Труды Кольского научного центра РАН. Серия: Технические науки. 2025. Т. 16, № 3. С. 117–130.
Transactions of the Kola Science Centre of RAS. Series: Engineering Sciences. 2025. Vol. 16, No. 3. P. 117–130.

Научная статья
УДК 004.832
doi:10.37614/2949-1215.2025.16.3.008

РЕШЕНИЕ ЗАДАЧ ГЕНЕРАТИВНОГО ДИЗАЙНА С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ УДОВЛЕТВОРЕНИЯ ОГРАНИЧЕНИЙ

Полина Владимировна Таран[1], Александр Анатольевич Зуенко[2]✉
[1, 2]Институт информатики и математического моделирования имени В. А. Путилова Кольского научного центра Российской академии наук, Апатиты, Россия
[1]p.taran@ksc.ru, https://orcid.org/0009-0003-9485-7004
[2]zuenko@ksc.ru✉, https://orcid.org/0000-0002-7165-6651

Аннотация

Данная работа посвящена рассмотрению методов решения задачи генеративного дизайна. В настоящее время подобные задачи решаются, как правило, с использованием нейросетевого подхода. В представленных исследованиях предлагается задачу генеративного дизайна ставить как задачу удовлетворения ограничений и решать с использованием технологии программирования в ограничениях. Предлагаемый подход иллюстрируется на примере задачи проектирования двумерной пространственной среды с учетом разнородных требований к взаимному расположению объектов среды.

Ключевые слова:

генеративный дизайн, задача удовлетворения ограничений, программирование в ограничениях, комбинаторный поиск, распространение ограничений

Благодарности:

работа выполнена в рамках темы НИР «Методы и информационные технологии мониторинга и управления региональными критическими инфраструктурами Арктической зоны Российской Федерации» (FMEZ-2025-0054).

Для цитирования:

Таран П. В., Зуенко А. А. Решение задач генеративного дизайна с использованием методов удовлетворения ограничений // Труды Кольского научного центра РАН. Серия: Технические науки. 2025. Т. 16, № 3. С. 117–130. doi:10.37614/2949-1215.2025.16.3.008.

Original article

SOLVING GENERATIVE DESIGN PROBLEMS USING CONSTRAINT SATISFACTION METHODS

Polina V. Taran[1], Aleksandr A. Zuenko[2]✉
[1, 2]Putilov Institute for Informatics and Mathematical Modeling of the Kola Science Centre of the Russian Academy of Sciences, Apatity, Russia
[1]p.taran@ksc.ru, https://orcid.org/0009-0003-9485-7004
[2] a.zuenko@ksc.ru✉, https://orcid.org/0000-0002-7165-6651

Abstract

The paper is devoted to the consideration of methods for solving the generative design problems. Traditionally such problems are solved using a neural network approach. In the study, it is proposed to represent the problem of generative design as a constraint satisfaction problem and solving it using constraint programming technology. The proposed approach is illustrated by the example of the problem of two-dimensional spatial scena design, taking into account the heterogeneous requirements for the relative location of objects.

Keywords:

generative design, constraint satisfaction problem, constraint programming, combinatorial search, constraint propagation

Acknowledgments:

The work was carried out within the framework of the current research topic "Methods and information technologies for monitoring and management of regional critical infrastructures in the Arctic zone of the Russian Federation" (registration number FMEZ-2025-0054).

For citation:

Taran P. V., Zuenko A. A. Solving generative design problems using constraint satisfaction methods. *Trudy Kol'skogo nauchnogo centra RAN. Seriya: Tekhnicheskie nauki* [Transactions of the Kola Science Centre of RAS. Series: Engineering Sciences], 2025, Vol. 16, No. 3, pp. 117–130. doi:10.37614/2949-1215.2025.16.3.008.

Введение

Процесс получения решения в рамах технологии генеративного дизайна (проектирования), иногда называемой технологией порождающего проектирования, заключается в том, что пользователь задает технические требования и параметры задачи, а программа генерирует варианты ее решения [1]. Пространство поиска при этом достаточно велико, чтобы человек вручную мог корректно построить все интересующие альтернативы, а затем их проанализировать. Алгоритм построения решений, как правило, не оперирует принципиально новыми идеями, а комбинирует хорошо зарекомендовавшие себя варианты реализации отдельных компонентов проектируемой системы и предлагает решения, оптимальные по определенной системе критериев.

Среди методов генеративного дизайна можно выделить следующие группы [2–4]: методы с использованием грамматик формы [5; 6], L-системы [7], клеточные автоматы [8], методы на основе эволюционных вычислений, генетических алгоритмов, роевого интеллекта [9; 10].

В последнее время подобные задачи решаются с использованием нейросетевого подхода, а именно с помощью генеративных нейронных сетей [11; 12]. В представленных исследованиях предлагается задачу генеративного дизайна ставить как задачу удовлетворения ограничений и решать с использованием технологии программирования в ограничениях [13]. Предлагаемый подход иллюстрируется на примере задачи проектирования двумерной пространственной среды с учетом разнородных требований к взаимному расположению объектов среды.

Задача генеративного дизайна

Генеративный дизайн (Generarive Desing) — парадигма проектирования, в которой процесс создания проектных решений автоматизирован и делегирован вычислительной системе [4]. В отличие от традиционного компьютерного моделирования, в котором инженер вручную создает проектное решение, использование парадигмы генеративного дизайна позволяет описать задачу, задать целевую функцию и ограничения, а алгоритм автоматически сгенерирует множество решений, удовлетворяющих заданным условиям. Автоматическая генерация множества решений позволяет сократить время проектирования и выбрать из предложенных альтернатив лучшую согласно различным критериям.

Алгоритмы генеративного дизайна изначально использовались в области архитектуры и строительства, но в настоящее время данный подход применяется в различных областях промышленного и потребительского дизайна [4].

Ключевыми этапа генеративного дизайна являются: 1) описание задачи (определение проектной области, задание ограничений и целевой функции); 2) поиск всех решений на основе описанной задачи; 3) анализ решений (интерпретация полученных решений); 4) оценка решений по заданным критериям; 5) постобработка, уточнение предметной области путем задания новых ограничений.

В настоящее время второй этап, который может быть автоматизирован, как правило, реализован с помощью нейросетевого подхода (нейроморфные методы). В настоящей работе предлагается описывать и решать задачу генеративного дизайна как задачу удовлетворения ограничений.

Задача удовлетворения ограничений

Задача удовлетворения ограничений — это задача поиска решений для сети ограничений, которая формализуется в виде трех множеств [13]:

$X = \{X_1, X_2, \ldots X_n\}$ — множество переменных, представляющих элементы задачи;

$D = \{D_1, D_2, \ldots D_n\}$ — множество областей определения (доменов) переменных, причем каждый домен D_i определяет множество допустимых значений, которые может принимать переменная X_i;

$C = \{C_1, C_2, \ldots C_n\}$ — множество ограничений, где каждое ограничение C_j — это отношение, определяющее допустимые комбинации значений для некоторого подмножества переменных.

Существует три вида ограничений. *Унарное* — ограничение затрагивает домен единственной переменной. Например, ограничение $X_1 < m$, где m — любое число. *Бинарное* — ограничение, связывающее между собой две переменные. Например, ограничение $X_1 > X_2$. *Ограничения высшего порядка* — ограничение, связывающее между собой три и более переменные. Каждое ограничение высокого порядка с конечной областью определения можно свести к множеству бинарных

ограничений, введя достаточное количество вспомогательных переменных. Примером ограничения высшего порядка является ограничение *alldiff* [13], которое требует, чтобы все переменные задачи принимали различные значения.

Процесс поиска решения состоит из двух чередующихся этапов:

1) распространение — общее название методов вывода на ограничениях, сводящихся к пошаговому удалению из доменов заведомо недопустимых значений. Изменения в доменах одних переменных по принципу «цепной реакции» приводит к усечению доменов других переменных. Когда пространство задачи больше нельзя сократить за счет распространения, используется ветвление;

2) ветвление. Суть процедуры ветвления заключается в разделении пространства задачи на подпространства меньшей размерности, соответствующие подзадачам. Одним из способов ветвления является присваивание рассматриваемой переменной выбранного значения и дальнейшее распространение для корректировки доменов остальных переменных.

Путем чередования шагов ветвления и распространения каждой переменной задачи присваивается значение, которое является одним из *решений* задачи.

В случае, когда в процессе поиска возникает противоречие (домен одной из переменных становится пустым), используется алгоритм поиска с возвратом, при котором осуществляется возврат из текущего состояния задачи до предыдущего непротиворечивого, и исследуется другая ветвь дерева поиска.

Правило, устанавливающее, какую переменную и значение выбирать при ветвлении, называется *эвристикой*. Удачный выбор эвристики позволяет значительно сократить дерево поиска.

Представление задачи генеративного дизайна как задачи удовлетворения ограничений

В настоящей работе предлагается задачу генеративного дизайна представлять и решать в виде задачи удовлетворения ограничений. В качестве примера задачи генеративного дизайна здесь и далее рассматривается задача проектирования двумерной среды с учетом разнородных требований к взаимному расположению объектов среды.

Для представления данной задачи в рамках задачи удовлетворения ограничений необходимо задать сеть ограничений. В качестве переменных в задаче выступают объекты, которые необходимо расположить в двумерном пространстве. В работе двумерное пространство представлено в виде прямоугольной сетки, разделенной на пронумерованные клетки (ячейки) одинакового размера. Объект описывается в виде набора клеток пространства, а также параметров длины и ширины. Каждое значение домена переменной — это номер клетки, в которой может размещаться верхняя левая клетка объекта. Ниже приведено графическое представление пространства, объекта и его домена (рис. 1).

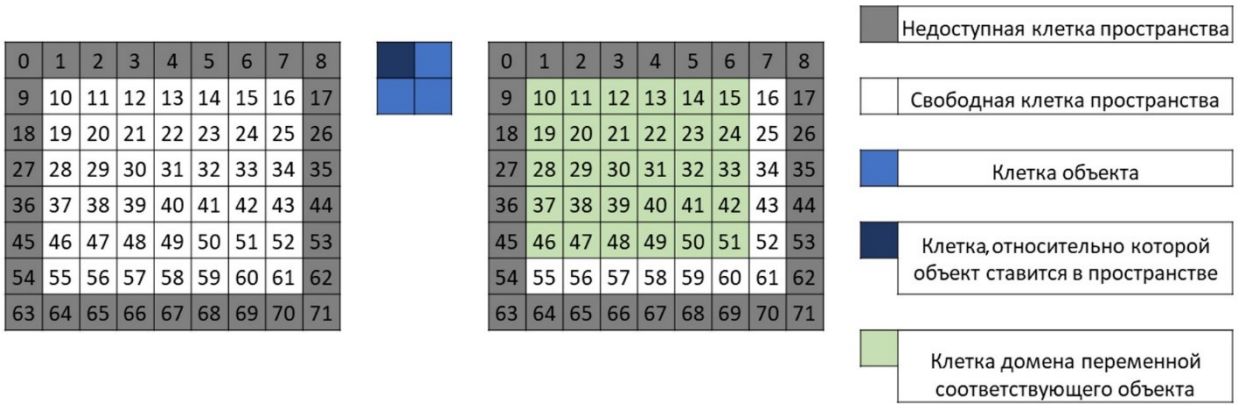


Рис. 1. Графическое представление двумерной сцены, объекта и его домена

Серым на рис. 1 отмечены клетки, которые нельзя занимать, в данном случае это только границы пространства, но незанимаемые клетки могут располагаться и внутри пространства. *Белым* отмечены свободные клетки, в которых можно расположить клетки объектов. *Синим* отмечены клетки объекта, а *темно-синим* — клетка, относительно которой объект ставится в пространстве. *Зеленым* отмечены клетки, в которые можно поставить показанный на рис. 1 объект верхней левой клеткой.

В рамках работы было разработано четыре вида ограничений:

1) ограничение *GDConstraint*, согласно которому в одной клетке пространства не может располагаться два объекта одновременно. Данное ограничение обязательное и является глобальным, т. е. затрагивает все переменные задачи;

2) ограничение *CornerCosntraint* — унарное ограничение, согласно которому объект должен располагаться в углу (т. е. с двух сторон от объекта должны быть недоступные клетки);

3) ограничение *DistanceToWall* — унарное ограничение, согласно которому объект должен располагаться на заданном расстоянии от заданной стены (стеной считается граница пространства или набор недоступных клеток внутри пространства). Расстояние считается в клетках пространства;

4) ограничение *NearConstraint* — бинарное ограничение, связывающее два объекта правилом, что они должны располагаться рядом друг с другом на расстоянии, не превышающем заданное (в клетках пространства).

Все ограничения, кроме первого вида, задаются пользователем факультативно для некоторых из размещаемых объектов.

Для каждого из ограничений был разработан свой алгоритм распространения. Данные алгоритмы подробно описываются далее.

Описание разработанных алгоритмов удовлетворения ограничений

Глобальное ограничение GDConstraint

После того, как у одной из переменных задачи определяется значение (т. е. объект окажется «привязан» к некоторой клетке пространства, расположен в пространстве), необходимо для всех нерасставленных объектов определить недопустимые клетки, чтобы исключить ситуацию, когда несколько объектов занимают одну область пространства

Для этого необходимо из доменов переменных, соответствующих нерасставленным объектам, удалить четыре типа клеток:

клетки, занимаемые поставленным объектом:

$$p + i + width_g \times j, \text{ при } i \in [0, width_1), j \in [0, height_1); \quad (1)$$

клетки выше поставленного объекта:

$$p + i - width_g \times j, \text{ при } i \in [0, width_1), j \in [1, height_2); \quad (2)$$

клетки левее поставленного объекта:

$$p - i + width_g \times j, \text{ при } i \in [1, width_2), j \in [0, height_1); \quad (3)$$

клетки выше и левее поставленного объекта:

$$p - I - width_g \times j, \text{ при } I \in [1, width_2), j \in [1, height_2). \quad (4)$$

В формулах используются следующие обозначения: $width_1$, $height_1$ — ширина и высота поставленного объекта; $width_2$, $height_2$ — ширина и высота объекта, у которого пересчитывается домен; $width_g$ — ширина пространства; p – ячейка, где расположена верхняя левая клетка объекта.

Пример: есть два объекта *Y* и *Z* (рис. 2), которые необходимо расставить в пространстве (рис. 3).

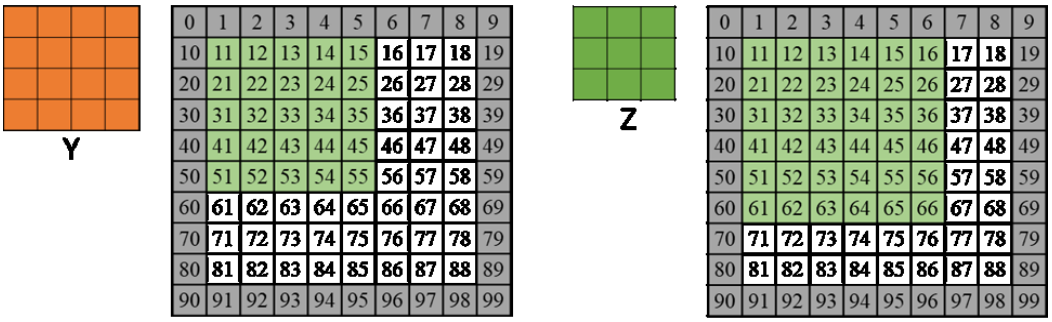
Пусть объект *Y* поставили в клетку 55. Необходимо удалить из домена переменной *Z* номера тех клеток, при постановке в которые объекта *Z* будет пересечение с объектом *Y*.

Согласно формуле 1 рассчитываются клетки, которые занимает объект *Y*:

$Del_1 = \{55, 56, 57, 58, 65, 66, 67, 68, 75, 76, 77, 78, 85, 86, 87, 88\}$.

Новым доменом переменной *Y* будет разность множества домена *Z* и множества Del_1 $D_Y = D_Y \setminus Del_1$. В результате из домена удалятся клетки {55,56,65,66}.

Аналогично, согласно формулам 2–4 рассчитываются недопустимые клетки вокруг *Y*: $Del_2 = \{35, 36, 37, 38, 45, 46, 47, 48\}$; $Del_3 = \{53, 54, 63, 64, 73, 74, 83, 84\}$; $Del_4 = \{33, 34, 43, 44\}$. В результате разности домена переменной *Y* с множествами Del_2, Del_3, Del_4 получится домен $D_Y = \{11, 12, 14, 15, 16, 21, 22, 23, 24, 25, 26, 31, 32, 41, 42, 51, 52, 61, 62\}$ (рис. 4).



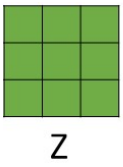
0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	**16**	**17**	**18**	19
20	21	22	23	24	25	**26**	**27**	**28**	29
30	31	32	33	34	35	**36**	**37**	**38**	39
40	41	42	43	44	45	**46**	**47**	**48**	49
50	51	52	53	54	55	**56**	**57**	**58**	59
60	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	69
70	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	79
80	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	89
90	91	92	93	94	95	96	97	98	99

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	**17**	**18**	19
20	21	22	23	24	25	26	**27**	**28**	29
30	31	32	33	34	35	36	**37**	**38**	39
40	41	42	43	44	45	46	**47**	**48**	49
50	51	52	53	54	55	56	**57**	**58**	59
60	61	62	63	64	65	66	**67**	**68**	69
70	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	79
80	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	89
90	91	92	93	94	95	96	97	98	99

Рис. 2. Переменные *Y*, *Z* и их домены

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Рис. 3. Пространство 10 × 10



0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Рис. 4. Домен переменной Z после распространения

Ограничение CornerConstraint

Так как данное ограничение унарное, то оно будет удовлетворено при первоначальном распространении. Для этого для каждого значения из домена переменной проверяется, будет ли объект, поставленный в эту клетку, располагаться в углу. Также можно задавать расположение объекта у конкретного угла (или нескольких углов).

Для каждого из четырех видов углов проверяются две опорные клетки, которые должны быть незанимаемыми. Если хоть одна из них свободна, то, значит, объект не будет располагаться в проверяемом углу.

Для ***верхнего левого*** угла опорными являются клетка выше $p - width_g$ и клетка левее $p - 1$.

Для ***верхнего правого*** угла вначале вычисляется верхняя правая клетка объекта $p_1 = p + width_{obj} - 1$, а после проверяются клетка выше $p_1 - width_g$ и клетка правее $p_1 + 1$.

Для ***нижнего левого*** угла вначале вычисляется нижняя левая клетка объекта $p_2 = p + (height_{obj} - 1) \times width_g$, а затем проверяются клетка ниже $p_2 + width_g$ и клетка левее $p_2 - 1$.

Для ***нижнего правого*** угла вначале вычисляется нижняя правая клетка объекта $p_3 = p + (height_{obj} - 1) \times width_g + width_{obj} - 1$, а после проверяются клетка ниже $p_3 + width_g$ и клетка правее $p_3 + 1$.

Если при проверке всех углов, объект не располагается ни в одном из углов, следовательно, проверяемая клетка p не удовлетворяет ограничению и удаляется из домена.

Пример: есть объект *Y* (см. рис. 2), который необходимо поставить в любой из углов пространства 10 × 10 (см. рис. 3).

При распространении проверяется каждая клетка, но в данном примере показывается алгоритм при рассмотрении клетки 51. Вначале необходимо определить четыре крайние клетки объекта:

верхняя левая клетка $p = 51$;

верхняя правая клетка $p_1 = p + width_{obj} - 1 = 54$;

нижняя левая клетка $p_2 = p + (height_{obj} - 1) \times width_g = 81$;

нижняя правая клетка $p_3 = p + (height_{obj} - 1) \times width_g + width_{obj} - 1 = 84$.

Затем для каждой из крайних точек проверяются две опорные клетки:

для верхней левой клетки $p = 51$: клетки выше ($p - width_g = 41$) и левее ($p - 1 = 50$). Так как клетка 41 является незанимаемой, то объект Y не будет располагаться в верхнем левом углу пространства.

для верхней правой клетки $p_1 = 54$: клетки выше ($p_1 - width_g = 44$) и правее ($p_1+1 =55$). Обе клетки являются свободными, поэтому условие на расположение в верхнем правом углу не выполняется.

для нижней левой клетки $p_2 = 81$: клетки ниже ($p_2 + width_g = 91$) и левее ($p_2 - 1 = 80$). Обе клетки являются незанимаемыми, а значит, выполняется условие на расположение объекта в нижнем левом углу.

для нижней правой клетки $p_3 = 84$: клетки ниже ($p_3 + width_g = 94$) и правее ($p_3 + 1 = 85$). Так как клетка 85 свободна, то условие на расположение объекта в нижнем правом углу не выполняется.

В результате проверки получилось, что если объект Y поставить в клетку 51, то объект будет располагаться в нижнем левом углу, что удовлетворяет ограничению, а значит, значение {51} остается в домене. В случае, когда при проверке значения выявляется, что объект не будет располагаться в каком-то из углов пространства, то проверяемая клетка удаляется из домена.

В результате распространения в домене переменной останутся только те значения, которые удовлетворяют ограничению (рис. 5).

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Рис. 5. Домен переменной Y после распространения

Ограничение DistanceToWallConstraint

Данное ограничение также является унарным и удовлетворяется при первоначальном распространении.

У ограничения два параметра — направление (*direction*) и расстояние (*distance*). Параметр направления (*direction*) может принимать четыре значения: «*North*» — вверх; «*South*» — низ; «*East*» — право; «*West*» — лево. Параметр расстояния (*distance*) измеряется в клетках пространства и может задаваться двумя вариантами: ***жестко*** ограничивать максимальное расстояние до «стены» или ***гибко*** задавать, что расстояние до стены должно быть не меньше заданного.

При создании пространства для каждого из направлений создается матрица, в элементах которой хранится информация о расстоянии от соответствующей клетки до стены. Например, ниже показано графическое представление пространства и матрицы для направления «*South*» (рис. 6).

При распространении рассматривается каждое значение p из домена переменной: вычисляется набор клеток $\{p_k\}$, которые займет объект, если поставить его в проверяемую клетку, по следующей формуле:

$$p + i + j \times width_g, \text{ при } i \in [0, width_1), j \in [0, height_1). \quad (5)$$

Далее для каждого значения $\{p_k\}$ из полученного набора вычисляются координаты x и y. Координатой y является результат целочисленного деления p_i на ширину пространства ($y = p_i \; DIV \; width_g$); а x — это остаток от деления p_i на ширину пространства ($x = p_i \; MOD \; width_g$).

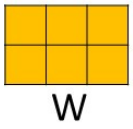
Затем из полученных пар координат из матрицы расстояний вычисляются соответствующие элементы и берется минимальное из них. Если полученное значение не равно заданному, значит, проверяемая клетка p не удовлетворяет ограничению и удаляется из домена.

0	1	2	3	4	5	6	7	8	9
10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	19
20	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	29
30	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	39
40	41	42	43	**44**	**45**	**46**	**47**	**48**	49
50	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	59
60	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	69
70	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	79
80	81	82	83	84	85	86	87	88	89

-1	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**
-1	**2**	**2**	**2**	**6**	**6**	**6**	**6**	**-1**
-1	**1**	**1**	**1**	**5**	**5**	**5**	**5**	**-1**
-1	**0**	**0**	**0**	**4**	**4**	**4**	**4**	**-1**
-1	**-1**	**-1**	**-1**	**3**	**3**	**3**	**3**	**-1**
-1	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**-1**
-1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**-1**
-1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**-1**
-1	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**

Рис. 6. Пространство и матрица расстояний для направления *«South»*

Пример: есть объект *W* (рис. 7) и пространство (см. рис. 6). Задано ограничение, что объект *W* должен располагаться строго на расстоянии 1 от «стены» по направлению «*South*».



0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89

Рис. 7. Переменная *W* и ее домен

При распространении проверяется каждое значение из домена, но в данном примере рассматривается клетка 13.

Вначале необходимо получить список $\{p_i\}$ всех клеток, которые займет объект при постановке его в клетке 13, согласно формуле 5: $p + i + j \times width_g$, при $i \in [0, 3)$, $j \in [0, 2)$. Получится список $p_i = \{13, 14, 15, 23, 24, 25\}$.

Для каждого из значения необходимо рассчитать координаты *x* и *y* и получить по полученным координатам значение d_j из матрицы расстояний для *«South»*:

$$\{13\}: x = 3; y = 1 \rightarrow d_{13} = 2;$$
$$\{14\}: x = 4; y = 1 \rightarrow d_{14} = 6;$$
$$\{15\}: x = 5; y = 1 \rightarrow d_{15} = 6;$$
$$\{23\}: x = 3; y = 2 \rightarrow d_{23} = 1;$$
$$\{24\}: x = 4; y = 2 \rightarrow d_{24} = 5;$$
$$\{25\}: x = 5; y = 2 \rightarrow d_{25} = 5.$$

Из полученных d_j берется минимальное ($d_{23} = 1$) и сравнивается с параметром *distance*. Так как d_{23} не превышает *distance*, то рассматриваемая клетка 13 удовлетворяет ограничению и не удаляется из домена. Если бы параметр *distance* был 0 или 2+, то ограничение не удовлетворялось бы и клетка 13 удалилась бы из домена. После проверки всех значений из домена *W* в домене останутся только те значения, которые удовлетворяют ограничению (рис. 8).

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89

Рис. 8. Домен переменной *W* после распространения

Ограничение NearConstraint

У данного ограничения два параметра: список из пары объектов и максимальное расстояние между ними (*distance*).

После того, как одна из переменных ограничения определила значение (т. е. объект оказался поставлен в пространстве), необходимо для второго объекта выделить область пространства, в которую можно его поставить, соблюдая ограничение на максимальное расстояние. Для области вычисляются координаты четырех углов:

$X_d = p \ MOD \ width_g$;
$Y_d = p \ DIV \ width_g$;
$X_{left} = X_d - width_u - distance$, если $X_{left} \leq 0$, то $X_{left} = 1$;
$X_{right} = X_d + width_d + distance$, если $X_{right} \geq width_g$, то $X_{right} = width_g - width_u$;
$Y_{top} = Y_d - height_u - distance$, если $Y_{top} \leq 0$, *то* $Y_{top} = 1$;
$Y_{bottom} = Y_d + height_d + distance$, если $Y_{bottom} \geq height_g$, то $Y_{bottom} = height_g - height_u$.

Обозначения: $width_g$ — ширина пространства; X_d, Y_d — координаты клетки p, в которую поставили объект; $width_d$, $height_d$ — ширина и высота поставленного объекта; $width_u$, $height_u$ — ширина и высота объекта, для которого строится область.

По полученным координатам строится область p_s из клеток:

$$p_s = y \times width_g + x, \text{ где } x \in [X_{left}, X_{right}], \text{ у } \in [Y_{top}, Y_{bottom}].$$

Полученное множество p_s содержит все клетки, которые удовлетворяют ограничению, и из домена переменной удаляются значения, которых нет в p_s.

Пример: есть переменные *W*, *Y*, *Z (*рис. 9) и пространство 10 × 10 с поставленным объектом *W* (рис. 10). Задано ограничение, что объект *W* должен стоять вплотную к объекту *Y* на расстоянии 0, а *Z* должен стоять на расстоянии не больше одной клетки от объекта *W*. Объект *W* был поставлен в клетку 44.

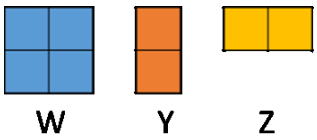


Рис. 9. Объекты *W*, *Y*, *Z*

0	1	2	3	4	5	6	7	8	9
10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	19
20	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	29
30	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	39
40	**41**	**42**	**43**	44	45	**46**	**47**	**48**	49
50	**51**	**52**	**53**	54	55	**56**	**57**	**58**	59
60	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	69
70	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	79
80	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	89
90	91	92	93	94	95	96	97	98	99

Рис. 10. Пространство с поставленным объектом *W*

После постановки объекта *W* необходимо определить допустимую область для объектов *Y*, *Z*.

Область для Y:

Координаты области: $X_d = 44\ MOD\ 10 = 4$; $Y_d = 44\ DIV\ 10 = 4$; $X_{left} = 4 - 1 - 0 = 3$; $X_{right} = 4 + 2 + 0 = 6$; $Y_{top} = 4 - 2 - 0 = 2$; $Y_{bottom} = 4 + 2 + 0 = 6$.

Клетки области $p_s = y \times width_g + x$, где $x \in [3, 6]$, у $\in [2, 6]$, показаны на рисунке ниже (рис. 11).

Область для Z:

Координаты области: $X_d = 44\ MOD\ 10 = 4$; $Y_d = 44\ DIV\ 10 = 4$; $X_{left} = 4 - 2 - 1 = 1$; $X_{right} = 4 + 2 + 1 = 7$; $Y_{top} = 4 - 1 - 1 = 2$; $Y_{bottom} = 4 + 2 + 1 = 7$.

Клетки области $p_s = y \times width_g + x$, где $x \in [1, 7]$, у $\in [2, 7]$, показаны на рисунке выше (рис. 12).

Полученные области становятся доменом переменных и могут содержать значения, которые не удовлетворяют ограничению *GDConstaint*, но они будут удалены при распространении ограничения *GDConstaint*.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Рис. 11. Область допустимых значений для переменной *Y*

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Рис. 12. Область допустимых значений для переменной *Z*

Программная реализация

Процедура решения рассматриваемой в работе задачи удовлетворения ограничений была реализована с применением библиотеки программирования в ограничениях *Choco* на языке программирования *Java*. Реализация включала формализацию переменных, их доменов и системы ограничений средствами используемой библиотеки. Поиск решений задачи выполнялся стандартными для решателя стратегиями поиска с возвратом, а распространение ограничений осуществлялось с применением оригинальных процедур вывода, разработанных в ходе исследований.

При ветвлении в качестве переменной выбиралась переменная с наименьшим размером домена, а значения переменной выбирались по порядку. Для каждого ограничения был реализован свой класс-распространитель.

Ниже показана общая схема работы программы (рис. 13).

Программа имеет пользовательский интерфейс, внутри которого пользователь создает пространство (рис. 14), объекты (рис. 15) и задает для них ограничения (рис. 16). После на основе полученных данных формируется задача в рамках модели *Choco*.

Первым этапом поиска решений является первичное распространение, которое удовлетворяет унарные ограничения, сокращает домены переменных, сужая пространство поиска, и в некоторых случаях позволяет сразу определить, что решений нет.

Если первоначальное распространение не выявило противоречий в задаче, то запускается ветвление. Среди переменных, у которых не определено значение, выбирается та переменная, у которой размер домена минимален. Значения переменной выбираются по порядку.

После ветвления запускается распространение для сокращения доменов неопределенных переменных. В результате распространения проверяется текущее состояние системы. У каждого ограничения есть параметр *ESat*, который принимает одно из трех значений:

ESat.TRUE — если ограничение удовлетворено;

ESat.FALSE — если ограничение не может быть удовлетворено на данной ветви поиска;

ESat.UNDEFINED — если ограничение пока не удовлетворено.

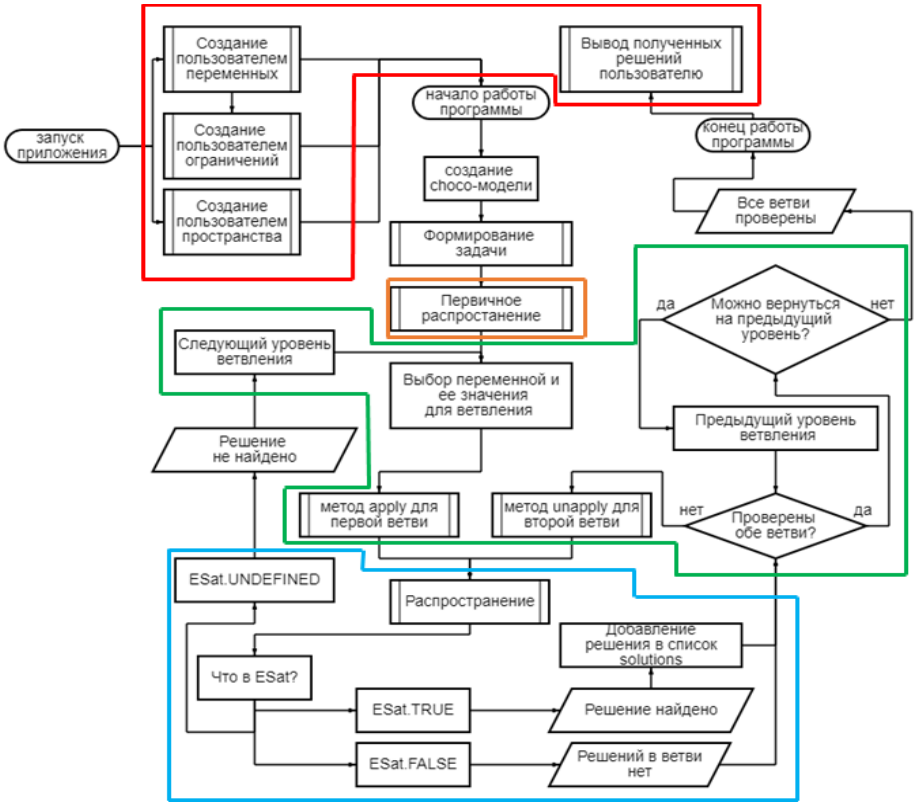


Рис. 13. Общая схема работы программы

Если у всех ограничений *ESat.TRUE*, значит, все объекты расставлены в соответствии со всеми ограничениями и найдено решение. Решение записывается в список и происходит возврат к предыдущему шагу поиска.

Если параметр *ESat* принял значение *UNDEFINED*, то решение пока не найдено и выполняется ветвление.

Если у ограничения в процессе распространения параметр *ESat* становится равен *FALSE*, то в данной ветви решений нет, задача откатывается в предыдущее состояние.

Пользователь может создать пространство (см. рис. 14) двумя способами: ***первый*** — задать ширину и высоту, а программа сама создаст пространство нужной размерности; ***второй*** — вручную описать каждую клетку, задав ей одно из состояний («0» — пустая клетка, «1» — незанимаемая клетка (стена), «2» — клетка, занятая объектом) или считать с файла описанное ранее пространство.

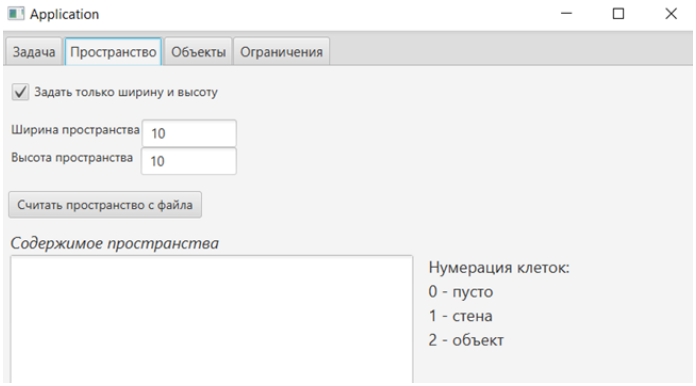


Рис. 14. Пользовательский интерфейс создания пространства

При создании объекта (см. рис. 15) пользователю необходимо задать имя объекта и его содержимое. Под содержимым подразумевается описание каждой клетки объекта его состоянием: «1» — клетка самого объекта; «0» — клетка, которая описывает не сам объект, а клетку, которая должна оставаться свободной для подхода к объекту (например, на рис. 15 показано создание объекта «кровать», для удобного подхода к которому необходимо дополнительно сделать некоторые клетки точно свободными).

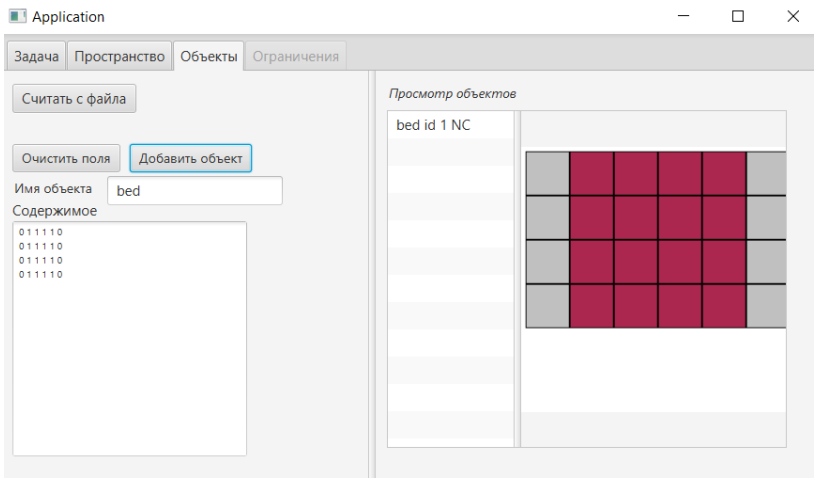


Рис. 15. Пользовательский интерфейс создания объектов

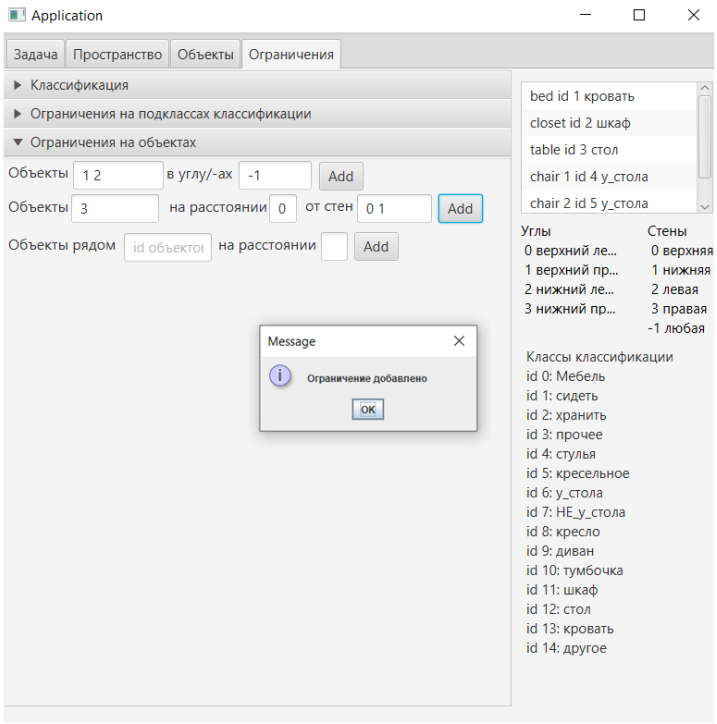


Рис. 16. Пользовательский интерфейс создания ограничений

При создании объектов им будет присвоен уникальный ID, по которому пользователь будет создавать ограничения. Также при создании объектов можно посмотреть их графическое представление. Помимо этого, объекты можно считать с ранее созданного текстового файла.

Для создания ограничений (см. рис. 16) необходимо заполнять текстовые поля:

для ограничения на расположение объекта в углу необходимо задать ID объекта\объектов и код угла\углов, в которых необходимо расположить заданный объект\объекты («0» — верхний левый угол; «1» — верхний правый; «2» — нижний левый; «3» — нижний правый; «-1» — любой);

для ограничения на расположение объекта на заданном расстоянии от заданной стены необходимо задать ID объекта\объектов, расстояние в клетках и код стены («0» — верхняя («*North*»); «1»— нижняя («*South*»); «2» — левая («*West*»); «3» — правая («*East*»); «-1» — любая);

для ограничения на расположение объектов рядом друг с другом необходимо задать ID объектов и максимальное расстояние между ними.

Если выполнение возврата в дереве поиска невозможно, то это означает, что всё дерево поиска пройдено и найдены все решения. После завершения поиска полученные решения выводятся пользователю в интерфейсной части программы (рис. 17).

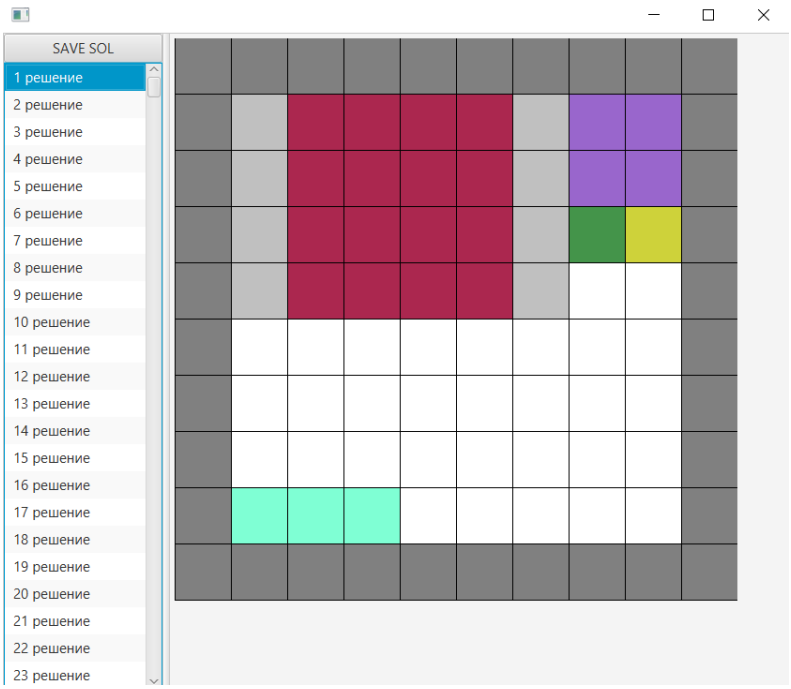


Рис. 17. Пользовательский интерфейс просмотра решений

Ограничения на группы объектов

Также в программе реализована возможность накладывания ограничений не только на отдельные объекты (см. рис. 16), но и на группы объектов (рис. 18).

При создании задачи загружается файл, в котором описывается дерево классификации. Это позволяет при создании объектов отнести их к какому-либо классу и при формировании ограничений задавать ограничения на группу объектов.

Дерево классификации хранится в текстовом файле и загружается пользователем. После загрузки файла в программе появляется графическое представление дерева и пользователю необходимо проклассифицировать объекты, т. е. соотнести с тем или иный классом (см. рис. 18, А). Также каждому классу присваивается свой идентификатор ID, который используется при создании ограничений. Для задания ограничений на подклассах пользователю необходимо в выбранной вкладке заполнить те же поля, что и при создании ограничений для объектов, только вместо ID объекта необходимо указать ID подкласса.

Например, на рис. 18, Б показан процесс создания ограничения, которое можно сформулировать как «все объекты, отнесенные к классу «стул, стоящий у стола» (ID класса 6) должны стоять вплотную (distance = 0) к объекту «стол» (ID объекта 3)».

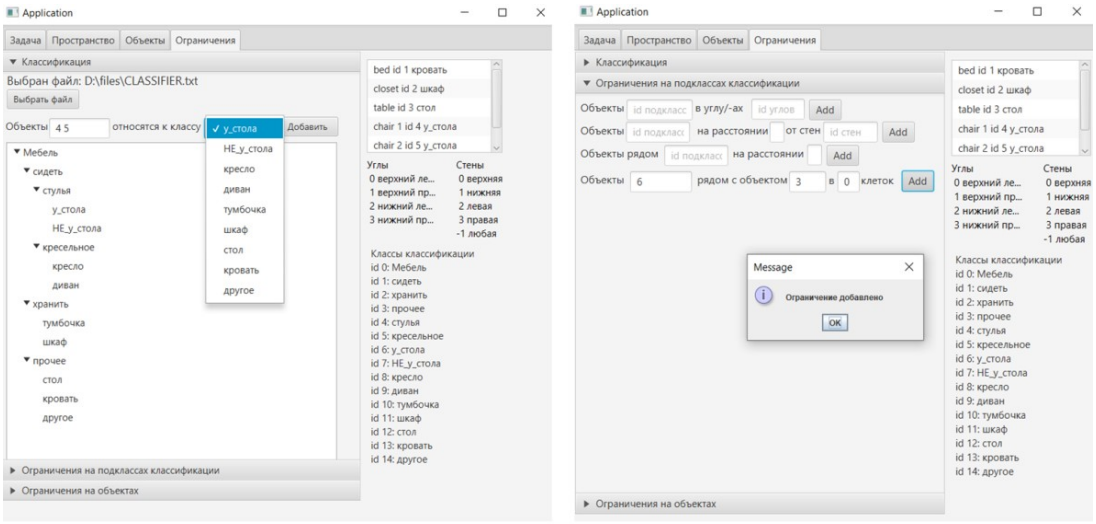


Рис. 18. Пользовательский интерфейс классификации объектов (А) и создания ограничений на подклассах (Б)

Заключение

Парадигма программирования в ограничениях широко применяется для решения задач комбинаторного поиска и комбинаторной оптимизации. Отличительной особенностью данной парадигмы является широкое применение методов рассуждения на ограничениях (методов распространения ограничений).

В работе рассмотрена применимость методов удовлетворения ограничений для решения задач генеративного дизайна. В качестве примера задач рассматриваемого класса в статье подробно разбирается задача проектирования двумерной пространственной среды с учетом требований к взаимному расположению объектов. Подобные требования включают, в частности, запрет на расположение нескольких объектов в одной и той же области двумерной сцены. Также на расположение объекта могут быть наложены ограничения следующих типов: объект должен располагаться в углу; объект должен располагаться на определенном расстоянии до границы пространства; объект должен располагаться на определенном удалении от другого объекта. Соблюдение всех перечисленных ограничений достигается за счет удаления из домена соответствующего объекта клеток пространства, которые не удовлетворяют предъявляемым требованиям. Исключение из доменов недопустимых клеток реализуется разработанными авторами процедурами распространения ограничений. Кроме того, обеспечивается поддержка возможности задавать ограничения не только на отдельные объекты, но и на группы объектов.

Представленные результаты исследований продемонстрировали применимость парадигмы программирования в ограничениях для решения задач генеративного дизайна.

Список источников

1. Krish S. A practical generative design method // Computer-Aided Design. 2011. Vol. 43, № 1. P. 88–100.
2. Mountstephens J., Teo J. Progress and Challenges in Generative Product Design: A Review of Systems // Computers. 2020. Vol. 9. P. 80.
3. Метелик Т. С. Генеративный метод проектирования и способы его реализации в графическом дизайне // Бизнес и дизайн ревью. 2017. Т. 1, № 2 (6). 11 с.
4. Jaisawal R., Agrawal V. Generative Design Method (GDM)—A State of Art // Proceedings of the ICOTRIME 2020 IOP Conf. Series: Materials Science and Engineering. 2021. Vol. 1104, № 012036.
5. Gu N., Behbahani A. P. Shape Grammars: A Key Generative Design Algorithm. Handbook of the Mathematics of the Arts and Sciences. 2021. P. 1385–1405.
6. McCormack J. P., Dorin A. Innocent T. C. Generative design: a paradigm for design research // Futureground. Monash University. 2005. Vol. 2.
7. Lindenmayer A., Rozenberg G. Developmental Systems and languages // Proceedings of the Fourth Annual ACM Symposium on Theory of Computing. New York, NY, USA, Springer International Publishing. Cham, Switzerland, 2012. P. 214–221.

8. Herr C. M., Ford R. C. Cellular automata in architectural design: From generic systems to specific design tools // Automation in Construction. 2016. Vol. 72, Part 1. P. 39–45.
9. Gero J. S., Kazakov V. A. Genetic engineering approach to genetic algorithms // Evol. Comput. 2001. № 9. P. 71–92.
10. Kennedy J., Eberhart R. C., Shi Y. Swarm Intelligence. Elsevier. Amsterdam, The Netherlands, 2001.
11. Родин И. С., Нестерова А. В., Кожевяткина М. А., Кечин Е. С. Применение глубокого обучения в генеративном дизайне: анализ и оптимизация алгоритмов // Наука молодых: Сборник статей по материалам XVI Всероссийской научно-практической конференции. Арзамас; Нижний Новгород: Нижегородский государственный технический университет им. Р. Е. Алексеева, 2024. С. 235–240.
12. Chao Q., Ren T., Wenjing Y. An adaptive artificial neural network-based generative design method for layout designs // International Journal of Heat and Mass Transfer. 2022. P. 1–31.
13. Poole D., Mackworth A. Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press, 2023. 903 p.

References

1. Krish S. A practical generative design method. *Computer-Aided Design*, 2011, vol. 43, no 1, pp. 88–100.
2. Mountstephens J., Teo J. Progress and Challenges in Generative Product Design: A Review of Systems. *Computers*, 2020, vol. 9, p. 80.
3. Metelik T. S. Generativnyy metod proyektirovaniya i sposoby yego realizatsii v graficheskom dizayne [Generative design method and ways of its implementation in graphic design]. *Biznes i dizayn rev'yu* [Business and design review], 2017, vol. 1, no 2 (6), 11 p. (In Russ.).
4. Jaisawal R., Agrawal V. Generative Design Method (GDM)—A State of Art. *Proceedings of the ICOTRIME 2020 IOP Conf. Series: Materials Science and Engineering*, 2021, vol. 1104, no 012036.
5. Gu N., Behbahani A. P. *Shape Grammars: A Key Generative Design Algorithm*. Handbook of the Mathematics of the Arts and Sciences, 2021, pp. 1385–1405.
6. McCormack J. P., Dorin A. Innocent T. C. Generative design: a paradigm for design research. *Futureground*. Monash University, 2005, vol. 2.
7. Lindenmayer A., Rozenberg G. Developmental Systems and languages. *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*. New York, NY, USA, Springer International Publishing. Cham, Switzerland, 2012, pp. 214–221.
8. Herr C. M., Ford R. C. Cellular automata in architectural design: From generic systems to specific design tools. *Automation in Construction*, 2016, vol. 72, Part 1, pp. 39–45.
9. Gero J. S., Kazakov V. A. Genetic engineering approach to genetic algorithms. Evol. *Comput*., 2001, no 9, pp. 71–92.
10. Kennedy J., Eberhart R. C., Shi Y. *Swarm Intelligence*. Elsevier. Amsterdam, The Netherlands, 2001.
11. Rodin I. S., Nesterova A. V., Kozhevyatkina M. A., Kechin Ye. S. Primeneniye glubokogo obucheniya v generativnom dizayne: analiz i optimizatsiya algoritmov [Application of deep learning in generative design: analysis and optimization of algorithms]. *Nauka molodykh: Sbornik statey po materialam XVI Vserossiyskoy nauchno-prakticheskoy konferentsii* [Science of the young: Collection of articles based on the materials of the XVI All-Russian scientific and practical conference]. Arzamas, Nizhniy Novgorod, Nizhegorodskiy gosudarstvennyy tekhnicheskiy universitet im. R. Ye. Alekseyeva [Arzamas, Nizhny Novgorod, Nizhny Novgorod State Technical University named after R. E. Alekseev], 2024, pp. 235–240. (In Russ.).
12. Chao Q., Ren T., Wenjing Y. An adaptive artificial neural network-based generative design method for layout designs. *International Journal of Heat and Mass Transfer*, 2022, pp. 1–31.
13. Poole D., Mackworth A. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, 2023, 903 p.

Информация об авторах

П. В. Таран — стажер-исследователь;
А. А. Зуенко — кандидат технических наук, ведущий научный сотрудник.

Information about the authors

P. V. Taran — Intern Researcher;
A. A. Zuenko — Candidate of Science (Tech.), Leading Researcher.

Статья поступила в редакцию 01.11.2025; одобрена после рецензирования 24.11.2025; принята к публикации 28.11.2025.
The article was submitted 01.11.2025; approved after reviewing 24.11.2025; accepted for publication 28.11.2025.