ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте на тему:

Алгоритмы на графах и автоматические решатели

Выполнил студент:

группы #БПМИ233, 2 курса Хорошилов Андрей Денисович

Принял руководитель проекта:

Трушин Антон Николаевч
Преподаватель
Факультет компьютерных наук НИУ ВШЭ

Москва 2025

Содержание

Аннотация

В ходе работы над проектом были изучены статьи, описывающие концепцию проектирования и автоматизации создания пользовательского интерфейса программ Property model, которая позволяет заметно сократить время написания кода взаимодействия архитектурных элементов и вероятность возникновение ошибок. Графический интерфейс в данной модели интерпретируется как набор переменных и наложенных на них ограничений, поэтому одним из основных компонентов Property model обязан является алгоритм решателя ограничений. В рамках проекта был рассмотрен алгоритм DeltaBlue, выведено полное доказательство его корректности и реализована библиотека для языка программирования C++, реализующая Property model на основе данного алгоритма.

Ключевые слова

Property model, пользовательский интерфейс, переиспользование ПО, системы ограничений, DeltaBlue, SkyBlue, QuickPlan.

1 Введение

Зачастую создание пользовательского интерфейса программы занимает значительную часть работы над приложением. Причиной этому служит большое количество переменных и взаимосвязей между ними, которые необходимо поддерживать для корректного функционирования приложения. Преобладание одноразовых архитектурных решений в вопросе синтеза командных параметров делает код громоздким и слабо переиспользуемым, помимо этого заметной проблемой является высокая вероятность возникновения ошибок и сложность их исправления. Появляется естественное желание автоматизировать процесс написания подобных интерфейсов. Sean Parent, старший научный сотрудник и архитектор программного обеспечения в компании Adobe, в своих статьях [1, 2, 3] разрабатывает концепцию Property model, как раз способную помочь в решении проблемы.

В рамках данной работы изучены теоретическая аспекты функционирования Property Model, рассмотрены алгоритмы автоматических решателей, необходимых для реализации модели, изучены приведенные в статьях доказательства корректности работы алгоритма DeltaBlue, исправлены ошибки и приведено полное корректное доказательство, а также разработана библиотека для языка программирования C++, реализующая Property model, внутри которой заложен алгоритм решения ограничений DeltaBlue.

Задачами данной работы являются:

1 Изучение статей Sean Parent-a о концепции Property Model;

2 Изучение алгоритмов автоматических решателей ограничений по статьям;

3 Доказательство корректности алгоритмы, который будет положен в основу имплементации Property Model в библиотеке, а именно DeltaBlue;

4 Полное описание доказательства корректности алгоритма;

5 Выбор архитектуры;

6 Имплементация фреймворка для генерации Property Model по заданному описанию;

7 Написание тестов для проверки работоспособности библиотеки;

8 Написание сопроводительной документации.

Исходный код проекта находится в репозитории по ссылке.

2 Изучение аналогов

Проведенный анализ показал, что на сегодняшний день существует несколько реализаций библиотек для работы с Property Model, но большинство из них либо являются встроенными частями более крупных проектов и экосистем, либо находятся в закрытом доступе.

Отдельного рассмотрения заслуживает разработанные компанией Adobe System в рамках своих графических редакторов и выложенные в 2021 году в открытый доступ на github [4] библиотеки специализированных языков Adam и Eve. Первая из них нацелена декларативное создание Property Model, через явное описание свойств и зависимостей системы, вторая - позволяет декларативно задавать вид диалоговых окон с автоматической привязкой к использующейся Property Model и осуществлять реактивное изменение параметров системы.

Данные библиотеки берут за основу алгоритм решателя ограничений QuickPlan [5], однако явное разделения кода на модель и алгоритм отсутствует.

В остальном, за время поиска не было найдено других аналогов библиотек для работы с Property Model, в особенности на языке программирования C++.

3 Теоретическая база

3.1 Property Model

Основой Property model является явный подход к заданию переменных и их зависимостей: вместо того, чтобы прописать функции для пересчета всех параметров интерфейса после изменения одного конкретного из них и скрипты, которые будут вызывать эти функции и следить за предсказуемостью поведения системы, рассматриваемая модель позволяет задать каждый параметр интерфейса, например, значение полей, положение курсора на экране, выбор варианта во всплывающем списке и т.п., как переменную внутри Property Model, а затем, явно передав капсулирование функции для пересчета и удовлетворения инвариантов между этими переменными, осуществлять интерактивное взаимодействие - менять значения переменных, включать и выключать ограничения, автоматически эффективно пересчитывать все параметры системы.

Таким образом, Property Model использует схожесть поведения большинства типичных пользовательских интерфейсов, таких как например, диалоговые окна, и интерпретирует их как системы ограничений, о которых подробнее мы поговорим далее.

Основа любой Property model - алгоритм для вычисления новых значений элементов

пользовательского интерфейса после изменения значений других элементов - алгоритмы решения ограничений. Глобально все алгоритмы можно разделить на группы в зависимости от того, какие системы ограничений они решают и какие методы используют. Основными требованиями к алгоритму решения ограничений для нашей системы автоматизированного проектирования были скорость пересчета, гибкость в разработке и реализации. Как наиболее известный вариант, на который ссылается в своих статья Sean Parent, были выбраны алгоритмы локального распространения. Они представляют системы ограничений в виде неориентированного двудольного графа, где переменные и ограничения являются вершина, и ребро между переменного и вершиной проводится, если вершина участвует в ограничении. Из сильных сторон таких алгоритмов можно выделить эффективность и отсутствие ограничений на сферу применения.

Конкретнее алгоритмы локального распространения можно разделить на типы в соответствии с:

1 Поддержкой одно- или много-направленных ограничений (допустимы один или несколько методов);

2 Поддержкой методов с одним или множественным выходом;

3 Поддержкой потенциально циклический систем (то есть систем ограничений, где в теории попытка удовлетворить ограничения может привести к зацикливанию операций);

4 Инкриментное обновление или нет;

5 Поддержкой иерархий ограничений.

В задачах проекта стояло в первую очередь рассмотреть алгоритм DeltaBlue [6, 7], который относится к инкриментным алгоритмам локального распространения, поддерживает иерархии ограничений и много-направленные ограничения, но не работает с методами с множественным выходом и потенциально циклическими ограничениями.

3.2 Общие понятия

Определение 1. Системой ограничений назвывается набор $\langle C, V \rangle$, где

1 V - множество переменных.

2 C - множество ограничений. Любое ограничение $c \in C$ представляет собой набор $\langle R, r, M \rangle$, где

- $R \subseteq V$
- r - некоторое n-арное отношение между переменными из R ($n = |R|$)
- M - множество методов решателей ограничения (далее метод). Каждый метод ограничения $m \in M$ принимает на вход $m.\text{in} \subseteq R$ и возвращает значения для переменных из $m.\text{out} \subseteq R$. Исполнение любого метода должно обеспечивать удовлетворение отношения r.

Систему ограничений можно интерпретировать как граф.

Определение 2. Пусть нам дана система ограничений $\langle C, V \rangle$. **Граф ограничений** $G_C = \langle V + C, E \rangle$ (от Constraint Graph) - двудольный граф, где вершинами одной доли являются все элементы из V, вершинами из другой доли все элементы из C. Неориентированное ребро $e \in E$ между $v \in V$ и $c \in C$, где $c = \langle R, r, M \rangle$ проведено тогда и только тогда $v \in R$.

Определение 3. Приоритет ограничения - число из $\mathbb{Z}_{\geqslant 0}$, где 0 - требуемое ограничение; 1 - самые приоритетные среди не требуемых и так далее.
Обозначение: $Pr(c)$, где $c \in C$.

Далее будем обозначать все ограничения из C с приоритетом k, как C_k.

Определение 4. Система ограничений, в которой каждой каждому ограничению присвоен приоритет, называется **иерархией ограничений**.

Далее по умолчанию будем считать, что мы работаем с иерархиями.

Определение 5. Пусть мы имеем граф ограничений $G_C = \langle V + C, E \rangle$. Тогда будем называть **графом решений** $G_S = \langle V + C_S, E_S \rangle$ (от Solution Graph) граф, который удовлетворяет следующим свойствам:

1 G_S - ациклический;

2 $\forall c \in C_S$ представляет собой ограничение из $\langle R, r, M \rangle \in C$, в котором выбрали ровно один метод $m \in M$ (будем обозначать его $c.\text{method}$). Отображение $C_S \longrightarrow C$, сопоставляющее ограничению $c \in G_S$ ограничение $c' \in C$, из которого c было получено выбором метода, инъективно, то есть возможно, что $|C_S| < |C|$;

3 $E_S = \{\forall c \in C_S,\ \forall v \in c.\text{method.in} \mid (v, c)\} \cup \{\forall c \in C_S,\ \forall v \in c.\text{method.out} \mid (c, v)\}$

4 $\forall v \in V : |v.\text{in}| \leqslant 1$ (здесь v.in - множество удовлетворенных ограничений, чьи выходные переменные выбранных методов содержат v);

5 в C_S присутствуют все ограничения из C_0;

Замечание 6. Если мы отождествим C_S с подмножеством C, куда оно инъективно вкладывается описанным в пункте 3 отображением, то получим, что $C_S \subseteq C$. Также учтём, что $E_S \subseteq E$. Из этого следует, что граф решений является двудольным как подграф двудольного графа.

Соответственно решением системы ограничений далее мы будем называть выбор методов в системе ограничений, который порождает корректный граф решения.

Определение 7. Пусть дано решение системы ограничений $G_C = \langle V + C, E \rangle$, которое соответствует графу $G_S = \langle V + C_S, E_S \rangle$. Тогда если для $c \in C$ существует такое $c' \in C_S$, что отображение из пункта 3 предыдущего определения переводит c' в c, то c называется **удовлетворенным**.

Определение 8. Пусть нам дан граф решения $G_S = \langle V + C_S, E_S \rangle$. $v \in V$ **определяется** ограничением $c \in C_S$, если c.method.out $= v$

Возникает вопрос: как из всех возможных решений выбрать наиболее предпочтительное? Для этого введем частичное отношение порядка на множестве решений **locally-predicate-better**:

Определение 9. Пусть у нас есть два решения A и B. Тогда $A > B$, если $\exists k \in \mathbb{Z}_{\geqslant 0}$ такой, что

1 $\forall n \in \mathbb{Z}_{\geqslant 0}, n < k$, $C_n^A = C_n^B$;

2 $C_k^A \subseteq C_k^B$;

где C_n^A, C_n^B - множества удовлетворенных ограничений с приоритетом n в решении A и B соотвтетвенно.

Далее иногда будем обозначать locally-predicate-better как LPB.

Замечание 10. locally-predicate-better не является линейным отношением порядка.

Доказательство. Приведем пример (см. схему на рисунке 3.1). Пусть у нас есть система ограничений, состоящая из двух ограничений c_0 и c_1 оба приоритета 0, каждое из которых имеет по одному методы, пишущему соответственно в v_0 и v_1. Существует два корректных

решения G_{s1} и G_{s2} (см. Рис. 3.1), которые несравнимы между собой, так как на минимальном уровне приоритета ограничений - нулевом - ни одно из множества удовлетворенных ограничений G_{s1} и G_{s2} не вложено в другое. Из этого можно сделать вывод, что не всегда существует единственное наибольшее решение по LPB, несмотря на это апеллировать максимальными решениями можно. □

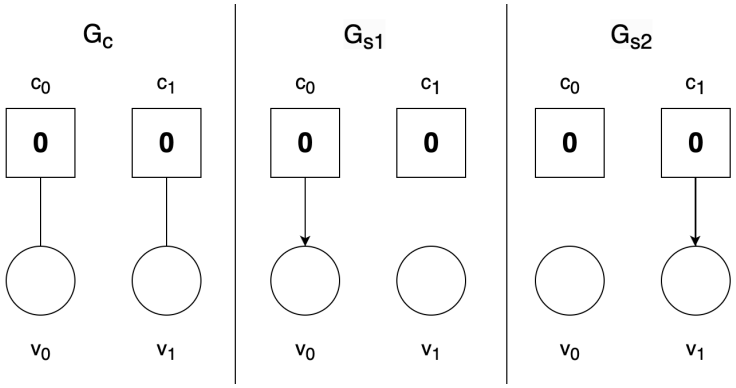

Рис. 3.1: Пример несравнимых решений

Наконец сформулируем математическую задачу:

Формулировка задачи: Для данной системы ограничений $\langle C, V \rangle$ мы хотим найти максимальное в терминах locally-predicate-better решение.

Дадим еще несколько вспомогательных определений:

Определение 11. Система ограничений называется:

1 **недоограниченной**, если существует несколько решений, в которых все ограничения удовлетворены;

2 **переограниченной**, если существует решения, которое удовлетворяет все ограничения.

3.3 Системы ограничений как часть Property Model

Давайте скажем пару слов о том, как Property Model использует системы ограничений и какие особенности накладывает на них.

Как уже было сказано ранее Property Model интерпретирует все параметры интерфейса как переменные системы ограничений, а инварианты и взаимосвязь между ними как набор ограничений, где методами являются функции пересчета параметров, заданные пользователем модели. В действительности мы ожидаем от модели предсказуемого поведения, что может проявляться в нескольких аспектах.

Во-первых, решение системы ограничений мы хотим интерпретировать как план пересчета параметров, то есть при каждом взаимодействии с моделью мы хотим получить граф решения, а дальше «развернуть» его в последовательность методов и поочередно их выполнить, при этом не нарушая инвариант о том, что метод определяющий новое значение переменной должен быть выполнен ранее, чем любой метод, использующий эту переменную как входную. Это действительно возможно, поскольку любой граф решения является ациклическим, следовательно, к нему можно применить топологическую сортировку и получить искомую последовательность методов.

Во-вторых, для более предсказуемого поведения нам бы не хотелось иметь недоограниченную систему ограничений. Это достигается с помощью добавления в систему stay-ограничений:

Определение 12. stay-ограничение - ограничение, привязанное к определенной переменной $v \in V$ и зависящее от его текущего значения (обозначим за v.value). Для переменной $v \in V$ stay задается следующим образом:

- $R = \{v\}$;
- $r = \{v = v.\text{value}\}$, то есть единственное отношение говорит v оставаться неизменной;
- $M = \{v := v.\text{value}\}$

Можно считать число приоритет stay-ограничения лежит в другом спектре и априори слабее (в численном отношении - больше) любого приоритета, не относящегося к stay.

Утверждение 13. *LPB решение системы ограничений $\langle C, V\rangle$ с включенными stay определяет все переменные системы, то есть для каждой переменной $v \in V$ существует такое $c \in C_S$, что c.mathod.out $= v$.*

Доказательство. Предположим, что это не так и переменная $v \in V$ не определяется ни одним методом решения G_S. Тогда stay-ограничение переменной v точно не удовлетворено. Добавим это ограничение в граф решения, выбрав единственный метод из его набора методов. Заметим, что решение всё ещё корректно: граф остался ациклическим, так как у метода stay-ограничения нет входных переменных, то есть он не сможет стать частью цикла; входная степень любой вершины осталась не больше 1, так как с добавлением stay в G_S степень изменилась только у переменной v и стала равна 1. Получившееся решение строго содержит в себе множество удовлетворенных ограничений исходного решения, и, следовательно, лучше по LPB. Таким образом, мы пришли к противоречию. □

Данное замечание позволяет считать, что в контексте систем ограничений для Property Model можно уточнить 4 пункт из определения графа решения и полагать, что $\forall v \in V : |v.\text{in} = 1|$.

Развивая концепцию stay-ограничений, мы можем реализовать еще одну идею, обеспечивающую предсказуемость работы модели: принцип *«least surprise»* гласит, что при прочих равных предпочтительнее тот исход взаимодействия с моделью, который не изменяет значения переменных, которые менялись недавно. Для этого в спектре приоритета stay-ограничений мы так же введем иерархию и будем говорить, что при каждом изменении значения переменной пользователем её stay-ограничение становится самым приоритетным. Будем считать, что по умолчанию приоритеты всех stay-ограничений различны и линейно упорядочены.

Дополнительно отметим, что в одной из статей Sean Parent-a [2] приводятся правила построения систем ограничений для Property Model. Сформулируем их:

1 $\forall \langle R, r, M \rangle \in C, \forall m \in M \ m.\text{out} \sqcup m.\text{in} = R$;

2 $\forall m$ метод, возникающий в любом из ограничений системы ограничений $m.\text{out} \neq \varnothing$;

3 $\forall c_1, c_2 \in C : c_1 \neq c_2 \implies R_1 \neq R_2$, где $c_1 = \langle R_1, r_1, M_1 \rangle, c_2 = \langle R_2, r_2, M_2 \rangle \in C$;

4 $\forall \langle R, r, M \rangle \in C, \forall m_1, m_2 \in M : m_1 \neq m_2 \implies m_1.\text{out} \nsubseteq m_2.\text{out}$

Первое правило обеспечивает возможность решения системы ограничения за полиномиальное время [8]. Будем называть его правилом о входах и выходах метода. Методы, нарушающие второе правило, не нужны, так как они никогда не влияют на пересчет переменных системы. Ограничения, не удовлетворяющие третьему правилу можно объединить в одно. Четвертое правило убирает ситуации, когда нельзя определить, что один метод предпочтительнее другого, что гарантирует уникальность и предсказуемость работы модели. Далее будем считать, что эти правила выполняются.

3.4 Вспомогательные определения для DeltaBlue

Перейдем к алгоритму DeltaBlue и доказательству его корректности. Напомним, что данный алгоритм поддерживает только методы с единственным выходом и не способен обрабатывать потенциально циклические решения (в случае нахождения цикла считаем, что алгоритм завершается с сообщением о некорректности переданной системы ограничений).

Определение 14. Будем называть **потенциальными выходами** ограничения $c \in C, c = \langle R, r, M \rangle$ множество:

$$c.\text{pout} = \bigcup_{m \in M} m.\text{out}$$

множество переменных, которые являются выходами одного из методов ограничения.

Определение 15. Пусть нам дан граф решения $G_S = \langle V + C_S, E_S \rangle$. Тогда для любой переменной $\forall v \in V$, которая определяется ограничением c определим **силу** следующим образом:

$$F(v) = \max(\Pr(c), \max_{v' \in c.\text{pout} \setminus \{v\}} F(v'))$$

Утверждение 16. *Определение силы переменной корректно.*

Доказательство. В силу правила о входах и выходах метода, мы имеем, что для любой переменной $v \in V$ и ограничения c, которое ее определяет, верно, что $c.\text{pout} \setminus \{v\} \subseteq c.\text{mathod.in}$. Следовательно, сила конкретно выбранной переменной зависит только силы переменных, лежащих «выше» в графе G_S. Граф решения по определению ациклический, следовательно на нем можно осуществить топологическую сортировку. Рассматривая вершины в порядке топологической сортировки, мы получим, что все силы переменных, лежащих «выше» в графе G_S уже посчитаны. Значит, определение силы переменной корректно. □

Из определения видно, что переменные, определенные stay-огрничением имеют силу равную приоритету stay-ограничения.

3.4.1 Лемма о заблокированном ограничении

Лемма 17 (О универсальности сил переменных)**.** *A, B - решения системы ограничений $\langle C, V \rangle$ такие, что $\exists n \in \mathbb{Z}_{\geqslant 0}\ \forall k \in \mathbb{Z}_{\geqslant 0} : \ k \leqslant n \implies C_k^A = C_k^B$. Тогда $\forall v \in V$ либо $F_A(v) = F_B(v)$, либо одновременно $F_A(v) > n$ и $F_B(v) > n$.*

Доказательство. Пусть $k \leqslant n$. Будем считать, что G_A - граф решения A, G_B - граф решения B, $V_{\leqslant k}^A$ и $V_{\leqslant k}^B$ - множества вершин с силой $\leqslant k$ в графах G_A и G_B соответственно (см. схему на рисунке 3.2).

Пусть C_A - множество ограничений, определяющих переменные из $V_{\leqslant k}^A$. По определению силы переменной получаем, что $\forall c \in C_A \implies \forall v \in c.\text{pout}\ F_A(v) \leqslant k$ и $Pr(c) \leqslant k$, в противном случае $F(c.\text{method.out}) > k$. Первое следствие можно интерпретировать как $\bigcup_{c \in C_A} c.\text{pout} \subseteq V_{\leqslant k}^A$. Также отметим, что так как из каждого метода выходит ребро в ровно одну переменную, и каждую переменную определяет ровно одно ограничение, то $|C_A| = |V_{\leqslant k}^A|$.

В силу совпадения удовлетворенных ограничений на уровнях до n-го включительно имеем, что $\forall c \in C_A$ c удовлетворено и в решении B. Тогда рассмотрим C_B - подмножество ограничений G_B, соответствующие ограничениям C_A в графе G_A. Так как по сути своей C_A и C_B это одни и те же ограничения в исходной системе ограничений, то множество переменных, удовлетворенных ограничениями C_B вложено в $\bigcup_{c \in C_A} c.\text{pout}$, и, следовательно, вложено в $V^A_{\leqslant k}$. Но $|C_B| = |C_A| = |V^A_{\leqslant k}|$, значит, множество переменные, которые определяются ограничениями C_B в точности равно $V^A_{\leqslant k}$.

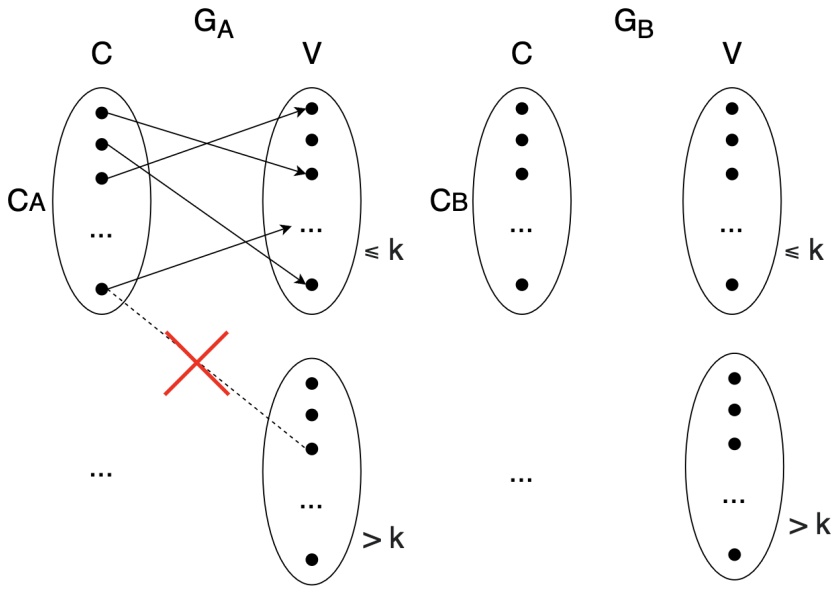

Рис. 3.2: Лемма о универсальности сил переменных

Далее заметим, что сила любой переменной из $V^A_{\leqslant k}$ в графе решения G_B так же $\leqslant k$. Действительно, для любой переменной $v \in V^A_{\leqslant k}$ в графе G_B «выше» нее могут лежать только переменные из $V^A_{\leqslant k}$ в силу того, что $\bigcup_{c \in C_B} c.\text{pout} \subseteq V^A_{\leqslant k}$. Каждую переменную $V^A_{\leqslant k}$ определяет ограничение силы не больше k, значит, сила любой переменной $V^A_{\leqslant k}$ не может превышать k.

Таким образом, мы получили, что $V^A_{\leqslant k} \subseteq V^B_{\leqslant k}$. Из симметричности получаем, что $V^A_{\leqslant k} = V^B_{\leqslant k}$. Так как данное утверждение верно для любого $k \leqslant n$, то верно, что $\forall k \leqslant n$ $V^A_{=k} = V^B_{=k}$ ($V^A_{=k} = V^A_{\leqslant k} \backslash V^A_{\leqslant k-1}$, для B аналогично).

□

Определение 18. Пусть нам дана система ограничений $\langle C, V \rangle$ и граф решения G_S. Будем называть ограничение c **заблокированным**, если оно не удовлетворено в G_S и сила одного из его потенциальных выходов численно больше, чем $Pr(c)$.

Лемма 19 (О заблокированном ограничении)**.** *Пусть нам дана система ограничений* $\langle C, V\rangle$ *и граф её решения* G_S*. Если в системе нет заблокированных ограничений, то данное решение максимальное по LPB.*

Доказательство. Предположим, что это не так и существует граф решения G'_S лучше по LPB. Тогда найдется такое $k \in \mathbb{Z}_{\geqslant 0}$, что на уровнях $0, \ldots, k-1$ множества удовлетворенных ограничений обоих решений совпадают, найдется непустое множество ограничений $\{c_1, c_2, \ldots, c_t \in C\}$ таких, что $\forall i \in \{1, \ldots, t\}$ c_i удовлетворено в графе G_S, но не удовлетворено в G'_S. Пусть c_i определяет в G'_S переменную v_i.

Для каждого $i \in \{1, \ldots, t\}$ добавим в исходную систему ограничений $\langle C, V\rangle$ дополнительное ограничение $s_i = \langle\{v_i\}, \{v_i = v_i.\text{value}\}, \{v_i := v_i.\text{value}\}\rangle$ с приоритетом $k+1$ (ограничение совпадает со stay-ограничением переменной v во всем, кроме приоритета). Обозначим получившуюся систему $\langle C', V\rangle$.

Рассмотрим два решения системы $\langle C', V\rangle$. Граф первого будет в точности совпадать с G_S. Это всё ещё корректное решение, так как все условия определения выполняются. Отметим, что численные значения сил переменных сохранятся. Второе решение построим следующим образом: возьмем граф G'_S, удалим из него все ограничения $c_i, i \in \{1, \ldots, t\}$ и добавим ограничения $s_i, i \in \{1, \ldots, t\}$, выбрав в каждом единственный доступный метод. Обозначим новый граф G''_S и заметим, что он порождает корректное решение: входная степень каждой вершины осталась равна 1, так как наши изменения затронули только входные ребра вершин $v_i, i \in \{1, \ldots, t\}$, для каждой из них мы обновили определяемое ограничение; циклы возникнуть не могли, так как входная степень выбранных методов ограничений $s_i, i \in \{1, \ldots, t\}$ равна 0.

Так как в графе G_S не было заблокированных ограничений, то все потенциальные выходы методов $c_i, i \in \{1, \ldots, t\}$, в том числе и переменные $v_i, i \in \{1, \ldots, t\}$ имеют силу $\leqslant k$. Графы решений G_S и G''_S удовлетворяют одинаковый набор ограничений с приоритетами $\leqslant k$, следовательно, по лемме о универсальности сил переменных получаем, что $\forall i \in \{1, \ldots, t\}$ $F(v_i)$ в графе G''_S так же $\leqslant k$. Возникло противоречие, поскольку каждая из этих переменных определяется ограничением с силой $k+1$, то есть по определению силы переменной $\forall i \in \{1, \ldots, t\}$ $F(v_i) \geqslant k+1$.

Значит, решения лучше чем G_S по LPB не существует. □

3.4.2 Лемма о обратимом пути

Определение 20. Пусть дан граф решений $G_S = \langle V + C_S, E_S \rangle$. Рассмотрим последовательность $c_0, v_1, c_1, \ldots, v_n, c_n$, где $c_0, \ldots, c_n \in C_S,\ v_1, \ldots, v_n \in V$, и переменную $v \in V$ которые удовлетворяет следующим условиям:

1 переменная v определяется ограничением c_0;

2 $\forall k \in \{1, 2, \ldots, n\}$ ограничение c_k определяет переменную v_k, которая;

3 $\forall k \in \{1, 2, \ldots, n\}$ переменная v_k является потенциальным выходом ограничения c_{k-1}.

Тогда такой путь будем называть *обратимым* от ограничения c_n до переменной v.

Лемма 21 (Лемма об обратимом пути)**.** *Пусть дан граф решений G_S. Тогда для любой вершины $v \in V$ существует обратимый путь от вершины v до ограничения $c_n \in C_S$ такой, что $Pr(c_n) = F(v)$. Причём сила любой переменные данного пути будет равна $F(v)$ и $\forall i, j \in \{1, \ldots, n\}\ Pr(c_i) \leqslant F(v_j)$.*

Доказательство. Рассмотрим ограничение c_0, определяющее переменную v. Если её приоритет равен $F(v)$, то обратимый путь найден и состоит из одного ограничения. Если нет, то по определению силы переменной среди $c_0.\text{pout}\backslash\{v\}$ существует вершина v_1 с силой $F(v)$. Повторим рассуждения для вершины v_1 и определяющего его ограничения c_1 и так далее. Так как граф решения ациклический и все потенциальные выходы ограничения в являются входными в графе G_S по правилу о входах и выходах метода, то все рассматриваемые ограничения и переменные на каждой итерации будут уникальны. В силу конечности графа получим, что когда-то алгоритм завершиться, так как приоритет рассматриваемого ограничения c_n будет равен $F(v)$. $c_0, v_1, c_1, \ldots, v_n, c_n$ - искомый обратимый путь для вершины v до ограничения c_n (см. пример построения обратимого пусти на рисунке 3.3.).

Из рассуждения видно, что силы всех переменных пути равны $F(v)$, откуда явно следует неравенство из условия. □

3.5 DeltaBlue

DeltaBlue - инкрементный алгоритм решения ограничений. Постулируем особенности его работы:

- имеем обновляющиеся графы G_C, G_S, начальное состояние G_C: V - фиксированное множество переменных системы, $C = \{\text{stay-оганичение для } \forall v \in V\}$; начальное состояние G_S - граф решения, полученный удовлетворением всех stay-ограничений;

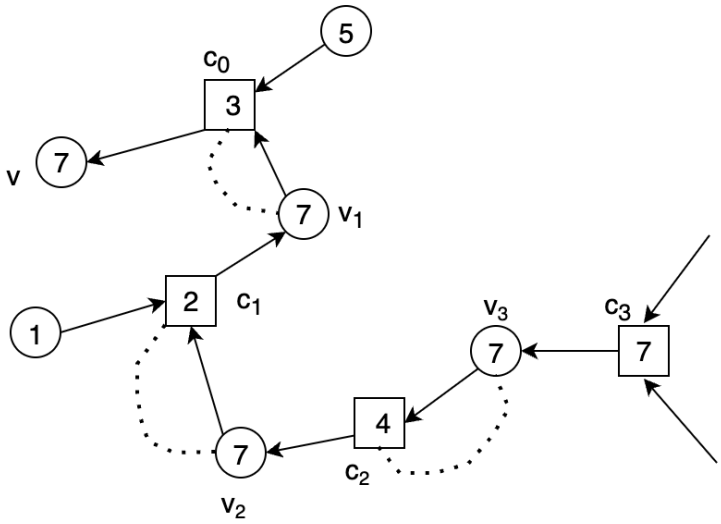

Рис. 3.3: Пример нахождения обратимого пути. Квадратами обозначены ограничения, кругами - переменные. Пунктирной линией обозначено отношение переменной к потенциальным выходам ограничения

- имеем функции: добавить ограничение c, удалить ограничение c, обновить приоритет stay-ограничения;
- после каждой выполненной функции, если алгоритм не завершился с ошибкой об ошибке, то в G_S гарантировано содержится максимальное по locally-predicate-better решение нынешней системы G_C;
- поддерживаются только методы с единственным выходом;
- алгоритм завершается с ошибкой при возникновении цикла в G_S или при невозможности удовлетворить все ограничения из C_0;

3.5.1 Добавление ограничения

Задача: добавить ограничение $c = \langle R, r, M \rangle$ в C, $R \subseteq V$ в систему ограничений $\langle C, V \rangle$ с графом ограничений G_C и текущим LPB лучшим графом решений G_S.

Результат: получить LPB лучшее решение G'_S системы ограничений $\langle C \cup \{c\}, V \rangle$ без заблокированных ограничений.

Алгоритм (см. блок-схему на рисунке 3.4.):

1 Добавить ограничение c и соответствующие ненаправленные ребра в граф ограничений G_C;

2 Проверить, является ли ограничение c заблокированным для решения G_S, если нет, то завершить алгоритм: если $Pr(c) \neq 0$, то G_S - искомое решение, иначе вернуть сообщение

о невозможности выполнения всех ограничений с нулевым приоритетом;

3 Если алгоритм не завершился на предыдущем шаге, то найти потенциальный выход v ограничения c с численно наибольшей силой;

4 Найти обратимый путь $c_0, v_1, c_1, \ldots, v_n, c_n$ для переменной v такой, что $Pr(c_n) = F(v)$;

5 Удалить ограничение c_n из графа решения;

6 $\forall i \in \{1, \ldots, n-1\}$ изменить выбранный метод ограничения c_i на тот, который определяет переменную v_{i+1}

7 Добавить ограничение c в граф решения, выбрав метод, определяющий переменную v;

8 Проверить наличие циклов в графе решения; если цикл есть, алгоритм возвращает сообщение об ошибке, иначе алгоритм завершается, новый граф решения - искомый.

3.5.2 Корректность алгоритма добавления ограничения

Сначала рассмотрим тривиальный случай завершения алгоритма на втором шаге: если добавленное в граф G_C ограничений c не является заблокированным, то во всей системе нет заблокированных ограничений относительно первоначального графа решения G_S. Это верно, поскольку изначальный G_S - не содержит заблокированных ограничений. G_S не изменился, следовательно, не изменились силы переменных. Значит, заблокированных ограничений появиться не могло. По лемме о заблокированном ограничении текущий G_S - лучший по LPB для новой системы ограничений. Если $Pr(c) = 0$, то в максимальном по LPB решении не выполняется условие корректного решения о удовлетворении всех ограничений с нулевым приоритетом, следовательно, не существует корректного решения.

Далее отметим, что на 3-м шаге алгоритма в силу определения заблокированного ограничения находится переменная, сила которой численно больше, чем приоритет ограничения c. Существование обратимого пути, указанного в шаге 4 обосновывается леммой об обратимом пути.

Будем обозначать G_S граф решения системы ограничений до начала выполнения алгоритма, а G'_S - текущий граф решения. Для $\forall u \in V$ $F(u)$ - сила переменной u в системе ограничений до начала выполнения алгоритма относительно решения G_S, $F'(u)$ - сила переменной u в текущей системе ограничений относительно решения G'_S,

Замечание 22. Измененный граф G'_S является корректным графом решения.

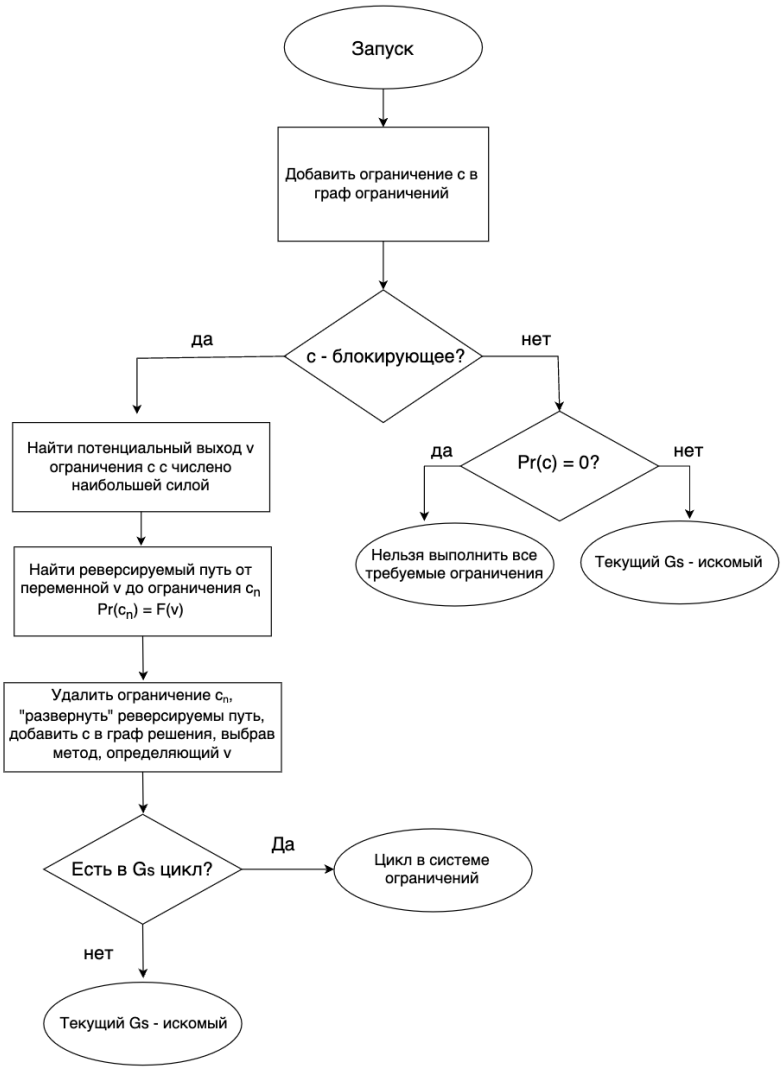

Рис. 3.4: Блок-схема алгоритма добавления ограничения

Доказательство. Отсутствие циклов в нем обеспечивается проверкой в пункте 7. В каждом ограничении системы ограничений выбран не более чем один метод, из чего так же вытекает корректность направленности ребер графа. Входная степень каждой переменной осталась равна 1, так как изменения затронули лишь переменные $v, v_1, v_2, \ldots, v_n$, для которых определяющее их ограничение было заменено. Из графа решения было удалено одно ограничение c_n, и его приоритет не мог быть нулевым, так как $Pr(c_n) = F(v) > Pr(c) \geqslant 0 \implies Pr(c_n) > 0$, следовательно, все ограничения с нулевым приоритетом выполнены. □

Последнее, что остается проверить - это факт, что в случае успешного завершения алгоритма, полученное решение действительно является максимальным по LPB. Для этого покажем, что относительно решения G'_S в системе нет заблокированный ограничений, то есть выполняется условие леммы о заблокированном ограничении.

Рассмотрим как поменялись силы переменных после выполнения алгоритма. Разделим

переменные текущего G'_S на три группы:

1 $M = \{v, v_0, v_1, \ldots, v_n\}$;

2 переменные достигаемые из переменных множества M;

3 все остальные переменные.

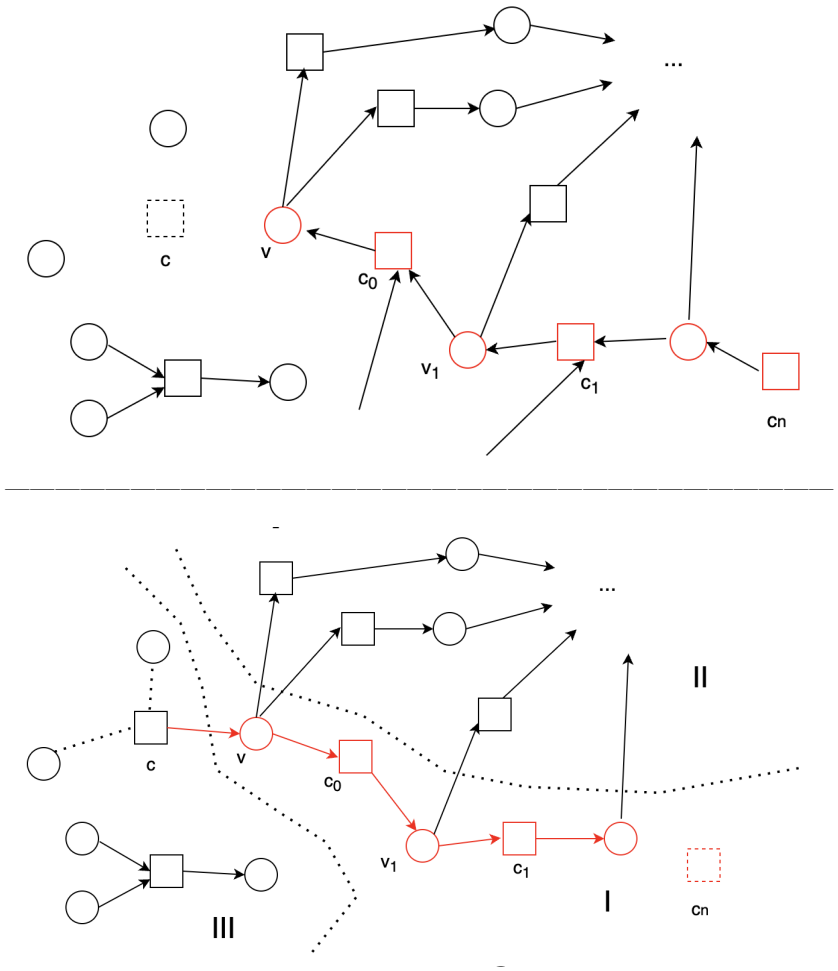

Рис. 3.5: Пример графа до и после работы алгоритма добавления ограничения c. Красным обозначен обратимый путь, пронумерованные области определяют группы переменных. Пунктирные линии из ограничения c обозначают потенциальные выходы

Замечание 23. Силы переменных третьей группы не изменились.

Доказательство. Поскольку сила переменной зависит только от сил переменных, лежащих «выше» в графе решения (достижимых по ребрам графа, полученного сменой направления каждого ребра), а все переменные и ограничения «выше» любой переменной из третьей группы остались неизменны по сравнению из изначальным графом решения, то силы переменных третьей группы остались неизменны. □

Замечание 24. Все ограничения из обратимого пути, рассмотренного в алгоритме, не имеют потенциальных выходов в переменные второй группы.

Доказательство. Предположим противное и у ограничения c_i из обратимого пути есть потенциальный выход v' из второй группы.

Предположим, что v' достижима из переменных v или $v_j, j < i$. В силу по правила о входах и выходах метода, в графе решения до добавления ограничения c из v' в c_i было проведено ребро. Тогда в этом графе был цикл: Из v' можно пройти по ребру в c_i, из c_i попасть в v' через одну из вершин множества $\{v\} \cup \{v_j \mid j \in \{1, \ldots, i-1\}$. Противоречие с определением графа решения.

Пусть v' достижима из переменных $v_j, j \geqslant i$. Тогда по тому же правилу о входах и выходах метода в графе решения после добавления ограничения c проведено ребро из v' в c_i. Тогда в этом графе есть цикл: из v' можно попасть в c_i по ребру, из c_i в v' через переменные обратимого пути. В этом случае алгоритм завершился бы раньше с сообщением об ошибке. □

Замечание 25. $\forall i \in \{0, \ldots, n-1\}$ ограничение c_i из обратимого пути , рассмотренного в алгоритме, не имеет потенциальных выходов в переменные обратимого пусти, кроме переменных v_i и v_{i+1} (для c_0 соответственно v и v_1).

Доказательство. Предположим противное и для c_i потенциальным выходом является переменная v_j так, что они нарушают условие замечания. Тогда рассмотрим ограничения обратимого путь в первоначальном графе и в графе после добавления. В силу инварианта о входах и выходах метода получаем, что ребро (v_j, c_i) проведено в обоих из них. Но так как в одном из этих графов v_j достижима из c_i, то либо максимальное по LPB решение первоначальной системы ограничений было некорректным, либо алгоритм завершится в ошибкой. Противоречие. □

Утверждение 26. *Силы переменных из множества M не увеличились.*

Доказательство. Будем рассматривать множество переменных множества M поочередно в порядке топологической сортировки:

$Pr(c) < F(v)$ и v имеет максимальную силу среди всех потенциальных выходов ограничения c. Заметим, что все потенциальные выходы c принадлежат третьей группе переменных, иначе, в графе бы возник цикл и алгоритм завершился с ошибкой, следовательно, их силы мы уже знаем, они сохранились из первоначального графа решения (см. замечание 25). $F'(v)$ равна максимуму из сил потенциальных выходов ограничения c и приоритета c - все

эти значения не больше силы $F(v)$, значит, сила переменной v после работы алгоритма не возрастет.

Далее рассматриваем переменную v_1. Определяющее её ограничение c_0 входит в обратимый путь, значит, по замечаниям 23 и 24 все потенциальные выходы - это переменные третьей группы и переменные v, v_1. В силу того, что в решении до добавления ограничения c по лемме об обратимом пути все переменные обратимого пути имели одинаковую силу, получаем, что

$$\max_{u \in c_0.\text{pout} \setminus \{v, v_1\}} F(u) = \max_{u \in c_0.\text{pout} \setminus \{v, v_1\}} F'(u) \leqslant F(v) = F(v_1) \implies$$

$$\implies F'(v_1) = \max\{Pr(c_0), \max_{u \in c_0.\text{pout}\{v, v_1\}} F'(u), F'(v)\} \leqslant F(v_1)$$

В переходы выше учитывалось, что $\forall i, j \in \{1, \ldots, n\}\ Pr(c_i) \leqslant F(v_j)$ - лемма об обратимом пути.

Аналогичные рассуждения можно проделать для любой переменной обратимого пути.

□

Замечание 27. Силы переменных из второй группы не увеличились.

Доказательство. Так как решение, возвращенное алгоритмом добавления корректно, то в нем нету циклом. Сделаем на нем топологическую сортировку рассмотрим переменные второй группы в этом порядке.

Пусть сейчас на рассмотрении u, и пусть ее определяет ограничение s. Заметим, что у s нету потенциальных выходов из второй группы, которые еще не рассмотрены, в силу определения топологической сортировки. Значит, множество по которому мы берем максимум в определении силы переменной u в новом решении состоит из $Pr(s)$, новых сил каких-то переменных первой и третьей группы и уже посчитанных новых сил переменных второй группы. Так как все силы переменных первой и третьей группы не возросли, то по индукции получаем, что и силы переменных из второй не возросли. □

Утверждение 28. *Если алгоритм добавления ограничения завершился без ошибки, то в новом графе решения нет заблокированных ограничений и он максимальный по LPB.*

Доказательство. В случае, если добавленное ограничение c не удовлетворено в графе решения G'_S, то оно не является заблокированным. Так как G_S максимальный по LPB граф решения системы до добавления c совпадает с G'_S, то и во всей системе нет заблокированных ограничений.

Теперь случай, когда c - удовлетворено в G'_S. По определению заблокированного ограничения - те, которые не входят в граф решения. После работы алгоритма такими являются ограничения, не входившие в первоначальное решение G_S (обозначим как множество N) и ограничение c_n (в терминах алгоритма).

В силу условия на персов начальное состояние системы среди элементов N нет заблокированных относительно G_S. Так как в новом решении G'_S силы всех переменных не увеличились (см. 23, 26, 27), то и в системе ограничений после добавления c ограничения из N не будут заблокированными относительно решения G'_S.

Остается рассмотреть ограничение c_n. До работы алгоритма оно определяло переменную v_n, причем $F(v_n) = Pr(c_n)$. В силу определения силы переменной получаем, что силы всех потенциальных выходов ограничения c_n первоначальной системе относительно решения G_S были $\leqslant Pr(c_n)$. После добавления все силы переменных не увеличились, значит, и в новой системе относительно G'_S силы всех потенциальных выходов будут $\leqslant Pr(c_n)$. Значит, c_n не является заблокированным.

Значит, заблокированных ограничений в системе после работы алгоритма добавления ограничения - нет. По лемме о заблокированном ограничении новое решение - максимальное по LPB. □

По итогу, алгоритм добавления ограничения в систему ограничений корректен.

3.5.3 Удаление ограничения

Задача: удалить ограничение $c = \langle R, r, M\rangle$ в C, $R \subseteq V$ из систему ограничений $\langle C, V\rangle$ с графом ограничений G_C и текущим LPB лучшим графом решений G_S (считаем, что c - не stay-ограничение).

Результат: получить LPB лучшее решение G'_S системы ограничений $\langle C\backslash\{c\}, V\rangle$ без заблокированных ограничений.

Алгоритм (см. блок-схему на рисунке 3.6.):

1 Если c было не удовлетворено в G_S, то удалить c из G_C и завершить работу;

2 Если алгоритм не завершился на предыдущем шаге, то рассмотреть $v = c$.method.out. Удалить c из графа решения и из системы в целом;

3 Добавить в граф решения stay-ограничение переменной v (обозначим его c_{stay}), выбрав единственный возможный метод;

4 Перебрать все заблокированные ограничения системы и взять из них c' - одно с численно наименьшим приоритетом; если такого не нашлось, то завершить алгоритм;

5 Запустить алгоритм добавления ограничения c' для текущего системы ограничения.

3.5.4 Корректность алгоритма удаления ограничения

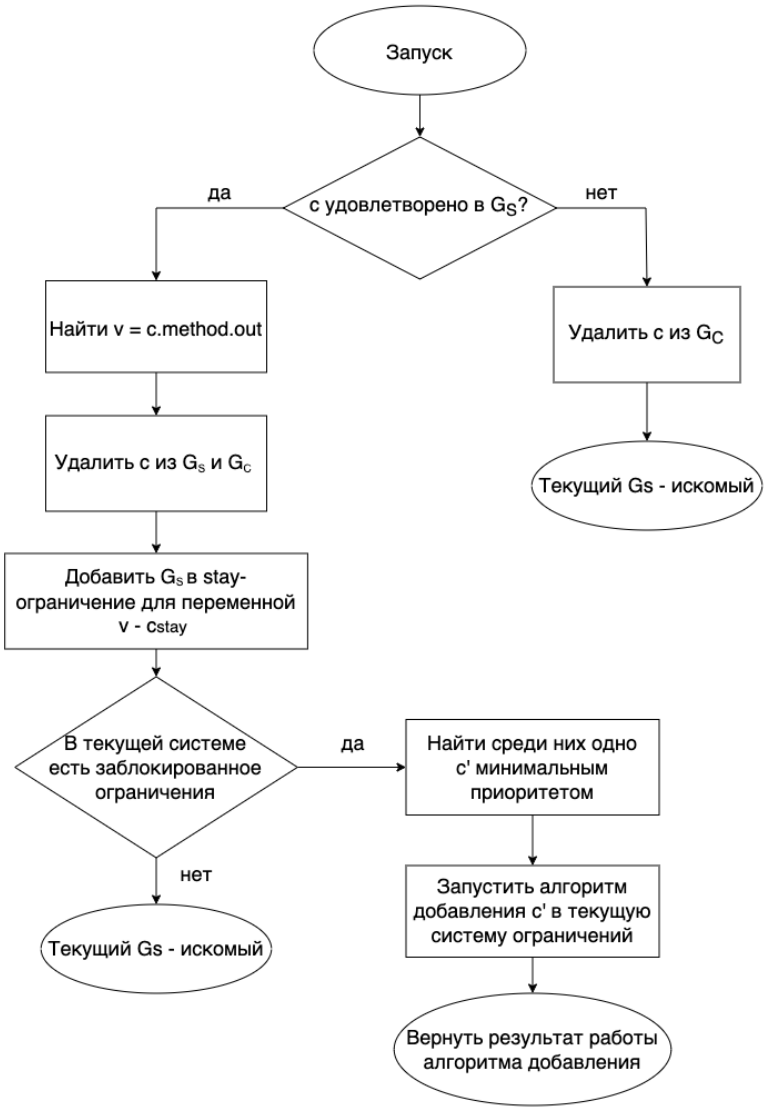

Рис. 3.6: Блок-схема алгоритма удаления ограничения

Сразу заметим важную деталь: в момент вызова функции добавления ограничения на этапе 5 не гарантируется, что текущий граф решений является максимальным по LPB, то есть мы интерпретируем данную функцию лишь как последовательность действий, полностью пользоваться его корректностью нельзя. Проанализировав доказательства из предыдущего параграфа, можно лишь утверждать, что алгоритм добавления ограничения, приме-

ненный не к максимальному по LPB решению вернет корректное решение, не обязательно максимальное.

Будем как и ранее обозначать G_S - граф решения системы ограничений до начала выполнения алгоритма, а за G'_S - текущий граф решения. Для $\forall u \in V$ $F(u)$ - сила переменной u в системе ограничений до начала выполнения алгоритма относительно решения G_S, $F'(u)$ - сила переменной u в текущей системе ограничений относительно решения G'_S,

Начнем с разбора случая завершения алгоритма на первом шаге: если G_S не содержало c, то, удалив c из решения, мы получим, что новая система ограничений не содержит заблокированных ограничений относительно G'_S, так как по вводным условиям их не содержал G_S. По лемме о заблокированном ограничении в новой системе G'_S - максимальный по LPB. Дальше считаем, что алгоритм не завершился на первом шаге.

Замечание 29. Граф G'_S, полученный после третьего шага алгоритма является корректным графом решения.

Доказательство. $G_{'S}$ был получен из корректного, а значит, ациклического, графа решения G_S удалением ограничения c, определявшего переменную v и добавлением c_{stay}, которое определяет ту же переменную v и выбранный метод которого не имеет входных переменных, то есть $\forall v \in V|v.\text{in}| = 1$ и ни один цикл, использующий c_{stay} не мог образоваться. □

После добавления c_{stay} в G'_S часть переменных обновили свои силы, соберем их во множество L.

Утверждение 30. *Все переменные множества L после выполнения третьего шага алгоритма удаления ограничения имеют силу $Pr(c_{stay})$ и лежат «ниже» переменной v.*

Доказательство. G'_S корректный, значит, к нему можно применить топологическую сортировку. Заметим, что все переменные до рассмотрения v не изменят свою силу по сравнению с первоначальным графом решения до удаления c в силу определения силы переменной.

Ограничение, определяющее переменную v, не имеет других потенциальных выходов, а значит, $F(v) = Pr(c_{stay})$.

Пусть дальше рассматривается переменная u, которую определяет ограничение c_u. Тогда по индукции для оставшейся последовательности переменных получаем, что потенциальные выходы ограничения c_u либо не изменили свою силу, либо увеличили силу до $Pr(c_{stay})$. Пересчет силы по определению - нахождение максимума на множестве значений.

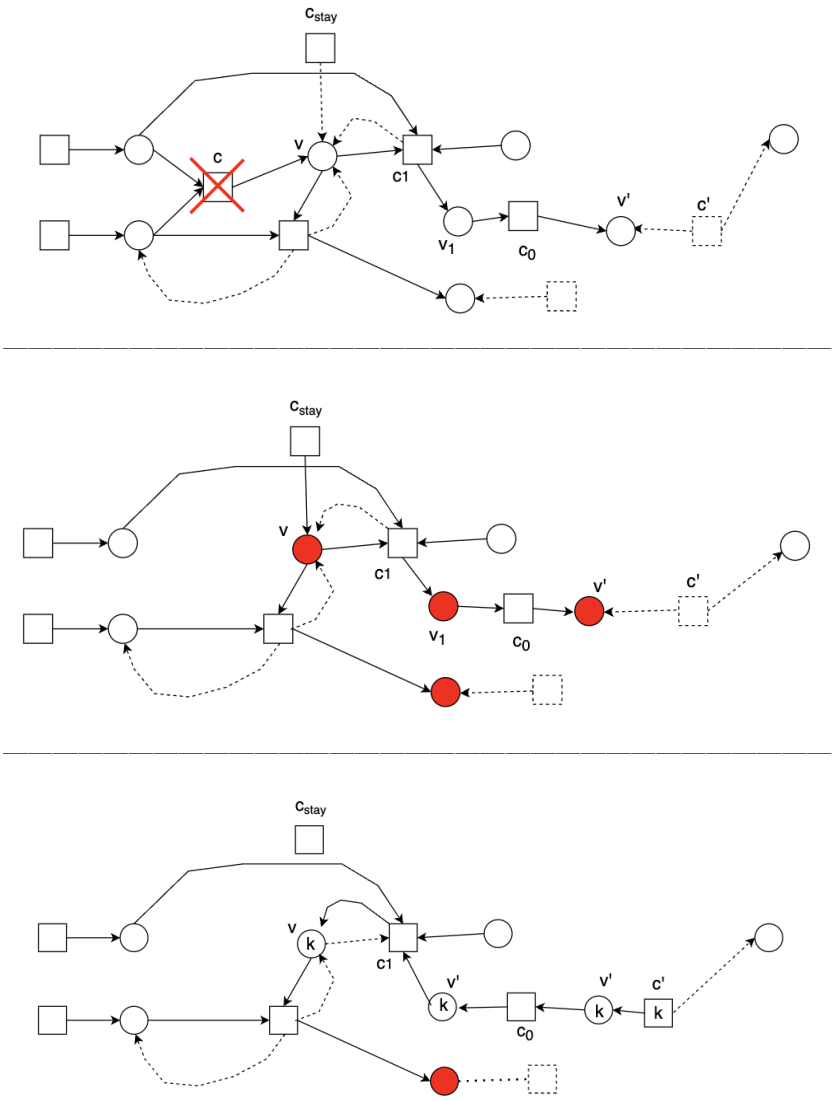

Рис. 3.7: Пример процесса удаления ограничения c из системы ограничений. Обозначения совпадают с введенными в алгоритме. Пунктирными стрелками обозначены потенциальные выходы переменных. Красным обозначены переменные, входящие во множество L. k - приоритет ограничения c'

Если часть элементов увеличить до фиксированного значения, то максимум либо не изменится, либо станет равен этому значению. Таким образом, новая сила переменной u равна либо силе до удаления c, либо $Pr(c_{stay})$. □

Далее рассмотрим подробнее 4 шаг алгоритма удаления ограничения. Если заблокированного ограничения в новой системе нет, то по лемме о заблокированном ограничении текущий граф решения соответствует максимальному по LPB решению, следовательно, можно корректно завершить алгоритм. Иначе можно утверждать, что существует заблокированное ограничение c', причем можно выбрать c' так, что его приоритет численно минимальный среди всех заблокированных ограничений системы.

Замечание 31. Все потенциальные выходы c', сила которых численно больше, чем $Pr(c')$

обязаны лежать в L, иначе c' был бы заблокированным и до начала работы алгоритма, что противоречит вводным данным о максимальности первоначального решения, то есть сила всех таких переменных по утверждению 30 равна $Pr(c_{stay})$.

Перейдем к этапу 5 и посмотрим, что происходит во время процесса добавления c' по шагам алгоритма добавления ограничения:

1 В графе ограничений c' уже присутствует;

2 Нам известно, что c' является заблокированным;

3 Найдется $v' \in c'$.pout с численно наибольшей силой, $F(v')$ относительно текущего G'_S точно больше $Pr(c')$;

4 Найденный обратимый путь $c_0, v_1, c_1, \ldots, v_n, c_n$ от переменной v' будет идти в точности до ограничения c_{stay} (то есть $c_n = c_{stay}, v_n = v$): в силу договоренности о том, что все stay-ограничения различны, c_{stay} - единственное ограничение, удовлетворяющее условие, о том, что $Pr(c_n) = F(v)$;

5 $\forall i \in \{1, \ldots, n-1\}$ изменится выбранный метод ограничения c_i на тот, который определяет переменную v_{i+1}

6 Добавить ограничение c в граф решения, выбрав метод, определяющий переменную v';

7 Из G'_S будет удалено ограничение c_{stay};

8 В граф G'_S будет добавлено отграничение c', выбранный метод будет определять переменную v'

.

Замечание 32. Если алгоритм добавления c' в систему ограничений не завершился с ошибкой о нахождении цикла, то $\forall i \in \{1, \ldots, n-1\}$ ограничение c_i из обратимого пути, полученного в процессе работы алгоритма добавления ограничения c' в систему ограничений, и ограничение c' не имеют потенциальный выход в переменную, лежащую «ниже» переменной v в первоначальном графе G_S, не совпадающую с соседними переменными в обратимом пути (то есть с v_i и v_{i+1} для $i \in \{1, \ldots, n-1\}$ и v и v_1 для $i = 0$).

Доказательство. Сначала для ограчниений обратимого пути. Предположим противное и найдется ограничение c_i и u, нарушающее это условие.

u не может лежать в самом обратимом пути в силу замечания 24, которое корректно и для нашей ситуации.

Иначе u не принадлежит в обратимому пути, но лежит «ниже» переменной v в графе G'_S до «разворота» обратимого пути. Найдём наименьшего общего предка u и c_i, обозначим его w. Тогда после осуществления «разворота» обратимого пути алгоритмом добавления ограничения из c_i будет путь до w, а из w путь до u. В силу правила о входах и выходах метода из u в c_i должно вести ребро, следовательно, в графе решения G'_S образовался цикл, что противоречит условию о том, что алгоритм завершился без ошибки.

Осталось проверить для ограничения c'. Если один из его потенциальных выходов - это переменная v_i обратимого пути, то выполнив алгоритм добавления мы получим, что из c' можно попасть в v' по ребру, из которой в свою очередь по обратимому пути можно попасть в v_i. Но по правилу входов и выходов метода в G'_S обязано присутствовать ребро (v_i, c') - цикл. Противоречие с успешным завершением алгоритма добавления.

Пусть u - потенциальный выход c', который не лежит на пути, но лежит «ниже» переменной v. Найдём w - наимаеньший общий предок v' и u в графе решения G'_S до осуществления разворота обратимого пути. После выполнения алгоритма добавления ограничения из c' по ребрам G'_S можно попасть в вершину w, а из неё по другим ребрам в вершину u. В силу правила входов и выходов метода в G'_S обязано быть проведено (u, c_i). Следовательно, в графе решения G'_S образовался цикл, что противоречит условию о том, что алгоритм завершился без ошибки. □

Утверждение 33. *Если алгоритм добавления c' в систему ограничений не завершился с ошибкой о нахождении цикла, то все переменные обратимого пути, возникшего на 4 этапе алгоритма добавления ограничения c', и переменная v' имеют силу $Pr(c')$.*

Доказательство. Так как в изначальном G_S не было заблокированных ограничений и c' не был удовлетворен, то в изначальной системе относительно G_S все потенциальные выходы c' имели силу $\leqslant Pr(c')$. В силу замечания 31 имеем, что все потенциальные выходы c' не лежат «ниже» переменной v, то есть по утверждению 29 не изменили свою силу по сравнения с состоянием системы до работы алгоритма удаления. Тогда сила переменной v' равна в точности $Pr(c')$.

Далее рассмотрим переменную v_1 данного обратимого пути. Предположим, что у c_0 есть потенциальный выход u такой, что $F(u) > Pr(c')$. Все потенциальные выходы c_0 не лежат на обратимом пути или «ниже» переменной v по замечанию 31, значит, сила u в текущем системе относительно G'_S совпадает с силой u в системе до применения алгоритма

удаления относительно G_S. В G_S c_0 определяло переменную v', значит, $F(v')$ относительно $G_S \geqslant Pr(c')$. Тогда ограничение c' - заблокированное в G_S. Противоречие. Значит, у c_0 нет потенциальных выходов с силой больше, чем $Pr(c')$.

$Pr(c_0) \leqslant Pr(c')$, иначе в изначальной системе ограничений относительно G_S $F(v') > Pr(c')$ и c' - заблокированное, чего быть не может. Таким образом, получаем, что

$$F(v_1) = \max(Pr(c_0), \max_{u \in c_0.\text{pout} \setminus v_1} F(u)) =$$

$$= \max(Pr(c_0), \max_{u \in c_0.\text{pout} \setminus \{v_1, v_1\}} F(u), F(v')) = \max(Pr(c_0), Pr(c')) = Pr(c')$$

Аналогичные рассуждения можно проделать для каждой переменной обратимого пути. □

Утверждение 34. *Если алгоритм добавления c' в систему ограничений не завершился с ошибкой о нахождении цикла, то $\forall u \in V$ такой, что u лежит «ниже» переменной v, выполняется:*

1 *Если $u \notin L$, то $F'(u) = F(u)$*

2 *Если $u \in L$, то $F'(u) = F(u)$ или $F'(u) = Pr(c')$ и $F(u) \leqslant Pr(c')$.*

что $F'(u)$ относительно G'_S либо не изменилась по сравнению с $F(u)$ в первоначальной системе относительно G_S, либо стала равна $Pr(c')$, при этом не уменьшившись.

Доказательство. Для u, принадлежащих обратимому пути (которые так же принадлежат и множеству L), верно, что силы равны $Pr(c')$ из утверждения 32. Остается заметить, что в первоначальной системе относительно G_S сила переменных была не больше. Помним, что в G_S c' не был блокирующим, значит, $F(v') \leqslant Pr(c')$. Сила $F(v_1)$ не могла быть больше $Pr(c')$, иначе, $F(v') \geqslant F(v_1) > Pr(c')$. Аналогичные рассуждения для все переменных обратимого пути.

Докажем утверждение по индукции, рассматривая переменные в порядке топологической сортировки переменных, лежащих ниже v без переменных обратимого пути, для которых силы уже известны.

Пусть рассматривается переменная u, которую определяет ограничение s.

Если $u \notin L$, то мы знаем, даже если численно увеличить силу всех потенциальных выходов s, лежащих в L до $Pr(c_{stay})$, этого не будет достаточно, чтобы изменить силу u, поскольку иначе переменная u так же бы попала во множество L. По индукции все потенциальные выходы не из L имеют ту же силу, что и в изначальной системе, а выходы из

L силу меньше, чем $Pr(c')$, следовательно, новая сила переменной u останется неизменной $F'(v) = F(v)$.

Если $u \in L$: потенциальными выходами s могут быть переменные либо переменные обратимого пути - их сила равна $Pr(c')$, либо переменные, сила которых не изменилась по сравнения с состоянием до применения алгоритма, либо переменные из L, уже обработанные индукцией - их сила совпадает с силой в первоначальной системе относительно G_S или равна $Pr(c')$, при этом она не изменилась в меньшую сторону. Тогда из определения силы переменной получаем, что $F'(u)$ действительно либо равна $F(v)$, либо $Pr(c')$ в силу рассуждений, изложенных в утверждении 25. □

Утверждение 35. *В случае завершения работы алгоритма удаления переменной без ошибки в системе ограничений с графом решения G'_S нет заблокированных ограничений.*

Доказательство. Рассмотрим множество B всех заблокированных ограничений в системе на 4 этапе алгоритма удаления. Для любого из них так же применимо замечание 30, то есть все потенциальные выходы, сила которых больше приоритета ограничения лежат во множестве L. По утверждению 33 силы всех переменных не возросли по сравнению с состоянием системы на 4 этапе. Значит, после завершения работы алгоритма заблокированным могут являться только ограничения из множества B.

Пусть найдется $s \in B$, оставшееся заблокированным и после завершения работы алгоритма. Рассмотрим любой его потенциальный выход u такой, что $F'(u) > Pr(s)$. По утверждению 33 $F'(u)$ равно либо $F(u)$, либо $Pr(c')$. В первом случае возникает следующее противоречие: алгоритм не делает ни одно ограничение неудовлетворенным (само удаляемое омрачение c удаляется даже из C), то есть s было не удовлетворено и в G_S, что означает, что $Pr(s) \geqslant F(u)\ \forall u \in s.\text{pout}$. Во втором случае получаем, что $Pr(c') > Pr(s)$, но $s \in B \implies$ на шаге 4 выбран был бы не ограничение c', а ограничение s. Снова противоречие.

Таким образом, в системе ограничений нет заблокированных ограничений и G'_S является максимальным по LPB в силу леммы о заблокированном ограничении. □

По итогу, алгоритм удаления ограничения из системы ограничений корректен.

3.5.5 Обновление приоритета stay-ограничения

Напомним, что для предсказуемости поведения Property model, от алгоритма решателя ограничений, заложенного в её ядро мы просим умение обновлять приоритет stay-ограничения при изменении значения переменной со стороны пользователя, делая его самым приоритетным на спектре stay-ограничений.

Этот функционал мы реализуем следующим образом через уже полученные функции.

Задача: обновить приоритет stay-ограничения переменной $v \in V$ (обозначим c_{stay}) из системы ограничений $\langle C, V \rangle$ с графом ограничений G_C и текущим LPB лучшим графом решений G_S.

Результат: получить LPB лучшее решение G'_S системы ограничений $\langle C \backslash, V \rangle$ без заблокированных ограничений.

Алгоритм:

1 Инициализировать ограничение c'_{stay} по аналогу c_{stay}, в котором заложен новый приоритет и вместо v.value из определения stay используется уже новое значение;

2 Вызвать алгоритм добавления ограничения c'_{stay} в систему ограничений.

3 Ограничение c_{stay} в любом случае окажется не удовлетворено, с данного момента считаем, что stay для переменной v является c'_{stay}

4 Ограничение c_{stay} в любом случае окажется не удовлетворено, вызвать на него удаление из системы ограничений.

Поскольку добавление и удаление ограничений корректны и новое ограничение c'_{stay} действительно может корректно стать stay для v на третьем этапе алгоритма, то полученная система действительно удовлетворяет желаемому результату. Остается вопрос, почему на третьем этапе c_{stay} не удовлетворено?

Замечание 36. На третьем этапе алгоритма обновления приоритета stay-ограничения c_{stay} не удовлетворен в текущем графе решения.

Доказательство. Предположим противное и c_{stay} удовлетворен. Тогда c'_{stay} удовлетворен быть не может, так как единственный потенциальный выход определяется другим ограничением. Возникает противоречие: по определению силы переменной текущая сила v равна $Pr(c_{stay})$, что численно больше, чем $Pr(c'_{stay})$, то есть c'_{stay} - заблокированное, но алгоритм добавления гарантирует возвращение системы ограничений без заблокированных ограничений. □

3.5.6 Асимптотика алгоритма

Будем апеллировать следующими величинами: V - суммарное количество ограничений и переменных в модели, E - суммарное количество неориентированных рёбер в графе ограничений G_C.

Пройдемся по введенным функциям:

Добавление ограничения:

1 Добавить ограничение и необходимые ребра в граф ограничений в можно точно за $O(V)$ (худший случай достигается, например, если в систему ограничений из переменных и stay-ограничений добавить ограничение, использующее все переменные системы);

2 Для проверки ограничения на заблокированность достаточно пройтись по всем методам, число которых оценивается сверху количеством переменных, задействованных в ограничении, в худшем случае это можно сделать за $O(V)$ операций (пример худшего исхода совпадает с предыдущим пунктом)

3 Нахождение потенциального выхода с численно максимальной силой осуществимо за то же количество операций, что и проверка на заблокированность; более того, эти действия можно совместить - асимптотика $O(V)$;

4 Нахождение обратимого пути описано в лемме о заблокированном пути, его асимптотика составляет в худшем случае составляет $O(V + E)$: нам необходимо итеративное для каждого ограничения пути найти потенциальный выход, удовлетворяющий конкретному условию, то есть в худшем случае нужно совершить $O(V + E_S)$ операций, где E_S - количество ребер в текущем графе решения. Воспользуемся очевидно оценкой $E_S \leqslant E$ и получим заявленную асимптотику;

5 Удаление ограничения из графа решения к перебору всех вершин, задействованных в нем. В худшем случае - $O(V)$;

6 Разворот обратимого пути сопровождается не только изменением направленности ребер, но и пересчетом сил переменных системы ограничений. Совершение первого действия требует количества операций, пропорционального количеству переменных в пути, в худшем случае $O(V)$. На пересчет сил придется затратить в худшем случае $O(V+E)$ операций, поскольку, как уже было указано выше, он содержит в себе топологическую сортировку графа, асимптотика которой равна сложность обхода в глубину - $O(V+E)$;

7 Проверка наличия цикла так же сводится к обходу в глубину, то есть асимптотика $O(V + E)$

Таким образом, итоговая асимптотика добавления ограничения в систему ограничений равна $O(V + E)$.

Удаление ограничения:

1 Удаление ограничения из графа G_C в худшем случае занимает $O(V)$;

2 Добавление ограничения в граф решения и пересчет сил займет $O(V + E)$;

3 Перебор всех ограничений и проверка на заблокированность так же сводится к сложности $O(V + E)$;

4 Алгоритм добавления ограничения, как было сказано выше работает за $O(V + E)$.

Таким образом, итоговая асимптотика удаления ограничения из системы ограничений равна $O(V + E)$.

Обновление приоритета stay-ограничения:

1 Алгоритм добавления ограничения требует $O(V + E)$ операций;

2 Алгоритм удаления ограничения требует $O(V + E)$ операций;

3 Остальные действия совершаются за $O(1)$.

Таким образом, итоговая асимптотика обновление приоритета stay-ограничения равна $O(V + E)$.

Если принять допущение, что количество переменных любом из ограничений системы ограничено небольшой константой C, то можно уточнить асимптотику и утверждать, что все операции выполняются за $O(V)$. Это возможно, так как в этом случае общее число ребер в графе ограничений оценивается, как количество ограничений умноженное на C. То есть $O(V + E) = O((1 + C)V) = O(V)$.

4 Ход работы и обзор литературы

Заметим однако, что приведенное выше доказательство не совпадает с доказательствами, приведёнными в известных нам статьях. Дело в том, что в ходе работы над проектом было рассмотрено несколько статей и диссертаций на тему описания работы и доказательства корректности алгоритма DeltaBlue. Рассмотрим ошибки, допущенные в этих статьях и приведем контрпримеры для утверждений, сформированных там.

4.1 «An Incremental Constraint Solver»

Первой статьей, рассмотренной в ходе работы была статья [6], на которую ссылались составители статей про Property Model.

В статье дано определение другое определение силы переменной:

Определение 37. Сила переменной (в статье - *walkabout strength*) $v \in V$, определяемая в текущем графе решения G_S системы ограничений $\langle C, V \rangle$ методом m ограничения c, равна численному максимуму множества, состоящего из приоритета ограничения c и сил переменных, являющихся входными переменными метода m.

Далее в статье вводится определение заблокированного ограничения, совпадающее с нашим, и доказывается лемма о заблокированном ограничении:

Лемма 38 (О заблокированном ограничении)**.** *Если в системе ограничений нет заблокированных ограничений, то то решение максимально по LPB (определение сравнения решений совпадает с нашим)*

Доказательство. Пусть дано решение R, не содержащее заблокированных ограничений. Пусть существует решение Q лучше по LBP. Тогда по определению LPB $\exists k \in \mathbb{Z}_{\geqslant 0}$ такое, что на все ограничения с приоритетом k решение R удовлетворяет все ограничения с приоритетом k решения Q и хотя бы ещё одно ограничение c. Пусть v_c - переменная с минимальной силой относительно решения Q, задействованная в ограничении c (то есть если $c = \langle R, r, M \rangle$, то $v_c \in R$). Пусть её сила равна $w^Q_{v_c}$ Так как c не удовлетворен в Q и в Q нет заблокированных ограничений, то $Pr(c) \leqslant w^Q_{v_c}$. Однако, в силу того, что c удовлетворен в R, $Pr(c) < w^Q_{v_c}$. Противоречие, не существует ограничения лучше, чем Q. □

Теперь разберемся, что не так с этим доказательством. Утверждение о том, что если c удовлетворен в R, то $Pr(c) < w^Q_{v_c}$ на самом деле не обосновано, так как в рамках статьи на этапе доказательства не вводились дополнительные леммы, позволяющие говорить о силе переменной в одном решении, опираясь на силу этой переменной в другом решении.

Далее в статье приводится алгоритм добавления переменной. Укажем на первую ошибку, допущенную в самом начале: в первом пункте алгоритма говорится, что у нового ограничения необходимо выбрать метод m, выход которой имеет силу не меньше, чем сила любого другого потенциального выхода. Далее идет ветвление, первый случай которого рассматривает исход "такой метод не нашёлся". В статье явно не дописано условие, что сила потенциальных выходов так же сравнивается с приоритетом ограничения, ведь иначе эта ветка никогда не исполняется: на непустом конечном множестве неотрицательных чисел обязательно найдется максимум.

Эта ошибка влечет за собой отсутствие проверки на невозможность выполнения всех ограничений с приоритетом 0. Рассмотрим следующую систему ограничений (рис. 4.1):

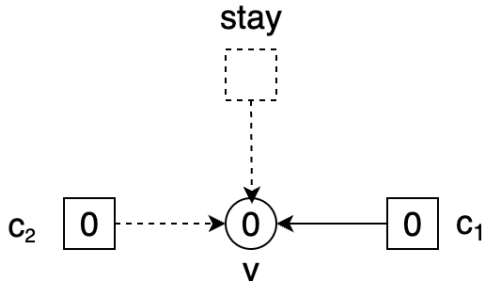

Рис. 4.1

c_1 и c_2 имеют нулевые приоритеты и только по одному методу, определяющему переменную v.

Предположим, что мы запускаем алгоритм добавления ограничения c_2, единственный потенциальный выход которого - v. Пройдем по шагам алгоритма дальше:

1 Выбираем потенциальный выход c_2 с наибольшей силой - v и определяющий его метод m; такой выход найдется и его "добавление"не вызовет цикл;

2 Удаляем ограничение c_1 из графа решения, определяющее v;

3 Добавляем c_2 в граф решения, выбрав метод m;

4 Пересчитываем силы;

5 Вызываем добавление ограничения c_1;

Нетрудно заметить, что перед исполнением пункта 5 система ограничений будет выглядеть следующим образом (рис. 4.2):

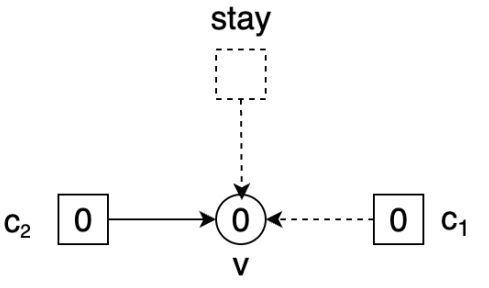

Рис. 4.2

Система совпадает с первоначальной с точностью до переименование c_1 повлечет непрекращающийся процесс изменения системы ограничения из состояния на рис. 4.1 в состояние на рис. 4.2 и обратно.

Из этого можно сделать вывод, что алгоритм, описанный в статье, явно не является корректным.

4.2 «Multi-way versus One-way Constraints in User Interfaces»

В поисках лучшего алгоритма была рассмотрена статья [9]. В ней действительно приведен другой алгоритм, изложенный псевдокодом. Из прочих заметных отличий от предыдущей статьи отметим, определение силы переменной совпадает с тем, которые используем мы. Однако доказательства корректности в статье не приводится, лишь идет ссылка на следующую диссертацию [10], которая заслуживает уже большего внимания.

В статье приведен следующий алгоритм добавления ограничения c в текущую систему ограничений $\langle C, V \rangle$ (задача и ожидаемый результат совпадают с нашим алгоритмом):

1 Инициализировать множество $U = \varnothing$. Туда будут добавляться переменные далее;

2 Добавить c в граф ограничений, если его там ещё нет;

3 Выберать метод m в c, потенциальный выход которого не лежит в U и имеет численно максимальную силу. Пусть потенциальный выход этого метода - v. Если сила v меньше или равна $Pr(c)$, то завершить алгоритм со следующим исходом: если $Pr(c) = 0$, то ошибка о невозможности удовлетворить все ограничения с приоритетом 0, иначе такущий граф решения - искомый;

4 Удалить ограничение D, ранее определявшее v из графа решения;

5 Добавить c в граф решения с выбранным методом m; добавить v в U; обновить силы переменных, лежащих «ниже» v. Если среди переменных «ниже» v найдется входная переменная c в текущем графе решений, то алгоритм завершается с ошибкой о цикле.

6 Осуществить шаги 3-6 для ограничения D.

Далее следует многостраничное доказательство корректности. Вернемся к статье Michael Sannella и заметим одну деталь: псеводокод добавления переменной реализует множество U через отметки *mark* на переменных системы ограничений. Однако во вложенной в *AddConstraint* функции *Enforce* заметно, что отметка *mark* проставляется не только на потенциальный выход выбранного метода добавляемой переменной, но и на инпуты этого метода. То есть в алгоритм добавятся новый шаг.

Тогда рассмотрим следующую систему ограничений (рис. 4.3) и попытаемся произвести добавление ограничения в соответствии с алгоритмом, полученным из псевдокода

c имеет единственный потенциальный выход v_3. У c_1 потенциальным выходом так же является v_1. До добавления c текущее решение действительно является лучшим.

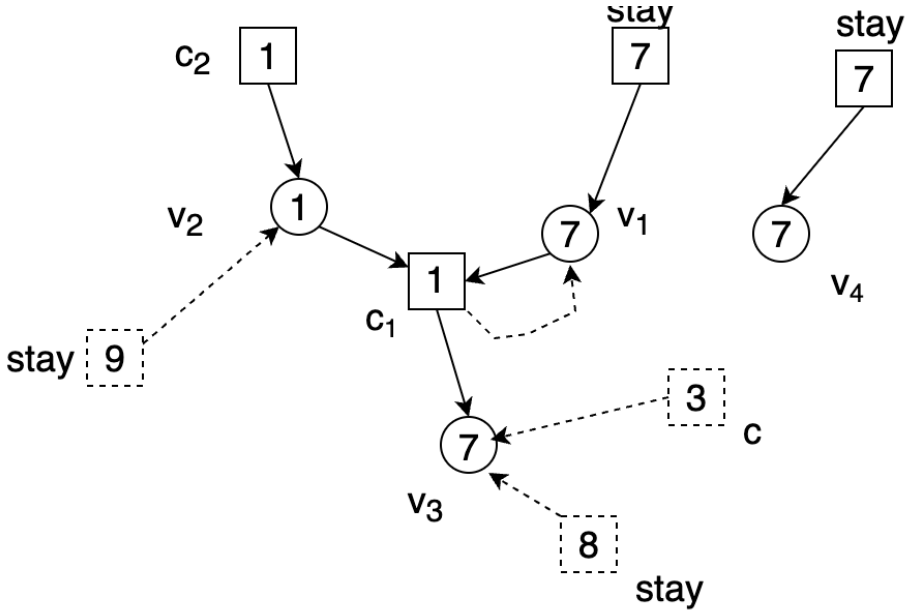

Рис. 4.3

Начнем проделывать алгоритм и, дойдя до этапа 6, мы получим следующую систему ограничений (рис. 4.4):

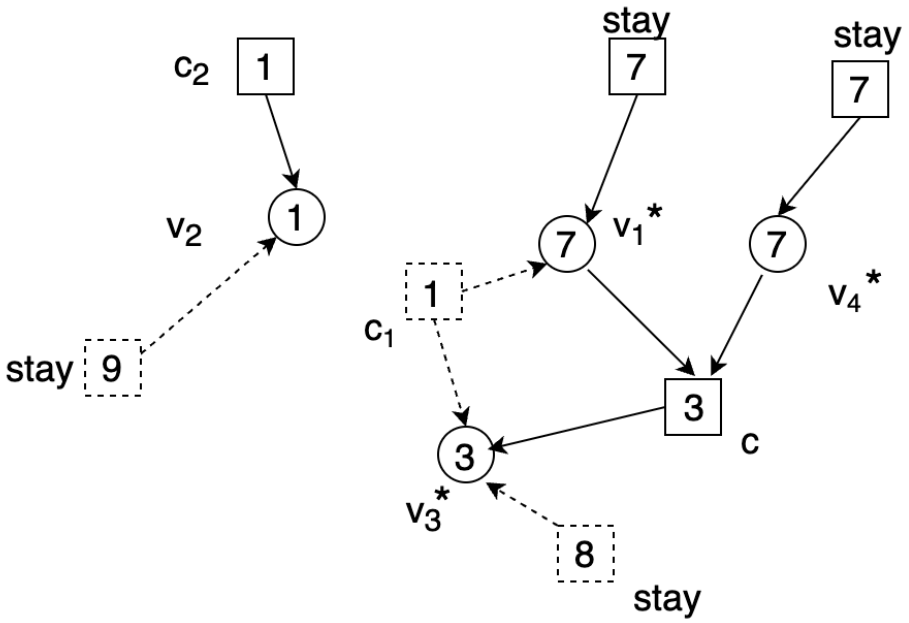

Рис. 4.4

Звездочками отмечены переменные, добавленные в U. По алгоритму из псевдокода в U будет лежать v_1, v_3, v_4. Тогда при попытке добавить в систему ограничение c_1, в котором заключается этап 6 алгоритма, ни один потенциальный выход не будет выбран, так как и v_1, и v_3 находятся во множестве U. Поскольку утверждается, что результатом работы алгоритма

будет либо выявление ошибки, либо корректное решение без заблокированных ограничений, то возникает противоречие: данная система потенциально цикличная(если удовлетворить c_1 выбрав метод, определяющий v_1, то действительно возникнет цикл), но алгоритм возвращает решение без цикла с заблокированным ограничением c_1 ($Pr(c_1) < F(v_1)$, v_1 - потенциальный выход c_1). Из этого можно сделать вывод, что псевдокод приведенный в статье некорректный.

Что касается алгоритма из диссертации [10], мы не стали использовать в точности его, поскольку:

- эта версия алгоритма не поддерживает иерархию stay-ограничений, необходимую нам для предсказуемой работы Property Model;
- в статье [11] доказано, что в функции пересчета сил переменных, которая реализована через поиск в ширину, допущена ошибка, которая превращает сложность работы алгоритма в экспоненциальную;

Однако нельзя не отметить, что в данной работе появляется идея о том, что при удалении ограничения из системы ограничений для нахождения нового лучшего решения достаточно запустить алгоритм добавления лишь раз (доказательство этой части тоже пришлось переделать в силу нужд Property Model).

4.3 Другие статьи

Так же в рамках дополнительной работы, было проведено ознакомление со статьями по другим алгоритмам решателей ограничений [12] - SkyBlue, [5] - QuickPlan.

5 Программная часть

В рамках реализации библиотеки я придерживался шаблонного подхода, позволяющего перенести максимальное количество действий на этап компиляции кода.

На этапе инициализации от пользователя требуется явно передать типы данных всех переменных и декларировать множество ограничений. Все переменные разделены на три логические группы *Data* - входные переменные интерфейса, с которыми взаимодействует пользователь, *Value* - вспомогательные переменные синтеза параметров модели, *Out* - выходные переменные синтеза параметров модели.

Обращение к определенной переменной системы со стороны пользователя происходит следующим образом:

```
1 Templates::Data<3>
```

После указания вспомогательного пространства имён *Templates* следует явное указание группы переменной с шаблонным параметром - индексом переменной в группе.

5.1 Верхние классы

5.1.1 Builder

Класс Builder - шаблонный класс, занимающийся первичной инициализацией Property Model.

Шаблонами класса *Builder* являются три шаблонных типа данных *Data<DataArgs...>*, *Value<ValueArgs...>*, *Out<OutArgs...>*. Каждый из них принимает набор произвольных типов данных *Args*, который уже в свою очередь определяет тип переменных *Property Model*. Разделение переменных на *Data*, *Value* и *Out* позволяет лучше структурировать работу модели.

Полями *Builder*-а являются нпосредственно конструируемая *Property Model* и буффер для создания текущего ограничения.

Рассмотрим функционал доступный пользователю:

1 Конструктор - явно принимает набор начальных значений переменных *Property Model* в порядке *Data*, *Value*, *Out*. Частичная спецификация шаблонов позволила реализовать следующее правило взаимодействия с пользователем библиотеки: если тип данных переменной T не удовлетворяет встроенное условие *std::is_ arithmetic_ v<T>*, то в объект в конструктор передаваться по rvalue ссылке, что позволит избежать лишних копирований тяжелых объектов.

2 Добавление ограничения - метод *AddNewConstraint* с параметром типа *Priority::Strength{k}*, где k имеет тип данных *int*. Данная функция инициализирует создание нового пользовательского ограничения с численным значением приоритета k;

3 Добавление метода в текущее ограничение - *AddMethod<Output, Inputs...>*. Единственный параметр - объект типа *std::function<Output(Inputs...)>* - задает функцию пересчета параметров, заложенных в метод.

4 Изменение начального значения переменной - метод *Set<MetaData>*. Единственный параметр - новое значение переменно. Тяжелые объекты (не удовлетворяющие условие *std::is_ arithmetic_ v<T>*) принимаются только по rvalue ссылке.

5 Изъятие *Proiperty Model - ExtractPM()*. Данный метод добавляет текущее инициализируемого ограничение в систему, создает для каждой переменной stay-ограничение и возвращает объект класса PropertyModel.

Пример взаимодействия с классом Builder:

```
1  using namespace NSPropertyModel;
2
3  using DataVars = Data<int, int, int>;
4  using ValueVars = Value<bool>;
5  using OutVars = Out<int>;
6  using PropertyModel = PropertyModel<DataVars, ValueVars, OutVars>;
7  using Builder = PropertyModel::Builder;
8
9  Builder b(0, 5, 5, false, -1);
10
11 b.AddNewConstraint(Priority::Strength{2});
12 b.AddMethod<Templates::Data<2>, Templates::Data<0>, Templates::Data<1>>(f_1);
13 b.AddMethod<Templates::Data<0>, Templates::Data<2>, Templates::Data<1>>(
14       f_2);
15
16 b.AddNewConstraint(Priority::Strength{1});
17 b.AddMethod<Templates::Value<0>, Templates::Data<0>, Templates::Data<1>,
18               Templates::Data<2>>(f_3);
19
20 b.Set<Templates<Data<0>>(1);
21
22 PropertyModel pm = b.ExtractPM();
```

5.1.2 PropertyModel

Класс PropertyModel - шаблонный класс, служащий объектом взаимодействия с моделью.

Шаблонными параметрами являются те же типы данных, что и у инициализирующего Builder-а. Класс PropertyModel содержит в себе единственное поле *std:unique_ptr<PropertyModelImpl>*, поэтому все взаимодействие происходит через вызов методов PropertyModelImpl через оператор →.

5.1.3 PropertyModelImpl

Класс PropertyModelImpl - шаблонный класс, содержащий все данные о модели и хранящий текущее состояние системы.

Пользователь не может явно обратиться к этому классу, но может вызывать его публичные методы через владеющий объект *PropertyModel*.

Отметим, что после инициализации с помощью *Builder*-а модель не может добавить совершенно новое ограничение. Однако библиотека позволяет гибко апеллировать уже имеющимся набором ограничений: все ограничения условно разделяются на удаленные, примененные в текущем графе решения и не включенные в текущий граф решения.

Рассмотрим функционал доступный пользователю:

1 Удалить ограничение - метод *RemoveConstraint*. Единственный параметр - индекс ограничения - объект типа *int*. Помечает ограничение как удаленное и вызывает пересчет графа решения. Если решение уже помечено как удаленное, не изменяет систему;

2 Добавить ограничение - метод *AddConstraint*. Единственный параметр - индекс ограничения - объект типа *int*. Добавляет удаленное ограничение в граф ограничения и переситывает граф решения;

3 Изменение значение переменной - *Set<MetaData>*. Единственные аргумент - объект value типа, совпадающего с типом изменяемой переменной - новое значение переменной *MetaData*. Меняет текущее значение переменной, обновляет приоритет stay-ограничения и пересчитывает граф решения.

Пример взаимодействия с классами PropertyModel и PropertyModelImpl:

```
1  int value;
2  std::cin >> value;
3  pm->Set<Templates::Value<0>>(value);
4
5  int index;
6  std::cin >> index;
7  pm->RemoveConstraint(index);
8
9  std::cin >> index;
10 pm->AddConstraint(index);
```

5.2 Вспомогательные объекты

5.2.1 Priority

Приоритет ограничений и силы переменных имплантированы через структуру *Priority*. содержащую в себе поля типа ограничени (*Stay / Reguar*) и численное значение приоритета/силы - целое неотрицательное число.

5.2.2 Ограничения, методы и переменные

Помимо явного представления переменных и ограничений как объектов внутри PropertyMod для создания графа ограничений были введены следующие структуры:

1 *Variable*. Хранит в себе текущее значение силы относительно графа решения, флаг для эффективного обхода графов и указатели: stay, ограничение, которое определяет переменную, и набор ограничений, в котором переменная задействована как потенциальный выход.

2 *Constraint*. Владеет собственным набором методов, а также хранит текущий статус (удалено/не удовлетворено/удовлетворено), приоритет, указатель на выбранный метод и флаг для эффективного обхода графов.

3 *Method*. Хранит указатели на переменные, которые являются входами и выходами метода, и непосредственно функцию пересчета.

5.2.3 ConstraintGraph

Класс ConstraintGraph - явно реализует граф ограничений системы ограничений, от которого он инициализируется.

Данный класс содержит поля *constraints* и *variables*. Оба они являются *std* :: *vector*-ом, содержащим указатели на хранимые в PropertyModel структуры ограничений и переменных соответственно.

Для данного класса реализованы некоторые публичные классы, полезные в имплантации алгоритмов:

```
1 Constraint* GetByIndex(IndexType index);
2 Constraint* FindLowestPriorityBlockedConstraint();
3 void ExecutePlan(StepType& propagation_counter);
```

Соответственно возвращение указателя на ограничение по индексу, нахождение заблокированного ограничения с минимальным приоритетом и исполнение плана пересчета (*propagation_ counter* - вспомогательный флаг для эффективного обхода графа).

Как уже было сказано ранее, удовлетворенность / не удовлетворенность ограничений определяется по полю структуры *Constraint*, то есть непосредственная реализация для графа решения не требуется.

5.2.4 DeltaBlue

Класс DeltaBlue - имплементирует вышеизложенный алгоритм на объекте ConstraintGrap

Публичными методами класса являются функции:

1 *AddConstraint* и *AddConstraintByIndex* - принимают ссылку на ConstraintGraph и ограничение (либо через указатель на Constraint, либо через индекс ограчения);

2 *RemoveConstraint* и *RemoveConstraintByIndex* - принимают ссылку на ConstraintGraph и ограничение (либо через указатель на Constraint, либо через индекс ограчения);

3 *UpdateStayPriority* - принимает ссылку на ConstraintGraph и указатель на stay-ограничение.

Действия методов считываются из их названий.

5.3 Тестирование

Тестирование каждого отдельного объекта проводилось с помощью юнит-тестов на корректность. Были реализованы и отработаны стресс-тесты для выявления нетривиальных сценариев поведения, которые не вывели неожиданностей.

Для тестирования библиотеки в целом были реализованы прототипы графических интерфейсов, например, диалоговое окно изменения размера графического объекта и диалоговое окно резервирования комнат в отеле, приведенные в качестве примеров в статьях Sean Parenta-а [1, 3] и т.п.

Ручное тестирование показало, что поведение алгоритма и модели соответствует ожиданиям. Добавление *assert*-ов в код программы показало, что не одна функция вызывается с нарушением требуемых инвариантов.

Список литературы

[1] Jaakko Järvi, Mat Marcus, Sean Parent, John Freeman и Jacob N Smith. “Property models: from incidental algorithms to reusable components”. В: *Proceedings of the 7th international conference on Generative Programming and Component Engineering*. 2008, с. 89—98.

[2] Jaakko Järvi, Mat Marcus, Sean Parent, John Freeman и Jacob Smith. “Algorithms for user interfaces”. В: *Proceedings of the eighth international conference on Generative programming and component engineering*. 2009, с. 147—156.

[3] John Freeman, Jaakko Järvi, Wonseok Kim, Mat Marcus и Sean Parent. “Helping programmers help users”. В: *Proceedings of the 10th ACM international conference on Generative programming and component engineering*. 2011, с. 177—184.

[4] Sean Parent. *Adobe Source Libraries*. URL: https://github.com/stlab/adobe_source_libraries (дата обр. 01.04.2025).

[5] Brad Vander Zanden. “An incremental algorithm for satisfying hierarchies of multiway dataflow constraints”. В: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 18.1 (1996), с. 30—72.

[6] Bjorn N Freeman-Benson, John Maloney и Alan Borning. “An incremental constraint solver”. В: *Communications of the ACM* 33.1 (1990), с. 54—63.

[7] Bjorn N Freeman-Benson и John Maloney. “The DeltaBlue algorithm: An incremental constraint hierarchy solver”. В: *1989 Eighth Annual International Phoenix Conference on Computers and Communications*. IEEE Computer Society. 1989, с. 538—539.

[8] B. Neveu. G. Trombettoni. “Computational complexity of multi- way, dataflow constraint problems”. В: *In IJCAI (1)*. 1997, с. 358—365.

[9] Michael Sannella, John Maloney, Bjorn Freeman-Benson и Alan Borning. “Multi-way versus one-way constraints in user interfaces: Experience with the deltablue algorithm”. В: *Software: Practice and Experience* 23.5 (1993), с. 529—566.

[10] John Harold Maloney. *Using constraints for user interface construction*. University of Washington, 1991.

[11] Tetsuya Suzuki и Takehiro Tokuda. “A repeated-update problem in the DeltaBlue algorithm”. В: *Constraints* 3 (1998), с. 331—341.

[12] Michael Sannella. “Skyblue: A multi-way local propagation constraint solver for user interface construction”. B: *Proceedings of the 7th annual ACM symposium on User interface software and technology*. 1994, c. 137—146.