Programming (CP'98) (Pisa, Italy), 1998.

257. Larrosa J., Morancho E., Niso D. On the practical applicability of bucket elimination: Still-life as a case study // Journal of Artificial Intelligence Research. 2005. V. 23. P. 421–440.

258. Lauriere J-L. ALICE: A language and a program for stating and solving combinatorial problems // Artificial Intelligence. 1978. V. 10(1). P. 29–127.

259. Leler Wm. Constraint Programming Languages, Their Specification and Generation. Reading: Addison-Wesley, 1988. 202 p.

260. Engineering dynamic scheduler for Work Manager / D. Lesaint [et al.] // BT Technology Journal. 1998. V. 16. P. 16–29.

261. Lloyd J. W. Foundations of Logic Programming. Berlin: Springer Verlag, 1987. 212 p.

262. Mackworth A.K. Consistency in networks of relations // Artificial Intelligence. 1977. 8(1). P. 99-118.

263. Mackworth A.K. Constraint satisfaction // Encyclopedia of Artificial Intelligence (second edition) / S. Shapiro (Ed.). New York: Wiley, 1992. Vol. 1. P. 285-293.

264. Mackworth A.K. On reading sketch maps // Proceedings of the Fifth International Joint Conference on Artificial Intelligence Cambridge, 1977. P. 598–606.

265. Mackworth A.K., Freuder E.C. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems // Artificial Intelligence. 1985. V. 25 (1). P. 65–74.

266. Marriott K., Stuckey P. J. Programming with Constraints: An Introduction. Cambridge: MIT Press, 1998. 483 p.

267. Zinc 0.1: Language and libraries. Technical report / K. Marriott [et al.]. Monash University, 2006.

268. The design of the Zinc modelling language / K. Marriott [et al.] // Constraints 2008. V. 13, N 3. P. 229-267.

269. Maximuck W., Reviakin M. A prototype of train schedule module // Proc. PACT98, 1998.

270. McAloon K. Cooperating solvers: A progress report // Proc. PACLP99, 1999.

271. McAloon K., Tretkoff C. 2LP: Linear programming and logic programming // Principles and Practice of Constraint Programming / V. Saraswat, P. Van Hentenryck (eds). MIT Press, 1995. P. 99-114.

272. McDermott D. A general framework for reason maintenance // Artificial Intelligence. 1991. V. 50. P. 289–329.

273. McGregor J.J. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms // Information Sciences. 1979. V. 19(3). P. 229-250.

274. McKenzie R.N., McNulty G.F., Taylor W.F. Algebras, Lattices and Varieties, volume I, California: Wadsworth and Brooks, 1987.

275. Mehlhorn K., Thiel S. Faster algorithms for bound-consistency of the sortedness and alldifferent constraint // Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming. Singapore, 2000. P. 306–319.

276. Melzer S. Verification of parallel systems using constraint programming // Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97). Springer-Verlag, LNCS 1330, 1997.

277. Menzel W. Constraint satisfaction for robust parsing of spoken language // Journal of Experimental and Theoretical Artificial Intelligence. 1998. V. 10. P. 77-89.

278. Meseguer P. Constraint satisfaction problems: an overview // AI Communications. 1989. V. 2. P. 3-17.

279. Meseguer P. Interleaved depth-first search // Proceedings of the 15th International Joint Conference on Artificial Intelligence (Nagoya, Japan), 1997.

280. Meyer auf'm Hofe H. ConPlan/SIEDAplan, personnel assignment as a problem of hierarchical constraint satisfaction // Proc. PACT 97, 1997. P. 257-271.

281. Michel L., Van Hentenryck P. Localizer // Constraints. 2000. V. 5(1/2). P. 43–84.

282. Miguel I., Shen Q. Extending qualitative modelling for simulation of time- delayed behaviour // Proceedings of The 12th International Workshop on Qualitative Reasoning. Massachusetts, Cape Cod, 1998. P. 161–166.

283. Miguel I., Shen Q. Solution techniques for constraint satisfaction problems: foundations // Artificial Intelligence Review. 2001. V. 15. P. 241-265.

284. Miguel I., Shen Q. Solution techniques for constraint satisfaction problems: advanced approaches // Artificial Intelligence Review. 2001. V. 15. P. 267-291.

285. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems / S. Minton [et al.] // Artificial Intelligence. 1992. 55(1-3). P. 161-205.

286. Mitchell D. Hard problems for CSP algorithms // Proceedings of the AAAI National Conference, 1998. P. 398–405.

287. Mohr R., Henderson T. C. Arc and path consistency revisited // Artificial Intelligence. 1986. 28(2). P. 225-233.

288. Mohr R., Masini G. Good old discrete relaxation // Proceedings of the 8th European Conference on Artificial Intelligence. München, 1988. P. 651–656.

289. Montanari U. Networks of constraints: Fundamental properties and applications to picture processing // Information Sciences. 1974. 7(2). P. 95-132.

290. Montelius J., Ali K.A.M. An and/or-parallel implementation of AKL // New Generation Computing. 1996. V. 14. P. 31-52.

291. Moskewicz M., Madigan C., Zhao Y., Zhang L., and Malik S. Chaff: Engineering an efficient sat solver / M. Moskewicz [et al.] // Proc. of the Design Automation Conference (DAC), 2001.

292. http://www.mozart-oz.org/. The Mozart project, 2008 (дата обращения 07.08.2010).

293. Nadel B.A. Constraint satisfaction algorithms // Comput. Intelligence. 1989. V. 5. P. 188–224.

294. Nadel B.A. Constraint satisfaction in Prolog: Complexity and theory-based heuristics // Information Sciences, An International Journal. 1995. vol. 83. P. 113-131

295. Neumaier A. Complete search in continuous global optimization and constraint satisfaction // Acta Numerica. Cambridge University Press, 2004. P. 271-369.

296. Le Pape C., Baptiste P. A constraint programming library for preemptive and non-preemptive scheduling // Proc. PACT97, 1997.

297. Le Pape C., Baptiste P. Resource constraints for preemptive job-shop scheduling // Constraints: An International Journal, 1998. V. 3. P. 263-287.

298. Pareschi R. Work force management in telecommunications and beyond telecommunications // Proc. PACLP99, 1999.

299. Pachet F. Constraints and multimedia // Proc. PACLP99, 1999.

300. Papadimitriou C.H. Computational Complexity. Reading: Addison-Wesley, 2005. 523 p.

301. Pearson J., Jeavons P. A survey of tractable constraint satisfaction problems. Technical report CSD-TR-97-15. U. of London, Royal Holloway College, 1997.

302. Peirce C.S. Collected Papers / C. Hartshorne, P. Weiss (eds.), Vol. III. Harvard University Press, 1933.

303. Perrett M. Using constraint logic programming techniques in container port planning // ICL Technical Journal. 1991. P. 537-545.

304. Perron L. Search procedures and parallelism in constraint programming // Proceedings of the 5th International Conference on the Principles and Practice of Constraint Programming (CP'99) (Alexandra), 1999.

305. Pesant G., Gendreau M., and Rousseau J.-M. GENIUS-CP: a generic single- vehicle routing algorithm // Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (CP97). Springer-Verlag, LNCS 1330, 1997.

306. Pöschel R., Kalužnin L.A. Funktionen- und Relationenalgebren. DVW, Berlin, 1979.

307. Le manuel de Prolog IV / F. Benhamou [et al.]. Marseille: PrologIA, 1996.

308. Prosser P. A reactive scheduling agent // Proceedings of the 11th International Joint Conference on Artificial Intelligence IJCAI-89. Detroit, 1989. P. 1004–1009.

309. Prosser P. Hybrid algorithms for constraint satisfaction problems // Computational Intelligence. 1993. V. 9. P. 268-299.

310. Purvis L., Jeavons P. Constraint tractability theory and its application to the product development process for a constraint-based scheduler // Proceedings of 1st International Conference on The Practical Application of Constraint Technologies and Logic Programming, PACLP'99. Practical Applications Company, 1999. P. 63–79.

311. Régin J. A filtering algorithm for constraints of difference in CSPs // Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94). Seattle: AAAI Press, 1994. P. 362-367.

312. Régin J. Generalized arc consistency for global cardinality constraint // Proceedings of the Thirteenth National Conference on Artificial Intelligence. Portland, 1996. P. 209–215.

313. Régin J. The symmetric alldiff constraint // Proc. IJCAI99, 1999.

314. Reith S., Vollmer H. Optimal satisfiability for propositional calculi and constraint satisfaction problems // Inform. and Comput. 2003. V. 186. P. 1–19.

315. Renker G., Ahriz H. Building models through formal specification // Proc. of the First Int. Conf. on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer, LNCS 3011, 2004. P. 395–401.

316. Revesz P.Z. Safe stratified Datalog with integer order programs // Proceedings of 1st Int. Conference on Principles and Practice of Constraint Programming (CP95). Springer-Verlag, LNCS 976, 1995. P. 154–169.

317. Revesz P.Z. Safe Datalog queries with linear constraints // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520, 1998. P. 355-369.

318. http://www.comp.rgu.ac.uk/staff/ha/ZCSP/ (дата обращения 07.08.2010).
319. Rit J.F. Propagating temporal constraints for scheduling // Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, 1986. P. 383–386.
320. Robertson N., Seymour P. D. Graph minors, ii. Algorithmic aspects of tree-width // Journal of Algorithms. 1986. 7(3). P. 309-322.
321. Rodosek R., Wallace M. A generic model and hybrid algorithm for hoist scheduling probleems // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520, 1998, pp. 385-399.
322. Rossi F. Constraint (logic) programming: A survey on research and applications // New Trends in Constraints / K.R. Apt [et al.] (eds.). Berlin: Springer, 1999. P. 40–74.
323. Rossi F., Petrie C., Dhar V. On the equivalence of constraint satisfaction problems. Technical Report ACT-AI-222-89, MCC, Austin, Texas, 1989.
324. Rossi F., van Beek P., Walsh T. (eds.) Handbook Of Constraint Programming. Elsevier, 2006. 978 p.
325. Rossi F., van Beek P., Walsh T. Chapter 4 "Constraint Programming" in Foundations of Artificial Intelligence. // Handbook of Knowledge Representation / F. van Harmelen, V. Lifschitz, B. Porter (eds.). 2008. Volume 3. P. 181-211.
326. Rossi F., Montanari U. Exact solution of networks of constraints using per- fect relaxation // Proc. International Conference on Knowledge Representation (KR89). Morgan Kaufmann, 1989.
327. Rossi F., Sperduti A. Learning solution preferences in constraint problems // Journal of Experimental and Theoretical Computer Science, 1998. Vol 10. P. 103- 116.
328. Ruttkay Zs. Fuzzy constraint satisfaction // Proc. 3rd IEEE International Conference on Fuzzy Systems, 1994. P. 1263-1268.
329. Ruttkay Zs. Constraint satisfaction a survey // CWI Quarterly. 1998. V. 11. P. 163–214.
330. Sabin D., Freuder E. C. Contradicting conventional wisdom in constraint satisfaction // Proceedings of 11th European Conference on Artificial Intelligence (ECAI 94). Amsterdam: Wiley, 1994. P. 125-129.
331. Sam-Haroud D., Faltings B. Consistency techniques for continuous constraints // Constraints: An International Journal. 1996. V. 1. P. 85-118.
332. Schaefer T.J. The complexity of satisfiability problems // Proceedings 10th ACM Symposium on Theory of Computing (STOC'78), 1978. P. 216–226.
333. Schiex T. Possibilistic constraint satisfaction problems, or "how to handle soft constraints?" // Proc. 8th Conf. of Uncertainty in AI, 1992. P. 269-275.
334. Schiex T., Fargier H., and Verfaille G. Valued constraint satisfaction problems: hard and easy problems // Proc. IJCAI95. Morgan Kaufmann, 1995. P. 631-637.
335. Schulte C. Oz explorer: a visual constraint programming tool // Proceedings of the 14th Int. Conf. on Logic programming. MIT Press, 1997. P. 286-300.
336. Schulte C. Programming constraint inference engines // Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming (Schloss Hagenberg, Linz, Austria, CP97). Springer-Verlag, LNCS 1330, 1997. P. 519–533.
337. Seidel R. A new method for solving constraint satisfaction problems // Proceedings of IJCAI-81, 1981. P. 338–342.

338. Sergiou K., Walsh T. The difference all-difference makes // Proc. IJCAI99, 1999.

339. Spatial modelling with geometric constraints / B. Seybold [et al.] // Proc. PACT97, 1997.

340. Shaw P. Using constraint programming and local search methods to solve vehicle routing problems // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520, 1998.

341. Shen Q., Leitch R. Fuzzy Qualitative Simulation // IEEE Transactions on Systems. Man, and Cybernetics. 1993. V. 23(4). P. 1038–1061.

342. Shvetsov I., Kornienko V., Preis S. Interval spreadsheet for problems of financial planning // PACT'97, England, London, 1997. P. 373-385.

343. Schwalb E., Vila L. Temporal Constraints: A Survey // Constraints. 1998. V. 3. P. 129-149.

344. Simonis H. Visual CHIP — a visual language for defining constraint programs // CCL II workshop, 1997.

345. Simonis H., Aggoun A. Search tree debugging. Technical report, COSYTEC SA, 1997.

346. Simonis H., Cornelissens T. Modelling producer/consumer constraints // Proceedings 1st Int. Conference on Principles and Practice of Constraint Programming (CP95). Springer-Verlag, LNCS 976, 1995. P. 449-462.

347. Simonis H. Application development with the CHIP system // Proceedings of the ESPRIT WG CONTESSA Workshop on Constraint Databases and Applications (September 08 - 09, 1995) / G. M. Kuper, M. Wallace (eds.). LNCS 1034. London: Springer-Verlag, 1996. P. 1-21.

348. Simonis H. Building industrial applications with constraint programming // Constraints in computational logics: theory a nd applications. New York: Springer-Verlag, 2001. P. 271-309.

349. Simonis H. Models for global constraint applications // Constraints. 2007. V. 12. P. 63-92.

350. Simonis H., Charlier P. Cobra — a system for train crew scheduling // Proc. DIMACS workshop on constraint programming and large scale combinatorial optimization, 1998.

351. Simons R.V. Mathematical programming modeling using MGG // IMA Journal of Mathematics in Management. 1987. V. 1. P. 267–276.

352. Smedback G., Carro M., and Hermenegildo M. Interfacing Prolog and VRML and its applications to constraint visualization // The Practical Application of Constraint Technologies and Logic Programming (PACLP99),1999. P. 453-471.

353. Smolka G. A foundation for higher-order concurrent constraint programming // 1st International Conference on Constraints in Computational Logics / Jean-Pierre Jouannaud (ed.). LNCS 845. Berlin: Springer-Verlag, 1994. P. 50-72.

354. Sosic R., Gu J. Efficient local search with conflict minimization: A case study of the n-queens problem // IEEE Transactions on Knowledge and Data Engineering. 1994. 6(5). P. 661-668.

355. Stallman R.M., Sussman G.J. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis // Artificial Intelligence. 1977. 9(2). P. 135-196.

356. Sutherland I. Sketchpad: A man-machine graphical communication system // Proceedings of the Spring Joint Computer Conference. IFIPS, 1963. P. 329-346.

357. Resource allocation in distributed factory scheduling / K.P. Sycara [et al.] // IEEE Expert. 1991. V. 6. P. 29-40.

358. Szendrei A. Clones in Universal Algebra, volume 99 of Seminaires de Mathematiques Superieures. University of Montreal, 1986.

359. A constraint-based approach for visualization and animation / S. Takahashi [et al.] // Constraints: An International Journal. 1998. V. 3. P. 61-86.

360. Tamassia R. Constraints in graph drawing algorithms // Constraints: An International Journal. 1998. V. 3. P. 87-120.

361. Tarjan R.E., Yannakakis M. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs // SIAM Journal of Computing. 1984. V. 13. P. 566–579.

362. Toman D. Memoing evaluation for constraint extensions of Datalog // Constraints: An International Journal. 1997. V. 2. P. 337-360.

363. Tsang E.P.K. The consistent labeling problem in temporal reasoning // Proceedings of the 6th National Conference on Artificial Intelligence, 1987. P. 251–255.

364. Tsang E. Foundations of Constraint Satisfaction. New York: Academic Press, 1993. 421 p.

365. A computer aided constraint programming system / E. Tsang [et al.] // Proc. PACLP99, 1999.

366. Ullman J.R. An algorithm for subgraph isomorphism // Journal of the ACM. 1976. V. 23. P. 31–42.

367. Van Beek P. Reasoning about qualitative temporal information // Artificial Intelligence. 1992. V. 58. P. 297-–326.

368. Van Hentenryck P. Constraint Satisfaction in Logic Programming. Cambridge: MIT Press, 1989. 224 p.

369. Van Hentenryck P., Deville Y., Teng C.M. A generic arc-consistency algorithm and its specializations // Artificial Intelligence. 1992. V. 57. P. 291–321.

370. Van Hentenryck P. Helios: a modeling language for global optimization // Proceedings PACT'96, 1996. P. 317-335.

371. Van Hentenryck P., Michel L., Deville Y. Numerica: A Modeling Language for Global Optimization. Cambridge: MIT Press, 1997. 210 p.

372. Van Hentenryck P., Saraswat V. (eds.) Special issue on Strategic Directions on Constraint Programming // Constraints: An International Journal. 1997. V.2, N 1.

373. Van Hentenryck P. Visual solver: A modeling language for constraint program- ming // Proc. 3rd Int. Conf. on Principles and Practice of Constraint Program- ming (CP97). Springer-Verlag, LNCS 1330, 1997.

374. Van Hentenryck P. The OPL Optimization Programming Language. MIT Press, 1999. 255 p.

375. Van Hentenryck P., Perron L., Puget J.-F. Search and strategies in OPL // ACM Transactions on Computational Logic. 2000. V. 1. P. 1–36.

376. Van Hentenryck P., Michel L. Constraint-Based Local Search. Cambridge: MIT Press, 2005. 442 p.

377. Wallace M. Practical applications of constraint programming // Constraints: An International Journal. 1996. V. 1. P. 139-168.

378. Wallace M., Novello S., and Schimpf J. Eclipse: A platform for constraint logic programming. Technical report, IC-Parc, London, Imperial College, 1997.

379. Wallace M., Novello S., and Schimpf J. Eclipse — a platform for constraint programming // ICL Systems Journal. 1997. V. 12(1). P. 159-200.

380. Wallace R.J., Freuder E.C. Stable solutions for dynamic constraint satisfaction problems // Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP98). Springer-Verlag, LNCS 1520, 1998.

381. Walsh T. Depth-bounded discrepancy search // Proceedings of the 15th International Joint Conference on Artificial Intelligence. Nagoya, Japan, 1997.

382. Walsh T. Reformulating propositional satisfiability as constraint satisfaction // Proceedings of the 4th International Symposium on Abstraction, Reformulation, and Approximation (July 26-29), 2000. P. 233-246.

383. Waltz D. Generating semantic descriptions from drawings of scenes with shadows. Technical Report AI271. MIT, 1972.

384. Waltz D.L. Understanding line drawings of scenes with shadows // Psychology of Computer Vision. New York: McGraw-Hill, 1975. P. 19-92.

385. Yakhno T.M., Zilberfaine V.Z., Petrov E.S. Applications of ECLiPSe: Interval domain library // Proc. PACT97, 1997.

386. Yokoo M. Asynchronous weak-committment search for solving distributed constraint satisfaction problems // Proc. Principles and Practice of Constraint Programming (CP95). Springer-Verlag, LNCS 976, 1995.

387. Zabih R., McAllester D. A Rearrangement Strategy for Determining Propositional Satisfiability // Proceedings of The 7th National Conference on Artificial Intelligence. St Paul, 1988. P. 155-160.