Постановка задачи Линейного программирования

<u>Линейное программирование</u> - это раздел прикладной математики, изучающий теорию, приложения и методы решения конечных систем линейных неравенств с конечным числом вещественных неизвестных $x_1, \ldots, x_n$:

$$\left.\begin{array}{rl} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n & \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n & \leq b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n & \leq b_m \end{array}\right\} \qquad (1)$$

или в сокращенной записи $Ax \leq b$. Считаем, что матрица A не содержит нулевых строк a_i. Основная задача ЛП состоит в нахождении такого решения (1), которое максимизирует заданную линейную функцию $< c, x >$ = $c_1x_1 + c_2x_2+...+c_nx_n$ вектора неизвестных x по всем вещественным x, удовлетворяющим системе (1):

$$\max_{x \in \mathbf{R}^n:\ Ax \leq b} \langle c, x \rangle; \qquad (2)$$

озЛП (2) с n неизвестными и m ограничениями называется задачей размерности (n, m) и задается числовой таблицей своих коэффициентов:

$$\left|\begin{array}{ccccc} a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} & b_m \\ c_1 & c_2 & \ldots & c_n & 0 \end{array}\right| \qquad (3)$$

В частном случае $c = (0,\ldots, 0)$ задача (2) эквивалента (1), поэтому умение решать озЛП предполагает умение решать системы линейных неравенств (ЛН). В форме (2) может быть представлена любая задача ЛП с ограничениями равенствами и неравенствами, в том числе ***каноническая задача ЛП:***

$$\max_{Ax=b,\ x\geq\bar{0}} \langle c, x\rangle$$

Определение: Точка $x \in D$ называется угловой точкой, если представление $x = \alpha x^1 + (1 - \alpha)x^2$, где $x^1, x^2 \in D$; $0 < \alpha < 1$ возможно только при $x^1 = x^2$.

Другими словами, невозможно найти две точки в области, интервал проходящий через которые содержит x (т.е. x - не внутренняя точка).

Несмотря на то, что формально задачи ЛП не являются дискретными ($x \in R^n$), их решение нетрудно свести к перебору конечного числа угловых точек на основании *принципа граничных решений*:

Если задача (2) имеет решение, то найдется такая подматрица A_l матрицы A, что любое решение системы уравнений $A_l x = b_l$, т.е.

$$\{a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n = b_i \mid i \in I\}$$

реализует максимум в (2).

Отметим, что для невырожденных A_l решение соответствующей системы уравнений $A_l x = b_l$, удовлетворяющее ограничениям (1), является угловой точкой (1).

Из принципа граничных решений следует, что если угловая точка (1) существует, то разрешимая задача (2) имеет решение и в угловой точке (1), т.е. она эквивалентна максимизации $< c, x >$ на конечном множестве вершин полиэдра (1). Процедура решения системы линейных уравнений методом Гаусса требует не более полинома 3-й степени от $m,\ n$ (точнее, $max(m,n)[min(m,n)]^2$) арифметических операций с элементами A и b. Однако число возможных подматриц матрицы A экспоненциально, и метод полного их перебора не эффективен.

Симплекс-метод

Симплекс-метод позволяет эффективно найти оптимальное решение, избегая простой перебор всех возможных угловых точек. Основной принцип метода: вычисления начинаются с какого-то «стартового» базисного решения, а затем ведется поиск решений, «улучшающих» значение целевой функции. Это возможно только в том случае, если возрастание какой-то переменной приведет к увеличению значения функционала.

Необходимые условия для применения симплекс-метода:

1. Задача должна иметь каноническую форму.
2. У задачи должен быть явно выделенный базис.

Определение: Явно выделенным базисом будем называть вектора вида: $(..0100..)^T$, $(..010..)^T$, $(..0010)^T$ …, т.е. только одна координата вектора ненулевая и равна 1.

Замечание: Базисный вектор имеет размерность (m*1), где m – количество уравнений в системе ограничений.

Для удобства вычислений и наглядности обычно пользуются симплекс-таблицами:

Базис:	x_1	x_2	…	x_n	
x_n	a_{11}	a_{12}	…	a_{1n}	b_1
x_{n-1}	a_{21}	a_{22}	…	a_{2n}	b_2
…	…	…	…	…	…
x_{n-2}	a_{m1}	a_{m2}	…	a_{mn}	b_m
Z	c_1	c_2	…	c_n	0

- В первой строке указывают «наименование» всех переменных.

- В первом столбце указывают номера базисных переменных, а в последней ячейке – букву Z (это строка функционала).
- В «середине таблицы» указывают коэффициенты матрицы ограничений — a_{ij}.
- Последний столбец – вектор правых частей соответствующих уравнений системы ограничений.
- Крайняя правая ячейка – значение целевой функции. На первой итерации ее полагают равной 0.

Замечание: Базис – это переменные, коэффициенты в матрице ограничений при которых образуют базисные вектора.

Замечание: Если ограничения в исходной задаче представлены неравенствами вида $\leq$, то при приведении задачи к канонической форме, введенные дополнительные переменные образуют начальное базисное решение.

Замечание: Коэффициенты в строке функционала берутся со знаком “-”.

Алгоритм симплекс-метода:

1. Выбираем переменную, которую будем вводить в базис. Это делается в соответствии с указанным ранее принципом: мы должны выбрать переменную, возрастание которой приведет к росту функционала. Выбор происходит по следующему правилу:

- Если задача на минимум – выбираем максимальный положительный элемент в последней строке.
- Если задача на максимум – выбираем минимальный отрицательный.

Такой выбор, действительно, соответствует упомянутому выше принципу: если задача на минимум, то чем большее число вычитаем – тем быстрее убывает функционал; для максимума наоборот – чем большее число добавляем, тем быстрее функционал растет.

Замечание: Хотя мы и берем минимальное отрицательное число в задаче на максимум, этот коэффициент показывает направление роста функционала, т.к. строка функционала в симплекс-таблице взята со знаком “-”. Аналогичная ситуация с минимизацией.

Определение: Столбец симплекс-таблицы, отвечающий выбранному коэффициенту, называется **ведущим столбцом**.

2. Выбираем переменную, которую будем вводить в базис. Для этого нужно определить, какая из базисных переменных быстрее всего обратится в нуль при росте новой базисной переменной. Алгебраически это делается так:

- Вектор правых частей почленно делится на ведущий столбец
- Среди полученных значений выбирают минимальное положительное (отрицательные и нулевые ответы не рассматривают)

Определение: Такая строка называется **ведущей строкой** и отвечает переменной, которую нужно вывести из базиса.

Замечание: Фактически, мы выражаем старые базисные переменные из каждого уравнения системы ограничений через остальные переменные и смотрим, в каком уравнении возрастание новой базисной переменной быстрее всего даст 0. Попадание в такую ситуацию означает, что мы «наткнулись» на новую вершину. Именно поэтому нулевые и отрицательные элементы не рассматриваются, т.к.

получение такого результата означает, что выбор такой новой базисной переменной будет уводить нас из области, вне которой решений не существует.

3. Ищем элемент, стоящий на пересечении ведущих строки и столбца.

Определение: Такой элемент называется **ведущим элементом**.

4. Вместо исключаемой переменной в первом столбце (с названиями базисных переменных) записываем название переменной, которую мы вводим в базис.

5. Далее начинается процесс вычисления нового базисного решения. Он происходит с помощью **метода Жордана-Гаусса**.

- Новая Ведущая строка = Старая ведущая строка / Ведущий элемент
- Новая строка = Новая строка – Коэффициент строки в ведущем столбце * Новая Ведущая строка

Замечание: Преобразование такого вида направлено на введение выбранной переменной в базис, т.е. представление ведущего столбца в виде базисного вектора.

6. После этого проверяем условие оптимальности. Если полученное решение неоптимально – повторяем весь процесс снова.

Постановка задачи Дискретного программирования

Рассматривается задача:

$$\min z = \min_{x \in G} \varphi(x), \quad (1)$$

где G – конечное множество.

Задача (1) называется задачей “Дискретного программирования”. В задачах дискретного программирования множество допустимых решений является невыпуклым и несвязным. Поэтому отыскание решения сопряжено со значительными трудностями. В частности, невозможно применение стандартных приемов, состоящих в замене дискретной задачи ее непрерывным аналогом, и в дальнейшем округлении найденного решения до ближайшего целочисленного.

Метод ветвей и границ

Одним из основных методов решения задач дискретного программирования является ***метод ветвей и границ***.

Метод ветвей и границ относится к группе комбинаторных методов и позволяет существенно уменьшить объем перебора вариантов.

В основу метода положены следующие построения:

1. ***Вычисление нижней оценки*** $\xi(G)$ для значения целевой функции на множестве G. Для любого множества G нужно уметь вычислять величину
$$\xi(G) \leq \varphi(x),\ x \in G$$
2. ***Ветвление (разбиение на подмножества)***. Реализация метода связана с постоянным ветвлением множества допустимых решений на дерево подмножеств. Процесс ветвления состоит из некоторых итераций:

ИТЕРАЦИЯ 0.

Полагаем $G^{(0)} = G$. Далее некоторым способом разбиваем исходное множество на систему подмножеств:

$$G^{(0)} = G_1^{(1)} \cup G_2^{(1)} \cup \ \ldots \ \cup G_p^{(1)}, \ \bigcap_{i=1}^{p} G_i^{(1)} = \varnothing.$$

ИТЕРАЦИЯ k. ($k \geq 1$).

Пусть $G^{(k)}_1$, $G^{(k)}_2$, …,$G^{(k)}_r$ - это множества, еще не подвергшиеся ветвлению. По некоторому по некоторому правилу выбираем множество $G^{(k)}_{v(k)}$, и это множество разбивается на подмножества:

$$G_{v(k)}^{(k)} = G_{v(k),1}^{(k)} \cup G_{v(k),2}^{(k)} \cup \ \ldots \ \cup G_{v(k),s}^{(k)}, \ \bigcap_{i=1}^{s} G_{v(k),i}^{(k)} = \varnothing.$$

Дерево подмножеств (процесс ветвления)

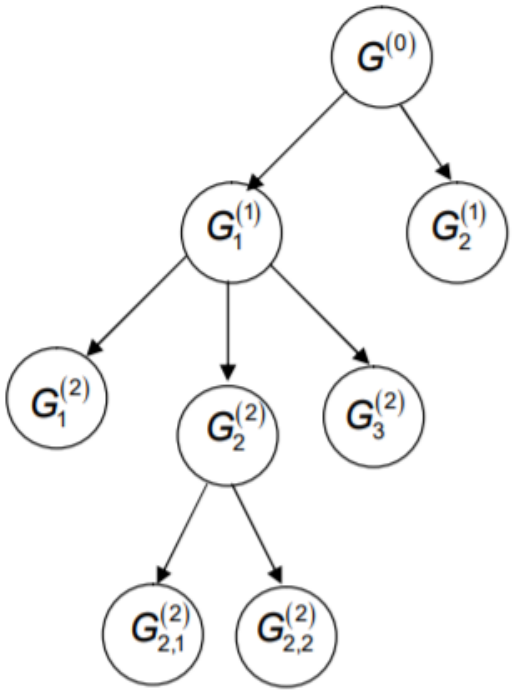

Рис.1

3. ***Границы.*** В процессе ветвления появляются множества, которые не могут быть подвергнуты дальнейшему ветвлению (граничные). Эти множества называются ***концевыми вершинами дерева подмножеств*** (см Рис. 1).

4. ***Нахождение решения***. Для граничных множеств нужно уметь находить допустимые решения, способ зависит от специфики рассматриваемой задачи.
5. ***Пересчет оценок***.
   Если $G_1 \subset G_2$, то $min_{x \in G_1} \varphi(x) \geq min_{x \in G_2} \varphi(x)$. Поэтому оценки не убывают:
   $$\xi(G^i) \geq \xi(G^0)$$
6. ***Признак оптимальности.*** Пусть $G = \bigcup_{i=1}^{s} G_i$, и некоторое решение $x_0 \in G_v$. Если при этом $\varphi(x_0) = \xi(G_v) \leq \xi(G_i)$, G_i- множества, не подвергавшиеся ветвлению, то текущее решение x_0- оптимально.
7. ***Оценка точности приближенного решения***. Пусть $G = \bigcup_{i=1}^{s} G_i$, $\xi = min_i \xi(G_i)$. Пусть x - некоторое решение, тогда $\xi \leq min_x \varphi(x)$ и $\Delta = \varphi(x) - \xi$ - точность.

Алгоритм метода ветвей и границ для задачи целочисленного программирования (ЗЦП)

Рассмотрим задачу:

1. $min\, z = min\, \varphi(x) = min \sum_{j=1}^{n} c_j x_j \quad (5.2.1)$

при ограничениях:

2. $\sum_{j=1}^{n} a_{ij} x_j \geq b_i, i = \overline{1,m}$,
3. $c_j \leq x_j \leq d_j$, $j = \overline{1,n}$,
4. $x_j -$ целые

Задача (1) - (4) называется ***задачей целочисленного программирования.***

ИТЕРАЦИЯ 0.

Шаг 1. Находим множество допустимых решений задачи $G^{(0)}$ без условия целочисленности. Множество $G^{(0)}$ определяется ограничениями (2), (3).

Шаг 2. Находим оптимальное решение X_0 задачи (1) при условиях (2), (3). Если X_0 удовлетворяет и условию (4), то X_0 оптимальное, если нет - переходим к итерации 1.

Шаг 3. Вычисляем оценку $\xi(G^{(0)}) = [\varphi(X_0)]$, где $[a]$ - наименьшее целое, не меньше чем a (округление до ближайшего целого с избытком).

ИТЕРАЦИЯ 1.

Шаг 1. *Ветвление.* Выбираем не целочисленную компоненту решения X_0: $x_i = x_i^0$, $i = \overline{1,n}$. Тогда $G^{(0)} = G_1^{(1)} \cup G_2^{(1)}$, где

$$G_1^{(1)} = \{X | X \in G^{(0)}, x_i \leq [x_i^0]\},$$

$$G_2^{(1)} = \{X | X \in G^{(0)}, x_i \geq [x_i^0] + 1\},$$

где $[a]$ - целая часть a.

Шаг 2. *Решаем задачи ЛП (1)* на множествах $G_1^{(1)}, G_2^{(1)}$. Находим соответствующие решения $X_1^{(1)}, X_2^{(1)}$.

Шаг 3. *Вычисляем оценки:* $\xi(G_1^{(1)}) = [\varphi(X_1^{(1)})], \xi(G_2^{(1)}) = [\varphi(X_2^{(1)})]$.

Шаг 4. *Проверка оптимальности.* Если $X_i^{(1)}$- целочисленное решение и $\xi(G_i^{(1)}) = min\{\xi(G_1^{(1)}), \xi(G_2^{(1)})\}$, то $X_i^{(1)}$- оптимальное решение.

ИТЕРАЦИЯ (k+1).

Пусть проведено k итераций, $G_1^{(k)}, G_2^{(k)}, \dots, G_{r_k}^{(k)}$ - множества, еще не подвергшиеся ветвлению, $\xi(G_i^{(k)})$ - оценки этих множеств, и не найдено оптимальное решение. Тогда перспективное для ветвления множество: $\xi(G_v^{(k)}) = min_i\{\xi(G_i^{(k)})\}$.

Шаг 1. Производим ветвление $G_v^{(k)}$ на подмножества $G_{v1}^{(k)}, G_{v2}^{(k)}$, $G_v^{(k)} = G_{v1}^{(k)} \cup G_{v2}^{(k)}$, где

$$G_{v1}^{(k)} = \{X | X \in G_v^{(k)}, x_s \le [x_s^k]\},$$

$$G_{v2}^{(k)} = \{X | X \in G_v^{(k)}, x_s \ge [x_s^k] + 1\},$$

где x_s^k - нецелочисленная компонента решения $X_v^{(k)}$.

Шаг 2. *Решаем задачи ЛП (1)* на множествах $G_{v1}^{(k)}, G_{v2}^{(k)}$. Находим соответствующие решения $X_{v1}^{(k)}, X_{v2}^{(k)}$.

Шаг 3. *Вычисляем оценки:* $\xi(G_{v1}^{(k)}) = [\varphi(X_{v1}^{(k)})], \xi(G_{v2}^{(k)}) = [\varphi(X_{v2}^{(k)})]$.

Шаг 4. Если $X_{v1}^{(1)}$ удовлетворяет условию целочисленности, то это концевая вершина. Если $\xi(G_{v1}^{(k)}) \le \xi(G_i^{(k)})$ для любого i, то $X_{v1}^{(k)}$- оптимальное решение.

Формулировка Целочисленного программирования для задачи TSP

*В конспекте под словом **“узел”** будет подразумеваться **вершина(город)***

Чтобы сформулировать задачу TSP, как задачу целочисленного программирования необходимо ввести бинарные переменные $x_e \in \{0,1\},\ e \in E$ и $y_i \in \{0,1\},\ i \in V$, где V- это набор узлов, E- набор ребер, чтобы указать содержится ли ребро e или узел i в гамильтоновом туре.

Гамильтоновым циклом является такой цикл (замкнутый путь), который проходит через каждую вершину данного графа ровно по одному разу; то есть простой цикл, в который входят все вершины графа.

Гамильтонов граф (путь) — это граф, содержащий *гамильтонов цикл*.

Задача о гамильтоновом пути и **задача о гамильтоновом цикле** — это задачи определения, существует ли гамильтонов путь или гамильтонов цикл в заданном графе(ориентированном или неориентированном). Обе задачи относятся к классу NP-Complete.

Возможное решение TSP можно рассматривать как цикл с m рёбрами, соединяющий все кластеры и ровно один узел из каждого кластера. Поэтому TSP можно сформулировать как следующую задачу целочисленного программирования:

$$min \sum_{e \in E} c_e x_e$$

при условии

$(1)\ y(V_k) = 1,\ \forall k \in K = \overline{1,m}$

(2) $x(\delta(v)) = y_v,\ \forall v \in V$

(3) $x(E(S)) \leq z(S - i),\ \forall i \in S \subset V,\ 2 \leq |S| \leq n - 2$

(4) $x_e \in \{0, 1\},\ \forall e \in E$

(5) $y_i \in \{0, 1\},\ \forall i \in V$

Здесь использованы следующие обозначения:

1) $x(F) = \sum_{e \in F} x_e,\ F \subseteq E$;

2) $y(S) = \sum_{i \in S} y_i,\ S \subseteq V$;

3) для $S \subseteq V$, ***the cutset*** (вырезка, набор сокращений), обозначаемая $\delta(S)$, обычно определяется следующим образом:

$$\delta(S) = \{e = (i, j) \in E \mid i \in S,\ j \notin S\}$$

4) c_{ij} - стоимость поездки из вершины i в j

В такой формулировке целевая функция четко описывает стоимость оптимального маршрута. **Ограничения (1)** гарантируют, что из каждого кластера выбирается ровно один узел, **ограничения (2**) являются *ограничениями степени*: они указывают на то, что если выбран узел G, то он остается и вводится ровно один раз. **Ограничения (3)** являются *ограничениями на исключение подтура*: они запрещают формирование подтуров, т.е. туров на подмножествах менее чем из n узлов. Из-за ограничений степени, подтуры состоящие из одного узла (и следовательно из $n - 1$ узлов) не могут выполняться. Следовательно, допустимо определить ограничения (3) только для $2 \leq |S| \leq n - 2$. Наконец, ограничения (4) и (5) накладывают бинарные(двоичные) условия на переменные.

Данная формулировка называется ***обобщенной формулировкой исключения подтура***, поскольку ограничения (3) исключают все циклы на подмножествах состоящих менее чем из n узлов.

Поскольку в TSP необходимо посетить ровно один узел из каждого кластера, то можно отбросить внутрикластерные ребра. Ввиду такого сокращения ограничения (2) эквивалентны:

$$x(\delta(V_k)) = 2,\ \forall k \in K = \{1,\dots, m\}$$

и ограничения (3) эквивалентны:

$$x(E(S)) \leq r - 1,\ \forall S = \bigcup_{j=1}^{r} V_j \text{ и } 2 \leq r \leq m - 2$$

Также можно заменить ограничения (3) на удаление подтура ограничениями на подключение, что приведет к ***обобщенной формулировке cutest*** :

$$min \sum_{e \in E} c_e x_e$$

при условии

$(1), (2), (4), (5)$ и

$(6)\ x(\delta(S)) \geq 2(y_i + y_j - 1),\ \forall S \subset V,\ 2 \leq |S| \leq n - 2,\ i \in S,\ j \in S$

Обе формулировки, которые были описаны имеют экспоненциальное число ограничений.

Формулировка целочисленного линейного программирования для задачи CVRP

Пусть $G = (V, H, c)$ полный ориентированный граф с $V = \{0, 1, 2,..., n\}$ - это набор узлов и $H = \{(i, j): i, j \in V, i \neq j\}$ - это набор вершин, состоящий из узла 0, который является депо для парка из p транспортных средств с одинаковой вместимостью Q и остальных n вершин, которые представляю собой географически удаленных клиентов. Каждый клиент $i \in V - \{0\}$ имеет определенный положительный спрос $d_i \leq Q$. Неотрицательная стоимость проезда c_{ij} связана с каждой дугой $(i, j) \in H$. Матрица стоимостей проездов симметрична, т.е. $c_{ij} = c_{ji}$ для всех $i, j \in V,\ i \neq j$ и удовлетворяет треугольному неравенству $c_{ij} + c_{jk} \geq c_{ik}$ для всех $i, j, k \in V$. Минимальное количество транспортных средств, необходимое для обслуживания всех клиентов: $\text{верхняя граница}(\frac{\sum_{i=1}^{n} d_i}{Q})$.

Модель

Двоичная переменная решения x_{rij} определяется для указания того, пересекает ли транспортное средство $r,\ r \in \{1, 2,..., p\}$ дугу (i, j) в оптимальном решении. Модель целочисленного линейного программирования для задачи CVRP может быть записана как:

Минимизировать

$$(1)\ \sum_{r=1}^{p} \sum_{i=0}^{n} \sum_{j=0, i\neq j}^{n} c_{ij} x_{rij}$$

При условии

$$(2)\ \sum_{r=1}^{p} \sum_{i=0, i\neq j}^{n} x_{rij} = 1,\ \forall j \in \{1,..., n\},$$

(3) $\sum_{j=1}^{n} x_{r0j} = 1,\ \forall r \in \{1,\dots,p\},$

(4) $\sum_{i=0,i\neq j}^{n} x_{rij} = \sum_{i=0}^{n} x_{rji},\ \forall j \in \{0,\dots,n\}, \forall r \in \{1,\dots,p\},$

(5) $\sum_{i=0}^{n} \sum_{j=1,i\neq j}^{n} d_j x_{rij} \leq Q,\ \forall r \in \{1,\dots,p\},$

(6) $\sum_{r=1}^{p} \sum_{i\in S} \sum_{j\in S,\, i\neq j} x_{rij} \leq |S| - 1,\ \forall S \subseteq \{1,\dots,n\}$

(7) $x_{rij} \in \{0,1\},\ \forall r \in \{1,\dots,p\},\ i,j \in \{0,\dots,n\},\ i \neq j$

Целевая функция (1) *минимизирует общую стоимость поездки*. Модельные ограничения (2) являются *ограничениями степени* и гарантируют, что *каждого клиента посещает ровно одно транспортное средство*. Ограничения потока (3) и (4) гарантируют, что *каждое транспортное средство может покинуть депо только один раз, а количество транспортных средств, прибывающих к каждому клиенту и въезжающих в депо, равно количеству выезжающих транспортных средств*. В ограничениях (5) указаны *ограничения пропускной способности*, гарантирующие, что *сумма требований клиентов, посещенных по маршруту, меньше или равна вместимости транспортного средства*. Ограничения (6) на исключения подутров гарантируют, что р*ешение не содержит циклов, отключенных от депо*. Остальные обязательные ограничения (7) определяют *области определения переменных*. Число неравенств ограничений на устранение подмаршрутов растет экспоненциально с увеличением числа узлов.

Модификация модели

Рассмотрим формулировку смешанного линейного программирования CVRP с полиномиальным числом ограничений на устранение подмаршрутов. Переменные решения с двумя индексами x_{ij} используются в качестве двоичных переменных, равных 1, если дуга (i, j) принадлежит оптимальному решению, и 0 в противном случае. Для всех пар узлов $i, j \in V - \{0\},\ i \neq j$ мы вычисляем экономию s_{ij} для присоединения к циклам $0 \to i \to 0$ и $0 \to j \to 0$ с использованием дуги (i, j):

$$s_{ij} = c_{i0} + c_{0j} - c_{ij}$$

Экономия s_{ij} показана на рисунке 2. В левой части клиенты i и j обслуживаются собственными транспортными средствами, в правой части клиенты i и j обслуживаются одним транспортным средством.

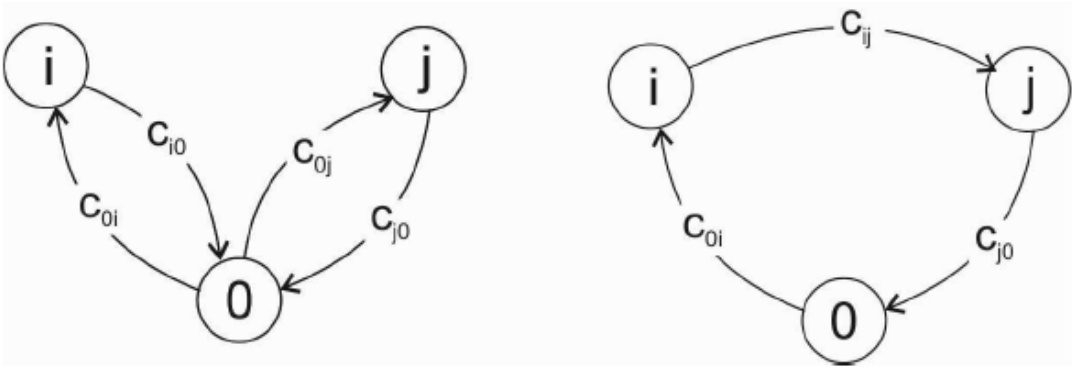

Figure 2: *Saving s_{ij} for customers i and j in cycle $0 \to i \to j \to 0$*

Окончательная стоимость маршрута для допустимого решения может быть записана как:

$$\sum_{i=0}^{n} \sum_{j=0, i \neq j}^{n} c_{ij} x_{ij} = \sum_{i=1}^{n} c_{i0} + \sum_{i=1}^{n} c_{0j} - \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} s_{ij} x_{ij}$$

где выражение $\sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} s_{ij} x_{ij}$ дают общую экономию. Теперь, вместо того, чтобы минимизировать общую стоимость, мы максимизируем общую экономию. Чтобы обеспечить непрерывность маршрута и исключить подмаршруты, мы определяем вспомогательную

непрерывную переменную y_i, $d_i \leq y_i \leq Q$ для $i \in V - \{0\}$, которая показывает (в случае получения товара) загрузку транспортного средства после посещения клиента i. Чтобы упростить моделирование, каждый возможный маршрут $0 \to v_1 \to v_2 \to \ldots \to v_k \to 0$ заменяется путем от узла 0 до узла v_k, т.е. $0 \to v_1 \to v_2 \to \ldots \to v_k$. Новая модель может быть сформулирована следующим образом:

Максимизировать

$$(8) \sum_{i=1}^{n} \sum_{i=1}^{n} s_{ij} x_{ij}$$

При условии

$$(9) \sum_{j=1}^{n} x_{0j} = p,$$

$$(10) \sum_{i=0, i \neq j}^{n} x_{ij} = 1, \ \forall j \in \{1,\ldots,n\},$$

$$(11) \sum_{j=1, i \neq j}^{n} x_{ij} \leq 1, \ \forall i \in \{1,\ldots,n\},$$

$$(12) \ y_i + d_j x_{ij} - Q(1 - x_{ij}) \leq y_j, \ \forall i, j \in \{1,\ldots,n\}, \ i \neq j,$$

$$(13) \ d_i \leq y_i \leq Q, \ \forall i \in \{1,\ldots,n\},$$

$$(14) \ x_{ij} \in \{0,1\}, \forall i, j \in \{0,\ldots,n\}, \ i \neq j.$$

В этой формулировке целевая функция (8) максимизирует общую экономию на поездках. Ограничения (9) налагают, что именно p дуг покидают депо, (10) и (11) являются (10) и (11) являются внутренними и внешними ограничениями для клиентов. Ограничения (12) это ограничения на непрерывность маршрута и устранение подмаршрутов, гарантирующие, что решение не содержит подмаршрутов, отключенных от депо, и что загрузка транспортного средства является функцией без уменьшения шага в соответствии с требованиями клиентов, которые находятся на маршруте транспортного средства. Ограничения, приведенные в (13), являются

ограничениями пропускной способности, которые ограничивают верхнюю и нижнюю границы y_i, и ограничения (14) являются обязательными ограничениями.

Стоит отметить, что трехиндексная формулировка первой модели использует в p раз больше целых переменных, чем двухиндексная формулировка второй модели. Формулировка второй модели имеет полиномиальный размер, т.е. имеет ограничения $O(n^2)$, поэтому обладает более высокой вычислительной эффективностью, чем формулировка первой модели.

Формулировка Смешанного целочисленного программирования для задачи VRPTW

Перед тем как сформулировать задачу введем **обозначения**:

$G(N, A)$ - ориентированный граф, где N - набор вершин, A - набор рёбер
N_C: набор всех клиентов
N: набор всех узлов ($N = \{0\} \cup N_C$)
Q: вместимость транспортного средства
d_i: спрос клиента $i \in N_C$ с $d_i \leq Q$
c_{ij}: стоимость дуги $(i, j) \in A$
t_{ij}: время пути по дуге $(i, j) \in A$
a_{ij}: самое раннее доступное время дуги $(i, j) \in A$
b_{ij}: самое позднее доступное время дуги $(i, j) \in A$

Бинарные переменные решения

$$x_{ij} = \begin{cases} 1, \text{если транспортное средство перемещается непосредственно из узла } i \text{ в узел } j \ (\forall\, i,j \in N) \\ 0, \text{в противном случае} \end{cases}$$

Дополнительные переменные для принятия решения

U_i: загрузка транспортного средства непосредственно перед обслуживанием клиента i ($\forall\, i \in N_C$)
S_i: время отправления транспортного средства из узла i ($\forall\, i \in N_C$)

Модель

(1) $min \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}$

При условии

(2) $\sum_{i \in N} x_{ij} = 1, \ \forall j \in N_C$

(3) $\sum_{j \in N} x_{ji} - \sum_{j \in N} x_{ij} = 0 \ \forall i \in N_C$

(4) $U_j - U_i + Q x_{ij} + (Q - d_i - d_j) x_{ji} \leq Q - d_i, \ \forall i, j \in N_C; \ i \neq j$

(5) $U_i \geq d_i + \sum_{j \in N_0, j \neq i} d_j x_{ij}, \forall i \in N_C$

(6) $U_i \leq Q - (Q - d_i) x_{i0}, \forall i \in N_C$

(7) $S_i - S_j + M1_{ij} x_{ij} \leq M1_{ij} - t_{ij}, \forall i, j \in N_C; \ i \neq j$

(8) $S_j \geq (a_{0j} + t_{0j}) x_{0j}, \forall j \in N_C$

(9) $S_i \geq a_{ij} x_{ij}, \forall i \in N_C; \forall j \in N; \ i \neq j$

(10) $S_i \leq b_{ij} + M2_{ij}(1 - x_{ij}), \forall i \in N_C; \forall j \in N; \ i \neq j$

(11) $x_{ij} \in \{0, 1\}, \ \forall i, j \in N$

(12) $U_j, S_j \geq 0, \ \forall j \in N_C$

где $M1_{ij}$ и $M2_{ij}$ - достаточно большие числа, удовлетворяющие наборам ограничений (7) и (10), когда $x_{ij} = 0$ соответственно. Кроме того, мы устанавливаем $x_{ij} = 0$ всякий раз, когда $(d_i + d_j > Q)$ или $(a_{ij} + t_{ij} > max_{(k \in N | k \neq i)}(b_{jk}))$.

В модели целевая функция (1) ***<u>минимизирует общую стоимость транспортировки</u>***. Наборы ограничений (2) и (3) известны как ***<u>ограничения степени</u>***. В то время как набор ограничений (2) гарантирует, что ***<u>каждый клиент должен быть посещен ровно один раз</u>***, набор ограничений (3) гарантирует, что ***<u>количество входных и выходных дуг из каждого узла одинаково</u>***.

Наборы ограничений (4) - (6) у***страняют подмаршруты с учетом ограничений пропускной способности*** транспортного средства. Наборы ограничений (7) - (10) определяют ***ограничения временных окон***. В то время как установленное ограничение (7) ***определяет время отправления к каждому клиенту в зависимости от предыдущего узла на маршруте***, наборы ограничений (8) и (9) гарантируют, что ***время отправления в каждый узел должно быть больше, чем самое раннее доступное время*** используемой дуги. Аналогично, набор ограничений (10) гарантирует, что в***ремя отправления в каждый узел должно быть меньше, чем самое позднее доступное время*** используемой дуги. И, наконец, наборы ограничений (11) и (12) являются ***ограничениями интегральности и неотрицательности*** соответственно.

Существует способ усиления релаксации Линейного программирования и это улучшение ограничений, которые включают в себя большие M константы (т.е. $M1_{ij}$ и $M2_{ij}$ в наборах ограничений (7) и (10)). Эти числа должны быть как можно меньше, чтобы получить хорошие нижние границы, которые сокращают процесс оптимизации. Поэтому установим $M1_{ij} = \overline{S}_i - \widehat{S}_j + t_{ij}$ и $M2_{ij} = \overline{S}_i - b_{ij}$ где $\overline{S}_i = max_{j \in N}(b_{ij} + t_{ij})$ и $\widehat{S}_j = min(a_{0j} + t_{0j})$.

В математике **релаксация** *(смешанной) целочисленной линейной программы* называется проблема, возникающая при снятии ограничения интегральности каждой переменной.

Например, в задаче двоичного целочисленного программирования все ограничения имеют вид

$$x_i \in \{0, 1\}$$

Релаксация исходной целочисленной программы вместо этого использует набор линейных ограничений

$$0 \leq x_i \leq 1$$

Результирующая релаксация представляет собой линейную

программу, отсюда и название. Этот метод релаксации преобразует NP-трудную оптимизационную задачу (целочисленное программирование) в смежную задачу, разрешимую за полиномиальное время (линейное программирование); решение расслабленной линейной программы может быть использовано для получения информации о решении исходной целочисленной программы.

Существует еще один, который состоит в том, чтобы включить действительные неравенства. Это один из практичных способов усилить релаксацию ЛП в формулировках. Поэтому воспользуемся следующим действительным неравенством для модели.

$$(13)\ \sum_{j \in N_C} x_{0j} \geq r_{VRPTW}(N_C)$$

где, $r_{VRPTW}(N_C) = \text{верхняя граница}(\sum_{i \in N_C} \frac{d_i}{Q})$. Это ограничение ограничивает количество маршрутов, исходящих из депо.