

Общие и специальные вопросы оптимизации

Подвойский А.О.

Краткое содержание

1	Полезные ссылки	2
2	Конспект диссертации Ахтерберга по SCIP	2
3	Методы решения задач линейного программирования	2
4	Методы решения задач линейного целочисленного программирования	4
5	Общие положения постановки частично-целочисленного линейного программирования	7
6	Presolving	9
7	Conflict Analysis	9
8	Приемы работы с решателем SCIP	9
	Список иллюстраций	13
	Список литературы	13

Содержание

1	Полезные ссылки	2
2	Конспект диссертации Ахтерберга по SCIP	2
2.1	Mixed Integer Programs	2
3	Методы решения задач линейного программирования	2
3.1	Симплекс-метод Данцига	2
4	Методы решения задач линейного целочисленного программирования	4
4.1	Метод ветвей и границ	4
4.2	Первичные эвристики в SCIP	6
4.2.1	Эвристики округления	7
5	Общие положения постановки частично-целочисленного линейного программирования	7
6	Presolving	9
6.1	Probing	9

7 Conflict Analysis	9
8 Приемы работы с решателем SCIP	9
8.1 Запуск решателя SCIP с частично-заданным решением	9
8.2 Unresolved numerical troubles in LP	10
8.3 Построение графа импликации бинарных переменных в SCIP	10
8.4 Анализ конфликтов	11
8.5 Управление процедурой поиска решения	11
Список иллюстраций	13
Список литературы	13

1. Полезные ссылки

https://github.com/ceandrade/brkg_mip_feasibility

2. Конспект диссертации Ахтерберга по СІР

2.1. Mixed Integer Programs

Задачи SAT относятся к классу *NP-полных*, а ВР, ІР и МІР – к классу *NP-трудных* [1, стр. 11].

3. Методы решения задач линейного программирования

Задачи линейного программирования относятся к подклассу задач выпуклого программирования [3, стр. 57].

3.1. 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 [3, 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 возмущении конкретных входных данных, а затем ищется зависимость производительности алгоритма от размера входа и от среднеквадратичного отклонения возмущений.

Так вот Спилман и Тенг показали, что симплекс-метод имеет полиномиальную сглаженную сложность¹. 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* [3, 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 внутренней 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

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

при ограничениях

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

$$x_j \geq 0, \quad j = 1, \dots, n.$$

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

Замечания:

- Максимизируемая функция и ограничения линейны по x_j , $j = 1, \dots, n$,
- 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 $x_{n+1}, x_{n+2} \geq 0$,
- В ограничениях $\sum_{j=1}^n a_{ij} x_j = b_i$, $i = 1, \dots, m$ ($m < n$) будем считать переменные $b_i \geq 0$, $i = 1, \dots, m$.

¹В 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

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

$$X = \{x \mid \sum_i^n a_{ij}x_j = b_i, i = 1, \dots, m, x \in \mathbb{R}^n, x_j \geq 0, j = 1, \dots, n\}$$

допустимых решений задачи – есть выпуклое множество, которое геометрически представляет собой *выпуклый полигон*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 полигона. Направленность перебора предполагает следующую организацию вычислительного процесса [2]:

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 полигона* к другой в направлении улучшения целевой функции).

4. Методы решения задач линейного целочисленного программирования

4.1. Метод ветвей и границ

Постановка задачи (Mixed Integer Linear Programming, MILP)

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

при ограничениях

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

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 $x^{0*} = (x_1^{0*}, \dots, x_n^{0*})$ вычисляется значение *целевой функции* $f(x^{0*})$.

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

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 [2, 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 основании приоритетов, определяемых физическим содержанием задачи.

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

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

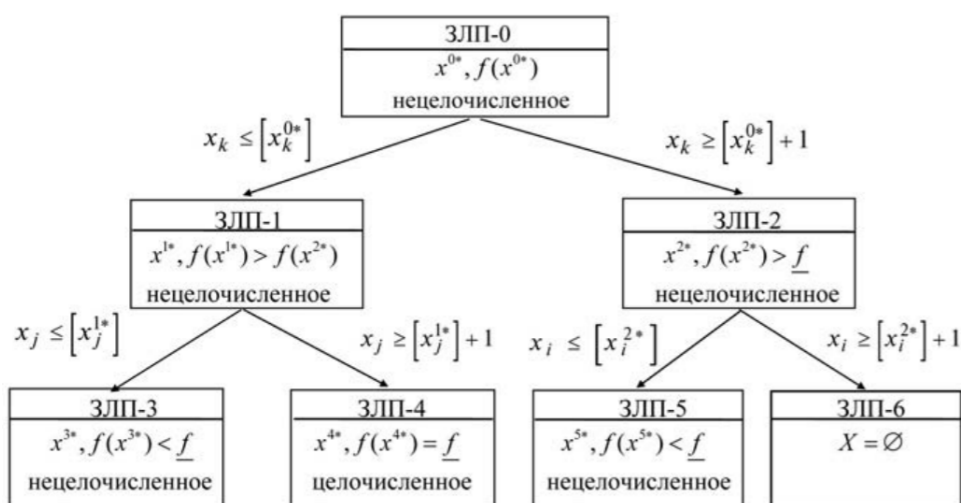


Рис. 1. Пример дерева ветвей-и-границ

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* координат x_j , $j = 1, \dots, n$. Вычисляются значения функции $f(x)$ на оптимальных решениях обеих задач. Если ни одна из них не имеет целочисленного решения, то выбирается 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

<p><u>ЗЛП-1</u></p> $f(x) = \sum_{j=1}^n c_j x_j \rightarrow \max;$ $\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, \dots, m;$ $x_k \leq [x_k^{0*}];$ $x_j \geq 0, \quad j = 1, \dots, n;$	<p><u>ЗЛП-2</u></p> $f(x) = \sum_{j=1}^n c_j x_j \rightarrow \max;$ $\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, \dots, m;$ $x_k \geq [x_k^{0*}] + 1;$ $x_j \geq 0, \quad j = 1, \dots, n.$
--	--

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

решений [2, 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: $\underline{f} = f(x^{4*})$. Соответствующее 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 $f(x^{k*}) \leq \underline{f}$ 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 производится. Если соответствующее значение целевой функции $\geq \underline{f}$, 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 [2]:

1. решение целочисленное,
2. значение целевой функции данной задачи $\leq \underline{f}$,
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.

4.2. 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 экспоненциально зависеть от размерности задачи.

В противоположность 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 heuristics) начинают с текущего релаксированного решения задачи и итеративно фиксируют целочисленные значения целочисленных переменных и решают текущую 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 (improvement 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.

4.2.1. Эвристики округления

RENS

5. Общие положения постановки частично-целочисленного линейного программирования

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

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

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

Если все переменные принимают 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

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

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

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

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.

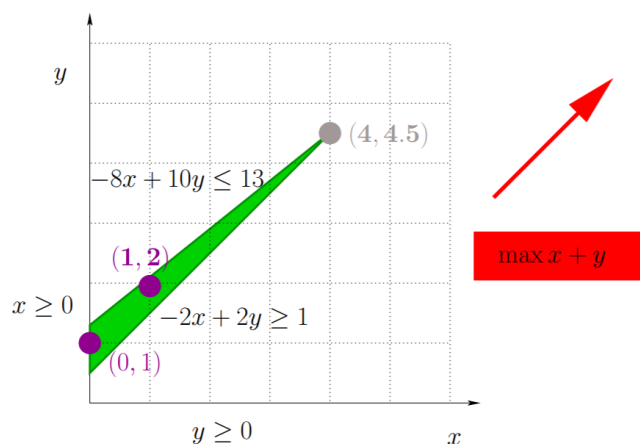


Рис. 3. Связь релаксированной и целочисленной постановок задачи

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

$$P_1 := P_0 \wedge x_j \leq \lfloor \beta_j \rfloor,$$

$$P_2 := P_0 \wedge x_j \geq \lceil \beta_j \rceil.$$

Тогда физическим решением исходной задачи будет

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

6. Presolving

6.1. Probing

Probing это очень затратная³, 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].

7. 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].

8. 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

8.1. 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

³Поэтому она вызывается в самую последнюю очередь, когда все прочие компоненты шага снижения размерности задачи оказались не способными решить свою задачу

Если сначала прочитать *частично-заданное решение*, то потом нельзя будет запустить *процедуру снижения размерности задачи*! Поэтому частично-заданное решение следует строить на подготовленной постановке задачи (т.е. после шага `presolve`)

8.2. 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 высокой 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 заданных допусков) и потому SCIP должен продолжать ветвиться без подрезки (cutting) в данном узле и информации о релаксированном решении.

Остается надеяться, 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 предупреждение 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
```

8.3. Построение 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/>

8.4. Анализ конфликтов

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

8.5. 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 *пресолвера*, 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 separators <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* (implicit integer 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, целочисленные, 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

1	Пример дерева ветвей-и-границ	5
2	Подзадачи корневого узла дерева ветвей-и-границ	6
3	Связь релаксированной и целочисленной постановок задачи	8

Список литературы

1. *Achterberg T.* Constraint Integer Programming, 2007
2. *Пантлеев А. В., Летова Т.А.* Методы оптимизации в примерах и задачах. – СПб.: Издательство «Лань», 2015. – 512 с.

3. *Вороноцова Е.А.* Выпуклая оптимизация. – М.: МФТИ, 2021. – 364 с.
4. *Бурков А.* Машинное обучение без лишних слов. – СПб.: Питер, 2020. – 192 с.
5. *Бизли Д.* Python. Подробный справочник. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 864 с.