Дата последней сборки документа:
23 июня 2022 г. в 18:33

Общие и специальные вопросы оптимизации

Подвойский А.О.

Краткое содержание

Содержание

1. Полезные ссылки

https://github.com/ceandrade/brkga_mip_feasibility

2. Конспект статьи Gamrath, G. Structure-driven fix-and-propagate heuristics ofr MIP

DINS, RINS, RENS, Crossover и Analytic Center Search определяют свои окрестности с помощью фиксации переменных. LNS-эвристики, базирующиеся на приеме фиксации переменных,

страдают от внутренних конфликтов: исходное пространство поиска должно быть значительно сжато (reduced); таким образом, приходится фиксировать большое количество переменных. Но при этом, чем больше переменных фиксируется, тем выше вероятность, что подпроблема не содержит улучшающего решения или вообще дает недопустимое решение.

Клика (clique) – это множество $\mathcal{C}$ бинарных переменных, в котором не более чем одна переменная может быть выставлена в *единицу*. Клика может быть представлена как линейное неравенство $\sum_{i \in \mathcal{C}} x_i \leqslant 1$.

По аналогии *обращенная клика* (negated clique) – это множество бинарных переменных, в котором не более чем одна переменная может быть выставлена в *ноль*.

В современных MIP-решателях множества выявляенных клик собираются в так называемую *таблицу клик* (clique table). Эта таблица используется компонентами решателя, например, для создания секущих (cuts) или для построения более интенсивного снижения размерности задачи на шаге пресолвинга и распространения.

Блокировки переменных (variable locks) определяются непосредственно по матрице ограничений и показывают сколько ограничений могут быть заблокированы при увеличении или уменьшении значения переменной. В случае MIP с $\leqslant$-ограничениями число ξ_i^+ *верхних блокировок* (up-locks) переменной x_i – это число ограничений, в которых эта переменная имеет положительный коэффициент a_{ri}. А число ξ_i^- *нижних блокировок* (down-locks) – это число ограничений с отрицательным значением коэффициента переменной.

The cliuqe-driven-fix-and-propagate heuristic. Если положить только одну бинарную переменную равной единице, то распространение домена зафиксирует оставшиеся бинарные переменные клики в 0, но кроме это еще и применит много различных изменений домена, сообразно соответствующим переменным. Выбирая для фиксации в единицу самую «дешевую» переменную, т.е. переменную с наименьшим коэффициентом в целевой функции, мы стремимся получить решение с низким значением целевой функции.

Важно: поиск новых существующих решений часто наиболее эффективен в коренвом узле, когда первичная эвристика может непосредственно привести к глобальным фиксациям, более жестким секущим (tighter cutting planes) и лучшим начальным решениям о ветвлении.

3. Конспект диссертации Ахтерберга по CIP

3.1. Базовые определения

3.2. Mixed Integer Programs

Задачи SAT относятся к классу *NP-полных*, а BP, IP и MIP – к классу *NP-трудных* [1, стр. 11].

4. Методы решения задач линейного программирования

Задачи линейного программирования относятся к подклассу задач выпуклого программирования [4, стр. 57].

4.1. Симплекс-метод Данцига

Симплекс-метод Данцига предназначен для решения задач линейного программирования в релаксированной постановке (то есть без учета ограничений на целочисленность переменных) [1, стр. 13].

Замечание

В худшем случае симплекс-метод работает за экспоненциальное время, но в целом на практике обычно он работает очень быстро, и многочисленные эксперименты и исследования метода подтвердили полиномиальное время работы [4, стр. 61]

Для оценки вычислительной сложности симплекс-метода Данцига Спилман и Тенг предложили использовать *сглаженный анализ*. Сглаженный анализ – вероятностный анализ алгоритма, при котором изучается работа алгоритма при незначительном случайном возмущеннии конкретных входных данных, а затем ищется зависимость производительности алгоритма от размера входа и от среднеквадратичного отклоненеия возмощений.

Так вот Спилман и Тенг показали, что симплекс-метод имеет *полиномиальную сглаженную сложность*[1]. Эти результаты означают, что хотя и существуют задачи, на которых симплекс-метод будет работать *экспоненциально* долго, но если исходные данные таких задач (коэффициенты целевой функции и ограничений) подвергнуть незначительному изменению, то с достаточно высокой вероятностью симплекс-метод на возмущенной задаче уже будет работать за *полиномиальное время* [4, стр. 62].

В конце 1970-х годов обнаружили, что алгоритмы, в общем случае решающие задачи линейного программирования за полиномиальное время, все-таки существуют. Все такие полиномиальные алгоритмы – *методы внутренней точки* (первый метод внутреней точки с полиномиальной сложностью – метод Кармаркара) и *метод эллипсоидов* – отличались от симплекс-метода геометрическим подходом.

В течение 50 лет оставался открытым вопрос, существует ли полиномиальный алгоритм, который работает подобно симплекс-методу, перебирая только вершины (угловые точки) допустимого множества задачи. Ответ на этот вопрос дали Келнер и Спилман в 2006 году: они представили рандомизированный симплекс-метод с *полиномиальным временем работы*.

Замечание

В современных оптимизационных пакетах задачи линейного программирования средней размерности (вплоть до сотен тысяч и даже миллиона переменных или ограничений) решаются *методами внутренней точки*.

Постановка задачи. Найти максимум функции

$$f(x) = \sum_{j=1}^{n} c_j x_j \leftarrow c^T x$$

[1] В том же 2006 году оценка симплекс-метода Спилмана и Тенга была улучшена. Р. Вершининым. Он показал, что ожидаемое время работы симплекс-метода на незначительно измененных входных данных является полиномом от логарифма количества ограничений n

при ограничениях

$$\sum_{j=1}^{n} a_{ij}x_j = b_i,\ i = 1, \ldots, m\ (m < n) \leftarrow Ax = b,$$

$$x_j \geqslant 0,\ j = 1, \ldots, n.$$

Такая постановка называется *канонической*, а искомое решение $x^* = (x_1^*, \ldots, x_n^*)^T$ – *оптимальным*.

Замечания:

- Максимизируемая функция и ограничения линейны по $x_j,\ j = 1, \ldots, n$,
- Задача содержит ограничения на неотрицательность переменных, присутствие которых диктуется процедурой симплекс-метода. Если по физической постановке задачи какая-либо переменная, например, x_n, неограничена по знаку, то ее можно представить в виде $x_n = x_{n+1} - x_{n+2}$, где $x_{n+1}, x_{n+2} \geqslant 0$,
- В ограничениях $\sum\limits_{j=1}^{n} a_{ij}x_j = b_i,\ i = 1, \ldots, m\ (m < n)$ будем считать переменные $b_i \geqslant 0,\ i = 1, \ldots, m$.

Стратегия метода Данцига решения описаной задачи основана на особенностях постановки этой задачи. Множество

$$X = \{x \mid \sum_{i}^{n} a_{ij}x_j = b_i,\ i = 1, \ldots, m,\ x \in \mathbb{R}^n,\ x_j \geqslant 0,\ j = 1, \ldots, n\}$$

допустимых решений задачи – есть выпуклое множество, которое геометрически представляет собой *выпуклый политоп*[2], имеющий конечное число *крайних точек*.

Крайней точкой выпуклого множества X называется точка $x \in X$, которая не может быть выражена в виде выпуклой комбинации других точек $y \in X, x \neq y$.

Стратегия решения задачи симплекс-методом состоит в направленном переборе базисных решений, определяющих крайние точки политопа. Направленность перебора предполагает следующую организацию вычислительного процесса [3]:

1. Нахождение базисного решения (метод Гаусса-Жордана, переход к M-задаче),
2. Переход от одного базисного решения к другому таким образом, чтобы обеспечить улучшение целевой функции (другими словами, переход от одной *вершины политопа* к другой в направлении улучшения целевой функции).

4.2. Двойственный симплекс-метод

Теорема двойственности подсказывает, что симплекм-метод, примененный к двойственной задаче, может дать и решение прямой задачи (по крайней мере, если обе ЗЛП имеют допустимые векторы). Различные реализцаии этой идеи приводят к семейству процедур, объединяемых под названием «двойственный симплекс-метод».

4.3. Метод барьеров (метод внутренней точки)

Метод барьеров (также называемый *методом внутренней точки* или *методом внутренних штрафов*) является методом решения условных задач оптимизации с ограничениями вида нера-

[2]Политоп – подмножество Евклидова пространства, представимое объединением симплексов (n-мерное обобщение треугольника)

венств

$$\min_x f(x), \qquad g_i(x) \leqslant 0,\ i = 1, \ldots, m,$$

где $f, g_1, \ldots, g_m : \mathbb{R}^n \to \mathbb{R}$ – дважды непрерывно дифференцируемые функции, $x \in \mathbb{R}^n$ – целевая переменная.

Основная идея метода барьеров заключается в сведении *исходной условной задачи* к последовательности специальным образом построенных *безусловных задач*, решения которых сходятся к решению исходной условной задачи. Это делается с помощью барьерной функции.

Пусть Q – множество, задаваемое функциональными ограничениями $g_1, \ldots, g_m$

$$Q := \{x \in \mathbb{R}^n : g_i(x) \leqslant 0 \quad \forall\ 1 \leqslant i \leqslant m\}.$$

Барьерной функцией (или *функцией внутренних шрафов*) для множества Q называется любая функция $F : \operatorname{int} Q \to \mathbb{R}$, определенная на внутренности множества Q и являющаяся достаточно гладкой, которая обладает барьерным свойством: $F(x) \to +\infty$ при приближении x к границе множества Q изнутри.

Имея в распоряжении барьерную функцию, определим для каждого $t \geqslant 0$ вспомогательную функцию $f_t : \operatorname{int} Q \to \mathbb{R}$ по формуле

$$f_t(x) := tf(x) + F(x)$$

Оказывается, что при $t \to +\infty$ точка $x^*(t)$ сходится ко множеству решений исходной условной задачи. Обозначим $x^*(t) := \arg\min_{x \in \operatorname{int} Q} f_t(x)$. Множество точек $\{x^*(t) : t \geqslant 0\}$ называется *центральным путем*.

Общая схема метода барьеров:

1. Выбрать начальное приближение $x_0 \in \operatorname{int} Q$ и начальный параметр $t_0 > 0$.
2. На каждой итерации $k \geqslant 0$:
   (a) Вычислить точку

$$x_{k+1} := \arg\min_{y \in \operatorname{int} Q} f_{t_k}(y),$$

   используя некоторый *метод безусловной оптимизации* с начальной точкой x_k.
   (b) Увеличить параметр t: выбрать $t_{k+1} > t_k$ (чтобы в итоге $t_k \to +\infty$)

Заметим, что задача оптимизации, возникающая на каждой итерации метода барьеров в реальности является «*безусловной*», несмотря на присутствие в этой задаче условия $x \in \operatorname{int} Q$. Действительно, за счет барьерного свойства[3] целевая функция f_t стремится к $+\infty$ при приближении к границе множества Q, поэтому никакой «разумный» метод оптимизации достаточно близко к границе множества Q никогда не подойдет, и, с точки зрения самого метода, целевая функция как бы задана всюду. Тем не менее, здесь необходима определенная аккуратность.

Например, пусть для решения вспомогательной задачи (задача безусловной оптимизации) используется метод спуска: сначала $y_0 = x_k$, далее на каждой итерации $l \geqslant 0$ строится направление

[3] Вернее за счет барьерного свойства барьерной функции

спуска $d_l \in \mathbb{R}^n$ для функции f_{t_k} в точке y_l, и выполняется шаг

$$y_{l+1} := y_l + \alpha_l d_l,$$

где $\alpha_l \geqslant 0$ – длина шагов. Поскольку метод барьеров по построению всегда работает с внутренними точками множества Q (отсюда и альтернативное название), то для всех достаточно маленьких длин шагов α_l новая точка y_{l+1} будет принадлежать множеству Q, а значение функции f_{t_k} в этой точке уменьшится по сравнению с текущей точкой y_l. Однако, если длина шага α_l выбрана слишком большой, то точка y_{l+1} может вообще оказаться за пределами множества Q – такое вполне может произойти при подборе шага в стандартном алгоритме линейного поиска. Чтобы избежать этой проблемы, стандартную процедуру линейного поиска необходимо модифицировать – запретить ей вычислять функцию в точках за пределами множества Q.

4.3.1. Полная спецификация метода барьеров

Несмотря на то, что гипотетически в методе барьеров можно использовать *любую* комбинацию *барьерной функции* и *метода безусловной оптимизации*, наиболее эффективной (как с точки зрения теория, так и практики) оказывается связка *логарифмического барьера* и *метода Ньютона* (имеет локальную квадратичную сходимость).

Логарифмический барьер в общем виде имеет вид

$$F(x) := -\sum_{i=1}^{m} \ln(-g_i(x))$$

В модели LASSO, как и в любой модели линейной регрессии, прогнозирование выполняется с помощью линейной комбинации компонент вектора a (вектор признаков) с некоторыми фиксированными коэффициентами $x \in \mathbb{R}^n$ (это параметры модели)

$$b(a) := \langle a, x \rangle$$

Коэффициенты x (параметры модели) настраиваются с помощью решения следующей оптимизационной задачи

$$\varphi(x) := \frac{1}{2}\sum_{i=1}^{m}(\langle a_i, x\rangle - b_i)^2 + \lambda \sum_{j=1}^{n} |x_j| := \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1 \to \min_{x\in\mathbb{R}^n} \tag{1}$$

Здесь $\lambda > 0$ – коэффициент регуляризации (параметр модели). Особенностью LASSO является использование имеено l_1-регуляризации. Такая регуляризация позволяет получить разреженное решение. В разреженном решении x^* часть компонент равна нулю. Нулевые веса соответствуют исключению соответствующих признаков из модели (признание их неинформативности).

Двойственной задачей к задаче (1) является

$$\max_{\mu\in\mathbb{R}^n} \left[-\frac{1}{2}\|\mu\|_2^2 - \langle b, \mu\rangle : \|A^T\mu\|_\infty \leqslant \lambda \right]$$

Таким образом, имея в распоряжении допустимую двойственную точку $\mu \in \mathbb{R}^n$, т.е. такую что $\|A^T\mu\|_\infty \leqslant \lambda$, можно вычислить следующую оценку для невязки в этой задаче

$$\varphi(x) - \varphi^* \leqslant \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1 + \frac{1}{2}\|\mu\|_2^2 + \langle b, \mu\rangle =: \eta(x, \mu) \tag{2}$$

Величина $\eta(x, \mu)$ называется зазором двойственности и обращается в ноль в оптимальных решениях (x^* и μ^* задач (1) и (2)). Заметим, что решения x^* и μ^* связаны между собой следующим соотношением: $Ax^* - b = \mu^*$.

Поэтому для фиксированного x естественным выбором соответствующего μ будет

$$\mu(x) := \min\left(1, \frac{\lambda}{\|A^T(Ax - b)\|_\infty}\right)(Ax - b)$$

Такой выбор обеспечивает стремление зазора двойственности $\eta(x, \mu(x))$ к нулю при $x \to x^*$, что позволяет использовать условие $\eta(x, \mu(x)) < \varepsilon$ в качестве критерия остановки в любом итерационном методе решения задачи (1).

Выполнив эквивалентное преобразование задачи через надграфик, задачу (1) можно переписать в виде гладкой условной задачи

$$\min_{x,u}\left(\frac{1}{2}\|Ax - b\|_2^2 + \lambda\,\langle 1_n, u\rangle\right) \qquad x \preceq u,\ x \succeq -u,$$

где $x, u \in \mathbb{R}^n$ и $1_n := (1, \ldots, 1) \in \mathbb{R}^n$. Эта задача является задачей квадратичного программирования, и ее можно решать методом барьеров.

5. Методы решения задач линейного целочисленного программирования

5.1. Общие положения

В случае метода ветвей-и-границ, основанного на решении задачи линейного программирования (LP-based Branch-and-Bound for MIP) порядок действий следующий:

- Сначала мы решаем исходную задачу линейного программирования (original problem) в релаксированной постановке (LP relaxation). Результатом может быть
  1. задача линейного программирования (LP) неограничена (unbounded), следовательно задача линейного программирования в частично-целочисленной постановке (MIP) либо тоже неограничена, либо недопустима (infeasible)
  2. LP недопустима, следовательно недопустима и MIP
  3. получено допустимое решение MIP, следовательно это оптимальное решение для MIP
  4. получено оптимальное решение для LP, которое является недопустимым для MIP

В первых трех случаях мы заканчиваем. В последнем случае мы должны ветвиться.

Замечание

Ключ к успеху останавливаться рано и часто. Другими словами, надо уметь быстро идентифицировать потенциально слабое решение и не тратить на него много времени

Замечание об эвристиках:

- Решение задачи линейного программирования в релаксированной постановке может интерпретироваться как эвристика. Часто не способна найти допустимое решение
- Rounding/Diving:
  - Rounding: Округляет нецелочисленные значения целочисленных переменных,
  - Diving: фиксирует только одну целочисленную переменную, решает LP
- Метаэвристики – имитация отжига, поиск Табу, генетические алгоритмы и пр.
- Optimization-based эвристики:
  - RINS, локальное ветвление
- Доменно-ориентированные эвристики. Часто оказываются очень полезны

Один из способов снизить временные издержки заключается в том, чтобы постараться сделать дерево поиска как можно более компактным

Стратегия ветвления (Branching Rule) очень важна.

Strong Branching на самом деле решает LP задачу в релаксированной постановке для каждой подпроблемы для каждой потенциальной переменной ветвления. Pseudo-costs приближает изменения границы на основе предыдущей информации, собранной при построении дерева ветвей-и-границ.

Эвристики округления [2]:

- Simple Rounding: всегда возвращает допустимое решение,
- Rounding: может нарушать ограничения,
- Shifting: может «освобождать» (unfix) целочисленные переменные,
- Integer Shifting: решает LP-задачу.

Алгоритм RENS – An LNS (Large Neighborhood Search) Rouding Heuristic:

1. $\bar{x} \leftarrow$ оптимальное решение задачи в релаксированной постановке,
2. Фиксирует все целочисленные переменные,
3. Урезает область,
4. Решает получившуюся подпроблему MIP.

Алгоритм Feasibility Pump (Fischetti, Lodi et al.):

1. Решаем LP-задачу
2. Округляем оптимальное решение LP-задачи
3. Если полученное решение допустимо, останавливаемся. В противном случае изменяем цель и возвращаемся к пункту 1

Советы:

- Использовать ограничения (limits) для снижения размерности задачи,
- Diving: стараться использовать округление на лету,
- Отадавать предпочтение бинарным переменным перед целочисленными.

5.2. Метод ветвей и границ

Постановка задачи (Mixed Integer Linear Programming, MILP)

$$f(x) = \sum_{j=1}^{n} c_j x_j \to \max$$

при ограничениях

$$\sum_{j=1}^{n} a_{ij}x_j \geqslant b_i,\ i = 1, \ldots, m,$$
$$x_j \geqslant 0,\ x \in \mathbb{Z},\ j = 1, \ldots, n.$$

Замечание

Описанная задача является задачей линейного целочисленного программирования. Ограничения, связанные с целочисленностью, могут быть наложены не на все переменные, а лишь на их часть

Стратегия поиска

Для корня дерева ветвей-и-границ (branch-and-bound tree) описанная задача решается *симплекс-методом* без учета ограничений на целочисленность (т.е. в релаксированной постановке). Считается, что она имеет решение. На полученном оптимальном решении $x^{0*} = (x_1^{0*}, \ldots, x_n^{0*})$ вычисляется значение *целевой функции* $f(x^{0*})$.

Если решение x^{0*} является целочисленным, то поставленная задача решена. Если решение x^{0*} оказывается нецелочисленным, то значение $f(x^{0*})$ (полученное для решаемой задачи в релаксированной постановке) является *верхней границей* (потому что мы решаем задачу на максимум) возможных оптимальных значений $f(x)$ на целочисленных решениях.

При нецелочисленном решении дальнейшая процедура решения задачи состоит в ее ветвлении на две подзадачи. Целью этого ветвления является разбиение множества допустимых решений на два подмножества путем построения дополнительных ограничений таким образом, чтобы исключить нецелочисленную точку x^{0*} и сделать решение по крайней мере одной из задач целочисленным по одной выбранной координате x_k.

Координатой x_k может быть [3, стр. 339]:

1. Нецелочисленная координата с наименьшим или наибольшим индексом.
2. Нецелочисленная координата с наименьшей или наибольшей дробной частью.
3. Нецелочисленная координата, которой соответствует наибольший коэффициент в целевой функции.
4. Нецелочисленная координата, выбранная на основани приоритетов, определяемых физическим содержанием задачи.

Для построения дополнительных ограничений округляем нецелочисленное решение вниз и исследуем область значений левее, т.е. $x_k \leqslant \lfloor x_k^{0*} \rfloor$, и округляем вверх и исследуем область правее, т.е. $\lceil x_k^{0*} \rceil \leqslant x_k$.

Построение дополнительных ограничений позволило исключить из рассмотрения оптимальное нецелочисленное решение x^{0*} и обеспечить целочисленность значений координаты x_k.

Задачи ЗЛП-1 и ЗЛП-2 записываются в виде, изображенном на рис. 2.

Задачи ЗЛП-1 и ЗЛП-2 решаются самостоятельно симплекс-методом *без учета ограничений на целочисленность* координта $x_j,\ j = 1, \ldots, n$. Вычисляются значения функции $f(x)$ на оптимальных решениях обеих задач. Если ни одна из них не имеет целочисленного решения, то выбрирается задача для приоритетного дальнейшего ветвления по установленному правилу: например, приоритетному ветвлению подлежит та задача, в которой значение $f(x)$ на оптимальном *нецелочисленном* решении максимально.

Пусть $f(x^{1*}) > f(x^{2*})$, тогда задача ЗЛП-1 первой ветвится на ЗЛП-3 и ЗЛП-4, которые решаются симплекс-методом *без учета требований на целочисленность* с последующим анализом

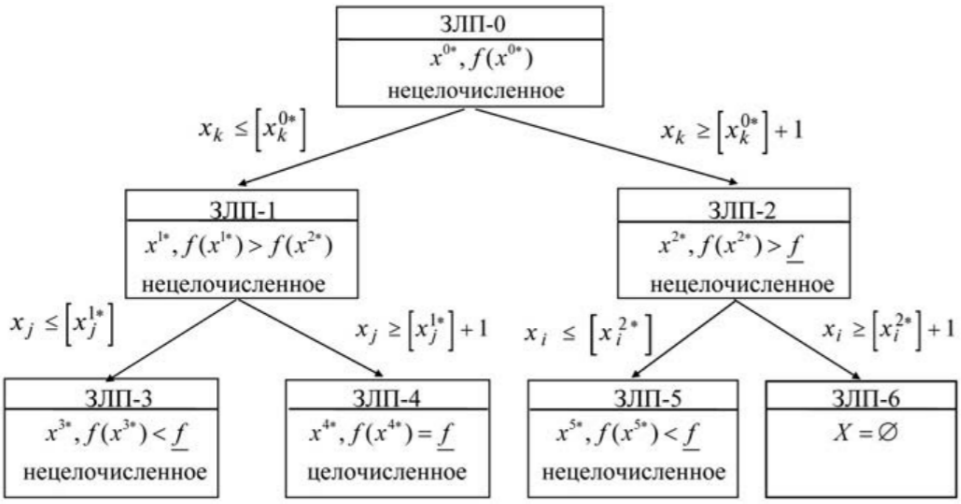

Рис. 1. Пример дерева ветвей-и-границ

ЗЛП-1

$$f(x) = \sum_{j=1}^{n} c_j x_j \to \max;$$

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, \ \ i = 1, \ldots, m;$$

$$x_k \le \left[x_k^{0*}\right];$$

$$x_j \ge 0, \ \ j = 1, \ldots, n;$$

ЗЛП-2

$$f(x) = \sum_{j=1}^{n} c_j x_j \to \max;$$

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i, \ \ i = 1, \ldots, m;$$

$$x_k \ge \left[x_k^{0*}\right] + 1;$$

$$x_j \ge 0, \ \ j = 1, \ldots, n.$$

Рис. 2. Подзадачи корневого узла дерева ветвей-и-границ

решений [3, стр. 340]. Если ни одна из задач ЗЛП-3 и ЗЛП-4 не имеет целочисленного решения, приступают к ветвлению задачи ЗЛП-2.

Процесс ветвления продолжается до тех пор, пока не будет получено в одной из ветвей целочисленное решение. Пусть задача ЗЛП-4 имеет целочисленное решение. Обозначим $\underline{f}$ – значение функции на первом целочисленном решении: $\underline{f} = f(x^{4*})$. Соответсвующее целочисленное решение включается в множество $\bar{X}^*$ возможных оптимальных решений исходной задачи.

После того, как найдено первое целочисленное решение, вопрос о дальнейшем ветвлении других задач решается на основании сравнения значений $f(x^{k*})$ на оптимальных нецелочисленных решениях в оставшихся ветвях со значением $\underline{f}$.

Если $f(x^{k*}) \leqslant \underline{f}$ для всех оставшихся k, то расчет закончен. Решениями исходной задачи являются те целочисленные решения x^{k*}, для которых $f(x^{k*}) = \underline{f}$.

Если $f(x^{k*}) > \underline{f}$, то соответствующая этому номеру k задача ветвится далее. Так, на рис. 1 имеем $f(x^{2*}) > \underline{f}$ и $f(x^{3*}) < \underline{f}$. Задача ЗЛП-2 подлежит ветвлению на ЗЛП-5, ЗЛП-6, а задача ЗЛП-3 не подлежит. Задача ЗЛП-6 не имеет решения, так как множество допустимых решений пустое, и далее не рассматривается. Задача ЗЛП-5 имеет нецелочисленное решение $x^{5*}, f(x^{5*})$. Если $f(x^{5*}) < \underline{f}$, то решение задачи закончено и $x^* = x^{4*}, f(x^*) = \underline{f}$. В противном случае задача ЗЛП-5 ветвится дальше.

Если в одной из задач получено целочисленное решение, то ее ветвление далее не прозводится. Если соответствующее значение целевой функции $\geqslant \underline{f}$, решение считается принадлежащим множеству X^* возможных оптимальных решений исходной задачи.

Если значение целевой функции $< \underline{f}$, целочисленное решение не включается в множество X^*.

Таким образом, ветвление какой-либо задачи заканчивается, если выполняется одно из условий [3]:

1. решение целочисленное,
2. значение целевой функции данной задачи $\leqslant \underline{f}$,
3. множество допустимых решений пустое.

Если ветвление всех задач закончено, то в множестве X^* выбирается решение (решения), которому соответствует наибольшее значение целевой функции. Оно является решением исходной задачи. Если множество X^* пустое, то исходная задача не имеет решения.

5.3. Пресолверы, пропагаторы и прайсеры

Пресолвинг предназначен для:

- снижения размерности задачи,
- усиливания релаксации LP-задачи,
- повышения численной стабильности релаксации LP-задачи,
- извлечения полезной информации.

Пропагаторы предназначены для:

- локального упрощения модели,
- улучшения двойственной границы,
- обнаружения недопустимых решений.

Прайсеры (pricing) предназначены для:

- поиска переменных с отрицательными сниженными потерями (negative reduced costs) или доказательства отсутствия последних,
- динамического старения переменных,
-

5.4. Первичные эвристики в SCIP

Алгоритм ветвей-и-границ представляет собой *полную* (complete) процедуру. Это означает, что алгоритм гарантирует оптимальное решение каждой проблемы за конечное время. Однако, это очень дорогой в вычислительном смысле метод и в худшем случае временные издержки будут экспоненйиально зависеть от размерности задачи.

В противоположенность первичные эвристики относятся к *неполным* (incomplete) методам. То есть они пытаются найти допустимое решение приемлемого качества за небольшой промежуток времени. Но они не гарантируют оптимальности решения.

Первичные эвристики в SCIP условно могут быть разделены на 4 категории [1]:

- Эвристики округления (rounding heuristics) пытаются округлять значения переменных релаксированного решения таким образом, чтобы полученное округленное решение оставалось допустимым,
- Глубокие эвристики (diving heurisctics) начинают с текущего релаксированного решения задачи и итеративно фиксируют целочисленные значения целочисленных переменных и решают текующую задачу заново,
- Целевые глубокие эвристики (objective diving heuristics) похожи на глубокие эвристики, но вместо фиксации переменных, они изменяют значения коэффициентов в целевой функции,

- Улучшающие эвристики (improvenment heuristics) исследуют одно и несколько допустимых решений и пытаются построить такое решение, которому отвечает более низкое (в случае задачи минимизации) или более высокое (в случае задачи максимизации) значение целевой функции.

5.4.1. Эвристики округления

RENS

6. Общие положения постановки частично-целочисленного линейного программирования

Задача линейного программирования в частично-целочисленной постановке (Mixed Integer Linear Program, MILP, MIP) записывается в форме

$$\begin{aligned} \min c^T x, \\ Ax = b, \\ x \geqslant 0, \\ x_i \in \mathbb{Z} \quad \forall i \in \mathcal{I} \end{aligned}$$

Задача, когда все переменные являются целочисленными, называется задачей линейного программирования в чистой целочисленной постановке (Pure Integer Linear Program, ILP, IP).

Если все переменные принимаются значения из множества $\{0, 1\}$, то задача называется задачей линейного программирования 0-1 (0-1 linear program).

Включение целочисленных переменных в постановку задачи расширяет возможности моделирования.

Задачи линейного программирования могут быть решены за полиномиальное время *методами внутренней точки* (метод эллипсоида, алгоритм Кармаркара).

Задачи целочисленного программирования относятся к классу NP-трудных:

- на текущий момент не известны алгоритмы, способные решить этот такого рода задачи за полиномиальное время,
- И, вообще говоря, есть мало шансов, что такие алгоритмы когда-нибудь будут найдены.

Релаксированное решение можно получить, сняв ограничения на целочисленность

$$\begin{aligned} \min c^T x, \\ Ax = b, \\ x \geqslant 0. \end{aligned}$$

Замечание

Задачу линейного программирования в частично-целочисленной постановке нельзя решить, просто перейдя от решения задачи в релаксированной постановке с последующем округлением переменных

Например, оптимальным решением задачи линейного программирования в чистой целочисленной постановке будет

$$\begin{aligned} \max x + y \\ -2x + 2y \geqslant 1, \\ -8x + 10y \leqslant 13, \\ x, y \geqslant 0, \\ x, y \in \mathbb{Z} \end{aligned}$$

вектор $(x, y) = (1, 2)$, которому отвечает целевая функция со значением 3.

А релаксированным оптимальным решением будет вектор $(x, y) = (4, 4.5)$ со значением целевой функции 9.5.

Не существует прямого способа перейти от релаксированного решения к целочисленному.

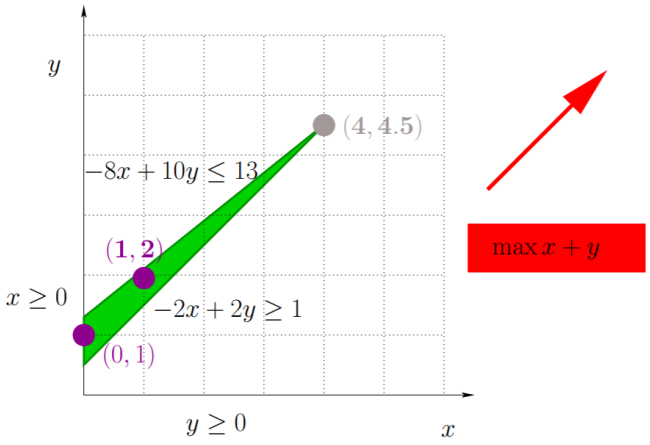

Рис. 3. Связь релаксированной и целочисленной постановок задачи

Предполагается, что переменные ограничены, т.е. имеют нижнюю и верхнюю границы.

Через P_0 обозначим рассматриваемую постановку задачи, а через $LP(P_0)$ релаксированное решение задачи P_0. Если в оптимальном релаксированном решении $LP(P_0)$ все целочисленные переменные принимают целочисленные значения, то это решение будет решением исходной задачи P_0.

В противном случае для целочисленной переменной x_j, которая принимает нецелочисленное значение β_j, $\beta_j \notin \mathbb{Z}$ в оптимальном релаксированном решении $LP(P_0)$, определяются подзадачи

$$\begin{aligned} P_1 := P_0 \wedge x_j \leqslant \lfloor \beta_j \rfloor, \\ P_2 := P_0 \wedge x_j \geqslant \lceil \beta_j \rceil. \end{aligned}$$

Тогда физичным решением исходной задачи будет

$$feasibleSols(P_0) = feasibleSols(P_1) \cup feasibleSols(P_2).$$

7. Presolving

7.1. Probing

Probing это очень затратная[4], но очень мощная техника препроцессинга, которая состоит в последовательной фиксации каждой бинарной переменной в ноль и единицу и вычислению соответствующих подзадач с *помощью техники распространения домена* (domain propagation techniques) [1, стр. 154].

8. Conflict Analysis

В алгоритмах, основанных на дереве ветвей-и-границ и решающих задачи в частично-целочисленной постановке, недопустимость подпроблемы почти всегда возникает либо по причине недопустимости релаксированного решения, либо по причине выхода релаксированного решения за первичную границу (primal bound) [1, стр. 171].

9. Приемы работы с решателем SoPlex

Установить SoPlex можно с помощью менеджера пакетов `conda`

```
conda install -c conda-forge soplex==5.0.2
```

10. Приемы работы с решателем SCIP

Полезный ресурс SCIP FAQ https://www.scipopt.org/doc-8.0.0/html/FAQ.php#howtousemakefiles

Значения полей в таблице результатов (Display Columns) https://www.gams.com/latest/docs/S_SCIP.html#SCIP_gr_table_conflict

10.1. Интерпретация stat-файла SCIP

10.2. Наиболее важные LP-параметры

- `lp/initalgorithm`, `lp/resolvealgorithm`: первичный/двойственный симплекс-метод, метод барьеров с/без кроссовером,
- `lp/pricing`,
- `lp/threads`.

10.3. strbr в логах

Параметр `reoptimization/strongbranchinginit` со значением `True` пытается зафиксировать переменные в коренвом узле перед реоптимизацией пробингом. А стоблец `strbr` в логах показывает общее число вызовов strong branching.

Часто решатель SCIP «залипает» после того, как появляется первые 2 необработнных узла (столбец `left` в логах). И в этой же строке в столбце `strbr` появляется число отличное от нуля, показывающее сколько раз вызывался механизм сильного ветвления

[4] Поэтому она вызывается в самую последнюю очередь, когда все прочие компоненты шага снижения размерности задачи оказались не способными решить свою задачу

```
  ...
  505 constraints of type <linear>
  11 constraints of type <orbitope>
  2857 constraints of type <logicor>
  transformed objective value is always integral (scale: 1)
  Presolving Time: 2.55

  time | node  | left  |LP iter|LP it/n|mem/heur|mdpt |vars |cons |rows |cuts |sepa|confs|strbr|
      dualbound   | primalbound  |  gap   | compl.
  5.1s|     1 |     0 | 10260 |     - |   502M |   0 |  25k|  17k|  17k|   0 |  0 |  99 |   0 |
    0.000000e+00 |      --      |    Inf | unknown
  9.9s|     1 |     0 | 20627 |     - |   514M |   0 |  25k|  17k|  17k|  72 |  1 | 100 |   0 |
    0.000000e+00 |      --      |    Inf | unknown
  12.7s|     1 |     0 | 25403 |     - |   527M |   0 |  25k|  17k|  17k| 128 |  2 | 102 |   0 |
     0.000000e+00 |      --      |    Inf | unknown
  15.0s|     1 |     0 | 28434 |     - |   540M |   0 |  25k|  17k|  17k| 171 |  3 | 105 |   0 |
     0.000000e+00 |      --      |    Inf | unknown
  19.1s|     1 |     0 | 35147 |     - |   544M |   0 |  25k|  17k|  17k| 207 |  4 | 107 |   0 |
     0.000000e+00 |      --      |    Inf | unknown
  26.9s|     1 |     0 | 46650 |     - |   550M |   0 |  25k|  17k|  17k| 264 |  5 | 108 |   0 |
     0.000000e+00 |      --      |    Inf | unknown
  33.5s|     1 |     0 | 56941 |     - |   561M |   0 |  25k|  17k|  17k| 329 |  6 | 109 |   0 |
     0.000000e+00 |      --      |    Inf | unknown
  42.1s|     1 |     0 | 68054 |     - |   567M |   0 |  25k|  17k|  17k| 376 |  7 | 113 |   0 |
     0.000000e+00 |      --      |    Inf | unknown
  44.9s|     1 |     0 | 71433 |     - |   576M |   0 |  25k|  17k|  17k| 443 |  8 | 114 |   0 |
     0.000000e+00 |      --      |    Inf | unknown
  55.2s|     1 |     0 | 82869 |     - |   578M |   0 |  25k|  17k|  17k| 494 |  9 | 117 |   0 |
     0.000000e+00 |      --      |    Inf | unknown
  64.0s|     1 |     0 | 93022 |     - |   586M |   0 |  25k|  17k|  17k| 544 | 10 | 118 |   0 |
     0.000000e+00 |      --      |    Inf | unknown
  68.0s|     1 |     0 | 95673 |     - |   594M |   0 |  25k|  17k|  17k| 583 | 11 | 119 |   0 |
     0.000000e+00 |      --      |    Inf | unknown
  161s|     1 |     2 |196837 |     - |   597M |   0 |  25k|  17k|  17k| 583 | 11 | 131 |  19 |
    0.000000e+00 |      --      |    Inf | unknown  # << NB!
  o4958s|  1066 |   977 |  5406k|4986.5 |objpscos|  85 |  25k|  18k|  17k|2757 |  1 |2168 |4865
    | 0.000000e+00 | 4.710000e+02 |    Inf | unknown
```

10.4. Запуск решателя SCIP с частично-заданным решением

Частично-заданное первичное решение (partial primal solution) описывается в файле с расширением `*.mst`

partial_sol.mst

```
# MIP start
alpha_or_1_35_3_1                                       0.0
...
```

В отличие от `sol`-файла, в котором каждая отсутствующая переменная считается *равной нулю*, в `.mst`-файле (MIP start) все отсутствующие переменные считаются *неизвестными* (unknown)

`.sol` формат		`.mst` формат	
x_1	1	x_1	1
x_4	0.5	x_2	0
x_5	unknown	x_3	0
		x_4	0.5

Формат `sol`-файла SCIP был расширен. В частности, переменная с неизвестным значением помечается как `unknown`. Таким образом, каждая переменная, указанная в `sol`-файле может принимать любое действительное значение, `+/-inf` или `unknown`.

Замечание

Если сначала прочитать *частично-заданное решение*, то потом нельзя будет запустить *процедуру снижения размерности задачи*!

Частично-заданное решение можно получить, например, на «освобожденных» целочисленных переменных (т.е. на целочисленных переменных, с которых сняты ограничения целочисленности).

Другими словами в исходной постановке задачи, влключающей вещественные, целочисленные и бинарные переменные можно снять ограничение целочисленности только с целочисленных переменных (тогда в постановке задачи будут исходно-вещественные переменные, «освобожденные» целочисленные переменные и бинарные переменные) и искать значения вещественных и бинарных переменных.

Получив решение задачи на «освобожденных» целочисленных переменных, в *частично-заданное решение* (`mst`-файл) следует включить только *исходно-вещественные* и *бинарные переменные* (то есть игнорируются значения «освобожденных» целочисленных переменных).

Остается на вход решателю подать полученное частично-заданное решение и исходную постановку задачи. Пример лога

```
CIP version 7.0.3 [precision: 8 byte] [memory: block] [mode: optimized] [LP solver: SoPlex
    5.0.2] [GitHash: 74c11e60cd]
Copyright (C) 2002-2021 Konrad-Zuse-Zentrum fuer Informationstechnik Berlin (ZIB)

External libraries:
SoPlex 5.0.2         Linear Programming Solver developed at Zuse Institute Berlin (soplex.zib.de
    ) [GitHash: e24c304e]
CppAD 20180000.0     Algorithmic Differentiation of C++ algorithms developed by B. Bell (www.
    coin-or.org/CppAD)
ZLIB 1.2.11          General purpose compression library by J. Gailly and M. Adler (zlib.net)
GMP 6.2.1            GNU Multiple Precision Arithmetic Library developed by T. Granlund (gmplib.
    org)
ZIMPL 3.4.0          Zuse Institute Mathematical Programming Language developed by T. Koch (
    zimpl.zib.de)
PaPILO 1.0.2         parallel presolve for integer and linear optimization [GitHash: e567fef]
bliss 0.73.3p        Computing Graph Automorphism Groups by T. Junttila and P. Kaski (http://www
    .tcs.hut.fi/Software/bliss/)
Ipopt 3.14.1         Interior Point Optimizer developed by A. Waechter et.al. (www.coin-or.org/
    Ipopt)

reading user parameter file <scip.set>


read problem <planner_bin_from_scip_337_22.03.lp>
============
```

```
original problem has 859230 variables (155 bin, 173622 int, 0 impl, 685453 cont) and 624637
    constraints
SCIP> read partial_sol_337_bin.mst

read problem <partial_sol_337_bin.mst>
============

unknown variable <#> in line 1 of solution file <partial_sol_337_bin.mst>
(further unknown variables are ignored)
partial primal solution from solution file <partial_sol_337_bin.mst> was accepted as candidate,
    will be completed and checked when solving starts
original problem has 859230 variables (155 bin, 173622 int, 0 impl, 685453 cont) and 624637
    constraints
SCIP> opt

feasible solution found by completesol heuristic after 1893.7 seconds, objective value 3.785961e
    +10
presolving:
(round 1, fast)       328603 del vars, 247760 del conss, 60 add conss, 643374 chg bounds, 3585
    chg sides, 595 chg coeffs, 0 upgd conss, 1 impls, 89 clqs
(round 2, fast)       335330 del vars, 257829 del conss, 61 add conss, 872350 chg bounds, 6393
    chg sides, 3237 chg coeffs, 0 upgd conss, 2 impls, 673 clqs
(round 3, fast)       342239 del vars, 266969 del conss, 63 add conss, 885867 chg bounds, 7415
    chg sides, 4227 chg coeffs, 0 upgd conss, 2 impls, 673 clqs
(round 4, fast)       349855 del vars, 269088 del conss, 63 add conss, 886528 chg bounds, 7488
    chg sides, 4227 chg coeffs, 0 upgd conss, 2 impls, 673 clqs
(round 5, fast)       351214 del vars, 269516 del conss, 63 add conss, 886528 chg bounds, 7522
    chg sides, 4227 chg coeffs, 0 upgd conss, 2 impls, 673 clqs
(round 6, fast)       351626 del vars, 269723 del conss, 63 add conss, 886528 chg bounds, 7544
    chg sides, 4227 chg coeffs, 0 upgd conss, 2 impls, 673 clqs
(1898.6s) running MILP presolver
(1906.0s) MILP presolver (30 rounds): 118425 aggregations, 1308 fixings, 164417 bound changes
(round 7, medium)     471545 del vars, 269901 del conss, 63 add conss, 1050945 chg bounds, 7554
    chg sides, 4227 chg coeffs, 0 upgd conss, 2 impls, 673 clqs
(round 8, fast)       471553 del vars, 389065 del conss, 63 add conss, 1050947 chg bounds, 18250
     chg sides, 4236 chg coeffs, 0 upgd conss, 2 impls, 692 clqs
(round 9, exhaustive) 471570 del vars, 394966 del conss, 63 add conss, 1050947 chg bounds, 18263
     chg sides, 4236 chg coeffs, 0 upgd conss, 2 impls, 692 clqs
(round 10, exhaustive) 471575 del vars, 394992 del conss, 63 add conss, 1052712 chg bounds,
    18263 chg sides, 4236 chg coeffs, 33381 upgd conss, 2 impls, 692 clqs
(round 11, exhaustive) 471600 del vars, 394992 del conss, 63 add conss, 1052712 chg bounds,
    18263 chg sides, 4236 chg coeffs, 34272 upgd conss, 2233 impls, 692 clqs
(1912.6s) sparsify finished: 13/1339222 (0.0%) nonzeros canceled - in total 13 canceled nonzeros
    , 13 changed coefficients, 0 added nonzeros
(1916.4s) probing: 200/12983 (1.5%) - 0 fixings, 0 aggregations, 111 implications, 2 bound
    changes
(1916.4s) probing aborted: 50/50 successive totally useless probings
(1917.4s) symmetry computation started: requiring (bin +, int -, cont +), (fixed: bin -, int +,
    cont -)
(1919.8s) symmetry computation finished: 165 generators found (max: 165, log10 of symmetry group
     size: 82.1)
(1919.8s) no symmetry on binary variables present.
presolving (12 rounds: 12 fast, 5 medium, 4 exhaustive):
473389 deleted vars, 396199 deleted constraints, 63 added constraints, 1052714 tightened bounds,
     0 added holes, 18263 changed sides, 5255 changed coefficients
2375 implications, 692 cliques
presolved problem has 385843 variables (12896 bin, 155651 int, 23 impl, 217273 cont) and 228515
    constraints
34000 constraints of type <varbound>
89 constraints of type <setppc>
```

```
194426 constraints of type <linear>
Presolving Time: 1918.03

time | node  | left  |LP iter|LP it/n|mem/heur|mdpt |vars |cons |rows |cuts |sepa|confs|strbr|
    dualbound   | primalbound  |  gap   | compl.
2540s|     1 |     0 |185787 |     - |  4198M |   0 | 385k| 228k| 228k|   0 |  0 |   0 |   0 |
    2.849245e+10 | 3.785961e+10 |  32.88%| unknown
2895s|     1 |     0 |185787 |     - |  4198M |   0 | 385k| 228k| 228k|   0 |  0 |   0 |   0 |
    2.849245e+10 | 3.785961e+10 |  32.88%| unknown
...
```

10.5. Unresolved numerical troubles in LP

Если верить переписке https://listserv.zib.de/pipermail/scip/2015-July/002463.html, то предупреждение типа «(node XXXX) unresolved numerical troubles in LP XXXXX» часто появляется в контексте задач с выской вычислительной сложностью (например, при решении задачи в релаксированной постановке MINLP) и не обязательно указывает на проблему.

Обычно это означает, что задача в релаксированной постановке (LP relaxation) не может быть решена с доказанной оптимальностью (в пределах заданых допусков) и потому SCIP должен продолжать ветвиться без подрезки (cutting) в данном узле и информации о релаксированном решении.

Остается надеятся, что проблема будет устранена в следующих узлах. Это может приводить к увеличению размеров дерева поиска, но не к неправильным результатам.

Если появление этого предупреждения не приводит к ошибкам или зависанию решателя, то это предупрежедение можно оставить без внимания.

https://stackoverflow.com/questions/24702747/scip-infeasibility-detection-with-a-minlp

Если возникают проблемы, связанные с вычислительной точностью, то можно попробовать снизить точность решения

```
numerics/feastol = 1e-05 # по умолчанию 1e-06
numerics/epsilon = 1e-07 # по умолчанию 1e-09
numerics/dualfeastol = 1e-06 # по умолчанию 1e-07
```

10.6. Постоение графа импликации бинарных переменных в SCIP

После решения задачи (как минимум после шага пресолвинга) в SCIP можно записать gml-файл *графа импликации бинарных переменных*

```
SCIP> write cliquegraph
```

Полученный gml-файл можно прочитать с помощью библиотеки `networkx`. Обычно, при попытке прочитать «сырой» gml-файл, собранный с помощью SCIP, возбуждается исключение вида «networkx.exception.NetworkXError: edge #36926 (3591->3590) is duplicated».

https://stackoverflow.com/questions/37931040/reading-a-multigraph-in-networkx-from-gml-file

Для того чтобы `networkx` могла корректно обработать эти ребра-«дубли» в gml-файл следует добавить строку «`multigraph 1`» как показано ниже

```
graph
[
    multigraph    1  # <=== NB!
    hierarchic    1
```

```
    directed        1
    node
    [
        id 2370
        label "t_y..."
        graphics
        [
            ...
        ]
    ]
]
```

После чего `networkx` будет переданный граф интерпретировать как *мультиграф* и проблема задублированных ребер снимется

```
import networkx as nx

G = nx.read_gml("./337_bin_cliquegraph.gml")
```

Однако, `networkx` в случае по-настоящему больших графов может использоваться только в качестве простого инструмента локальной валидации отдельно взятых узлов и пр.

Для визуализации больших сложных графов лучше подходит инструмент с открытым исходным кодом Gephi https://gephi.org/users/download/

Если окно Preview в GUI Gephi не отображается, то следует выполнить инструкции https://programmersought.com/article/63944156929/

10.7. Анализ конфликтов

https://www.scipopt.org/doc-3.2.1/html/CONF.php

10.8. Управление процедурой поиска решения

Для визуализации дерева ветвей-и-границ необходимо получить специальные файлы (`*.vbc`, `*.dat` и т.д.)

```
set visual vbcfilename somefilename.vbc
set visual bakfilename somefilename.dat
```

Запись этих файлов (`*.vbc`, `*.dat` и т.д.) здорово просаживает производительность решения! Информацию по дереву разумеется можно собирать, но не в эксплуатационном режиме.

Для того чтобы использовать стратегии *ветвления* (branching) или *выбора узла* (node selection) не по умолчанию, нужно задать наивысший приоритет интересующему элементу

```
SCIP> set branching <name of a branching rule> priority 9999999
SCIP> set nodeselectors <name of a node selector> priority 9999999
```

С помощью команд `display branching` и `display nodeselectors` можно запросить полный список допустимых правил и селекторов, соответственно.

Для того чтобы полностью отключить *эвристику* или *сепаратор*, следует положить частоту `freq` для эвристики и `sepfreq` для сепаратора равной «-1»

```
SCIP> set heuristics <name of a heuristic> freq -1
SCIP> set separators <name of a separator> freq -1
SCIP> set constraints <name of a constraint handler> sepfreq -1
```

Для отключения *пресловера*, следует параметр `maxrounds` положить равным 0

```
SCIP> set presolvers <name of a presolver> maxrounds 0
```

Для того чтобы нужная *эвристика* применялась чаще чем по умолчанию, следует задать более низкое значение частоты (положительное число) и/или увеличить `maxlpiterquot` для глубоких эвристик (diving heuristics) и `nodes` для LNS-эвристик

```
SCIP> set heuristics <name of a heuristic> freq <some value>
SCIP> set heuristic <name of a diving heuristic> maxlpiterquot <some value>
SCIP> set heuristic <name of a LNS heuristic> nodesquot <some value>
```

Для того чтобы нужный *сепаратор* применялся чаще чем по умолчанию, следует увеличить значение параметров `maxroundsroot` и `maxsepacutroot`

```
SCIP> set separating <name of a separator> maxroundsroot <some value>
SCIP> set separators <name of a separator> maxrounds <some value>
```

Еще можно специальным образом использовать свойство симметрии бинарных переменных в MIP

```
SCIP> set misc usesymmetry 1
SCIP> set misc usesymmetry 2
SCIP> set misc usesymmetry 0
```

Под *неявными целочисленными переменными* (impicit interger variables) понимаются переменные, которые гарантированно примут целочисленное значение в каждом оптимальном решении каждой подзадачи после фиксации всех целочисленных переменных. Если все целочисленные переменные допустимой точки (feasible point) принимают целочисленное значение, тогда либо все неявные целочисленные переменные принимают целочисленное значение, либо значение неявной целочисленной переменной может быть целочисленным без ухудшения целевой функции.

Удалить столбцы из логов можно так `set display memused active 0`.

Замечание

Массив переменных, который возвращает `SCIPgetVars()`, сортируется по *типам переменных*. Порядок следующий: бинарные, цилочисленные, неявные целочисленные и, наконец, вещественные

Если решатель «залипает» в подпроблеме или в корне дерева ветвей-и-границ, то можно попробовать ограничить количество итераций симплекс-метода (по умолчанию количество итераций не ограничено)

```
lp/iterlim = 200000
lp/rootiterlim = 555000
```

Список иллюстраций

Список литературы

1. *Achterberg T.* Constraint Integer Programming, 2007

2. *Berthold T.* Primal Hueristics in SCIP, 2007
3. *Пантлеев А. В., Летова Т.А,* Методы оптимизации в примерах и задачах. – СПб.: Издательство «Лань», 2015. – 512 с.
4. *Вороноцова Е.А.* Выпуклая оптимизация. – М.: МФТИ, 2021. – 364 с.
5. *Бурков А.* Машинное обучение без лишних слов. – СПб.: Питер, 2020. – 192 с.
6. *Бизли Д.* Python. Подробный справочник. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 864 с.