Санкт-Петербургский государственный университет

Кафедра моделирования социально-экономических систем

Ринчинов Роман Цыдыпович

Выпускная квалификационная работа бакалавра

Многокритериальная задача оптимального размещения производства в сети

Направление 010400

Прикладная математика и информатика

Научный руководитель:

ассистент кафедры МСЭС

Парфенов А. П.

Санкт-Петербург

2017

Содержание

Введение

Задача оптимального размещения производства в сети представляет практический интерес в экономике. При планировании развития производства часто возникает необходимость в решении задач оптимального размещения предприятий. Во многих случаях такие задачи являются весьма сложными и требуют применения методов математического моделирования, разработки специальных алгоритмов и программного обеспечения. Исследованию таких задач посвящено множество работ [1]. В качестве критериев может служить множество различных параметров — это усложняет задачу, ведь не всегда понятно какие критерии более важны. Чаще всего рассматриваются задачи размещения пунктов производства одного вида продукции, а в качестве критериев оптимальности, которые следует минимизировать, берутся расстояния от пунктов производства до пунктов потребления. В качестве принципа оптимальности, согласующего эти критерии, ранее рассматривались оптимумы по Парето, сумма расстояний, максимум из расстояний. Оптимизация по принципу минимума суммы сводится к задаче о поиске медиан графа, а оптимизация по принципу минимума максимума — к задаче о поиске центров графа [2]. Также в [2] рассматриваются λ—центидантные постановки задачи, совмещающие задачи о минимизации максимума и суммы критериев.

Подобные задачи являются задачами целочисленного линейного программирования, для которых возможны различные алгоритмы точного и приближенного решения. Алгоритмы поиска медиан и центров, преимущественно основанные на методе ветвей и границ, рассмотрены в [3].

В данной работе ставится многокритериальная задача, которая обобщает задачу из [2]: имеется несколько видов продукции, каждый

из которых необходим в пунктах потребления, необходимо разместить несколько групп заводов, производящих различные виды продукции.

В работе рассмотрены различные методы решения многокритериальной задачи размещения, представляющие различные подходы к задаче(алгоритм, основанный на методе ветвей и границ, также алгоритм, базирующийся на алгоритме для поиска p-центров), выясняются их достоинства, недостатки и ограничения. В рамках данной работы выполнена программная реализация этих методов на языке программирования C++, позволяющая проверить эффективность и провести более детальное сравнение.

1 Постановка задачи

Пусть имеется множество городов, соединенных между собой дорогами. Данную систему можно представить в виде ориентированного графа $G = (X,Y)$, X — множество вершин, Y — множество дуг. Вершинами $x \in X$ будут являться города, а дугами $(y^1,y^2) \in Y$ — дороги между городами. Пронумеруем вершины $X = \{1, \ldots, P\}$.

На дугах задана неотрицательная вещественная функция стоимости транспортировки товаров, обозначим ее $c(y^1,y^2)$. Помимо этого, существует N различных товаров, которые потребляются городами в определенном количестве, т.е. каждой вершине соответствует вектор $x_i = (x_{i1}, \ldots, x_{iN})$, x_{ij} — потребление i-ым городом j-того товара.

Необходимо разместить M пунктов производства товаров. Пусть множество заводов $Z = \bigcup_{i=1}^{N} Z_i$, где Z_i — множество из m_i заводов, производящих i-ю продукцию. Будем рассматривать случай, в котором возможно размещение пунктов производства в городах — вершинах графа. Причем в одном городе не может находиться более одного завода.

Для решения нашей задачи полезно знать c_{ij} — стоимость транспортировки единицы товара из пункта i в пункт j. Функцию стоимости транспортировки продукции будем считать линейной по количеству транспортируемого товара: $c_{ij}x$ — стоимость транспортировки x товара из пункта i в пункт j. Для поиска элементов матрицы c_{ij} можно воспользоваться алгоритмом Флойда. Сложность данного алгоритма принадлежит классу $O(P^3)$, где P — количество городов.

Пусть $A \subset X$ — подмножество вершин графа, в которых размещены пункты производства. Множество A разбивается на непересекающиеся подмножества A_i мощности m_i, $i = 1, \ldots, N$, каждое из которых соответствует заводам, производящим только i-ую продукцию. Размещение заводов $A = A_1 \cup \ldots \cup A_N$, в свою очередь, разбивает множество городов

X на группы N разными способами. Способ, соответствующий i-му виду продукции, выглядит так: $X = X_{j_1}(A_i) \cup \cdots \cup X_{j_{m_i}}(A_i),\ j_1, \ldots, j_{m_i} \in A_i$. Здесь $X_j(A_i)$ — множество городов, для которых завод в вершине j, производящий продукцию i, является ближайшим.

В роли критериев оптимальности может служить стоимость транспортировки продукции от l-го завода, производящего i-ую продукцию, расположенного в вершине l, до пунктов потребления(для заводов необходимо минимизировать эту величину)

$$f_{il}(A_i) = \sum_{k \in X_l(A_i)} c_{jk} x_{ik}, \quad i = 1, \ldots, N. \tag{1.1}$$

Также в роли критерия оптимальности выступает максимальное расстояние между заводом и городом (для городов нужно минимизировать данную величину). Пусть

$$E_{xi} = \min_{l \in A_i} c_{lx},\ i = 1, \ldots, N$$

— минимальное расстояние от ближайшего завода, производящего продукцию i до города x.

В качестве обобщенного критерия оптимальности для города x берем максимум

$$g_x(A) = \max_{i=1}^{N} E_{xi} \quad x \in X \tag{1.2}$$

2 Связь критериев с типовыми задачами на графах

В работе [2] расматривается задача, в которой необходимо разместить несколько пунктов производства, производящих один тип продукции. Если в качестве критериев брать только группу критериев 1.1, а в качестве принципа оптимальности — сумму критериев:

$$\sum_{l \in A_i} f_{il}(A_i),$$

то задача размещения, поставленная в этой работе, сводится к задаче поиска m_i-медиан. Если же брать в качестве критериев группу критериев 1.2, а в качестве принципа оптимальности — утилитарный:

$$\max_{x \in X} E_{xi},$$

то задача сводится к задаче поиска m_i-центров. Такие типовые задачи решаются методом ветвей и границ, либо другими уже известными алгоритмами. Например в [3] приведены алгоритмы поиска центров и медиан.

В данной работе ставится задача, в некотором всмысле обобщающая работу [2]. Мы рассматриваем несколько пунктов производства, производящих различную продукцию, иными словами, размещаем M заводов, производящих различные N видов продукции($M \geqslant N$). Были разработаны алгоритмы, решающие данную задачу. Они основаны на алгоритмах из [3].

Для начала сделаем обзор алгоритмов поиска центров, m_i-центров, медиан и m_i-медиан для поиска оптимального решения при размещении пунктов производства, выпускающих один вид продукции. Эти алгоритмы решают частные случаи задач, поставленных в первой главе. Они помогут нам составить алгоритмы для поиска решений.

2.1 Постановка задач поиска медиан и m_i-медиан

В этом параграфе, в предположении, что у нас есть только один тип продукции, рассматриваются задачи поиска медиан и m_i-медиан, а также их связь с критериями 1.1, поиск оптимального решения по утилитарному принципу.

Пусть товар l производит только один завод. Далее введем величины [3]

$$\begin{aligned} \sigma_{0l}(i) &= \sum_{j \in X} x_{jl} c_{ij} \\ \sigma_{tl}(i) &= \sum_{j \in X} x_{jl} c_{ji} \end{aligned} \tag{2.1}$$

$\sigma_{0l}(i)$ и $\sigma_{tl}(i)$ — называются соответственно *внешним* и *внутренним* передаточными числами вершины i для товара l.

Определение. *Внешняя медиана графа* G — Это такая вершина графа j для которой выполняется

$$\sigma_{0l}(j) = \min_{i \in X} \sigma_{0l}(i)$$

Определение. *Внутренняя медиана графа* G — Это такая вершина графа j для которой выполняется

$$\sigma_{tl}(j) = \min_{i \in X} \sigma_{tl}(i)$$

Нетрудно заметить, в случае если продукцию l производит только один завод, то задача поиска оптимального размещения для этого завода эквивалента задаче поиска внешней медианы графа.

Обобщим понятие медианы. В общем случае у нас количество заводов производящих продукцию l не менее одного. Введем обозначения:

$$\begin{aligned} c(A_i,x_j) &= \min_{l \in A_i} c_{ij} \\ c(x_j,A_i) &= \min_{l \in A_i} c_{ji} \end{aligned} \tag{2.2}$$

— здесь, как прежде, A_i — подмножество индексов вершин, в которых размещены m_i заводов, производящих продукцию i. Передаточные числа для множества вершин A_i определяются так же, как и для одиночной вершины:

$$\begin{aligned} \sigma_{0l}(A_i) &= \sum_{j \in X} x_{jl} c(A_i, x_j) \\ \sigma_{tl}(A_i) &= \sum_{j \in X} x_{jl} c(x_j, A_i) \end{aligned} \quad , \tag{2.3}$$

Определение. *Внешняя m_i-медиана графа G* — Это такое подмножество вершин графа A_{i0} для которой выполняется

$$\sigma_{0l}(A_{i0}) = \min_{A_i \subset X} \sigma_{0l}(A_i)$$

Определение. *Внутренняя m_i-медиана графа G* — Это такое подмножество вершин графа A_{i0} для которой выполняется

$$\sigma_{tl}(A_{i0}) = \min_{A_i \subset X} \sigma_{tl}(A_i)$$

Из определения внешней медианы и из первой главы видно, что случае если продукцию i производит m_i заводов, то задача поиска оптимального размещения этих m_i заводов по критериям 1.1, при использовании утилитарного принципа оптимальности, эквивалентна задаче поиска внешней m_i-медианы графа.

2.2 Способы решения задач поиска медиан и m_i-медиан

Несложно заметить, что алгоритм поиска медианы в графе тривиален (при условии что известна c_{ij} — матрица кратчайших расстояний между вершинами сложность алгоритма Флойда порядка $O(P^3)$). Пусть товар l производит только завод. Достаточно посчитать внешние передаточные для каждой вершины(сложность данной операции $O(P^2)$), вершины для которых передаточные числа минимальны — будут являться

медианами графа. Таким образом поиск медиан в графе имеет сложность порядка $O(P^3)$.

Пусть товар l производит $m_l > 1$ заводов. В таком случае для поиска m_l-медиан, нам нужно проверить все подмножества X мощности m_l. Количество таких подмножеств — это количество сочетаний из P по m_i. А точнее: $r = C_P^{m_i} = \frac{P!}{m_i!(P-m_i)!}$. Перебор можно осуществлять рекурсивно. Для каждого сочетания считать передаточное число(сложность порядка $O(P^2)$). Таким образом, сложность алгоритма поиска медианы имеет порядок $O(C_P^{m_i} * P^2)$. Очевидно, при достаточно большом P и $m_i > 1$, r — быстро возрастает. Например $C_{100}^2 = 4950$, $C_{100}^3 = 161700$, $C_{100}^{10} = 17310309456440$. Таким образом перебор не эффективен для больших P.

Формулировка задачи в терминах целочисленного программирования [3]

Пусть $[\xi_{ij}]$ — матрица распределения, в которой

$$\begin{cases} 1, & \text{если город } j \text{ прикреплен к заводу } i \\ 0, & \text{в противном случае.} \end{cases}$$

Далее примем $\xi_{ii} = 1$, если вершина i является медианной вершиной, иначе $\xi_{ii} = 0$. Тогда задача о p–медиане может быть сформулирована следующим образом:

Минимизировать функцию

$$z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \xi_{ij} \tag{2.4}$$

при ограничениях

$$\begin{cases} \sum_{i=1}^{n} \xi_{ij} = 1 & \text{для } j = 1,...,n, \\ \sum_{i=1}^{n} \xi_{ii} = p, & \\ \xi_{ij} \leqslant \xi_{ii} & \text{для всех } i,j = 1,...,n, \\ \xi_{ij} \in \{0,1\}, & \end{cases} \tag{2.5}$$

где d_{ij} — матрица взвешенных расстояний графа (она получается из матрицы расстояний после умножения j-того столбца на вес x_{jl} — потребление l–того товара j–ым городом $d_{ij} = c_{ij}x_{jl}$). Если $\overline{\xi}$ является оптимальным решением этой задачи, то p-медиана имеет вид:

$$\overline{X_p} = \{x_i | \overline{\xi_{ii}} = 1\}$$

Таким образом, для исходной задачи получена задача, сформулированная в терминах целочисленного программирования. Такие задачи являются NP-полными, задачу целочисленного линейного программирования можно решить методом ветвей и границ [4].

Также задачу можно решить изменив ограничения на переменную $\xi_{ij} \in \{0,1\}$ на $xi_{ij} \geqslant 0$ — получится задача линейного программирования [4]. Решение задачи ЛП будет не обязательно целочисленными. Ревель и Свэйн [5] показали, что при решение задачи ЛП дробные решения получаются редко. Поэтому в большинстве случаев можно решать задачу ЛП. Если попадаются дробные значения в векторе решения, например $\widetilde{\xi_{ij}}$ — не целое. Тогда нужно решить на две новые задачи ЛП с дополнительными условиями: $\widetilde{\xi_{ij}} = 0$ и $\widetilde{\xi_{ij}} = 1$. Продолжать данную операцию до того момента пока решение задачи ЛП $\widetilde{\xi}$ — станет целочисленным.

Также в [3] описан алгоритм основанный на методе ветвей и границ. Там же описан метод оценки нижней границы. Такой подход также подходит для поиска медиана в графе.

2.3 Постановка задач поиска центров и m_i-центров

В этом параграфе, в предположении, что у нас есть только один тип продукции, рассматриваются задачи поиска центров и m_i-центров, а также их связь с критериями 1.2 и поиск оптимального решения по эгалитарному принципу.

Пусть товар l производит только один завод. Далее введем величины:

$$\begin{aligned} s_{0l}(i) &= \max_{j \in X} x_{jl} c_{ij} \\ s_{tl}(i) &= \max_{j \in X} x_{jl} c_{ji} \end{aligned} \tag{2.6}$$

$s_{0l}(x_i)$ и $s_{tl}(x_i)$ — называются соответственно *внешним* и *внутренним* числами разделения вершины x_i для товара l.

Определение. *Внешний центр графа* G — Это такая вершина графа j для которой выполняется

$$s_{0l}(j) = \min_{i \in X} s_{0l}(i)$$

Определение. *Внутренний центр графа* G — Это такая вершина графа $\bar{x_0}$ для которой выполняется

$$s_{tl}(\bar{x_0}) = \min_{x_i \in X} s_{tl}(x_i)$$

Нетрудно заметить, в случае если продукцию l производит только один завод, то задача поиск оптимального размещения для этого завода эквивалента задаче поиска внутреннего центра графа.

В общем случае в нашей задаче количество заводов, производящих продукцию i, может быль больше одного. Как прежде, A_i — множество вершин заводов, m_i — количество заводов производящих продукцию i. Понятие центра допускает обобщение: можно рассматривать не отдельную точку(вершину графа), а множество из m_i-вершин. Как прежде определим $s_{0l}(A_i)$ и $s_t(A_i)$ — числа разделения для множества вершин A_i:

$$s_{0l}(A_i) = \max_{x_j \in X} x_{jl} c(A_i, x_j)$$
$$s_{tl}(A_i) = \max_{x_j \in X} x_{jl} c(x_j, A_i) \tag{2.7}$$

— где $c(A_i,x_j)$ и $c(A_i,x_j)$ определены из 2.2

Определение. *Внешний m_i-центр графа G* — Это такое подмножество вершин графа A_{i0} для которой выполняется

$$s_{0l}(A_{i0}) = \min_{A_i \subset X} s_{0l}(A_i)$$

Определение. *Внутренний m_i-центр графа G* — Это такое подмножество вершин графа A_{i0} для которой выполняется

$$s_{tl}(A_{i0}) = \min_{A_i \subset X} s_{tl}(A_i)$$

В случае если продукцию i производит m_i заводов, то задача поиска оптимального размещения для этой группы заводов эквивалента задаче поиска внешнего m_i-центра графа [2].

2.4 Решение задач поиска центров и m_i-центров

Задачу поиска m_i-центра для графа $G = (X,Y)$ можно решать следующим образом: [3].

Рассмотрим по очереди каждую вершину i и «углубимся» по всем возможным маршрутам, выходящим из нее, на расстояние $\delta_i = \lambda / x_{il}$ заданная константа, которую мы будем называть константой «проникновения».

Пусть

$$R_{\lambda,l}(i) = \{j | x_{jl} c_{ji} \leqslant \lambda, j \in 1, \ldots, P\} - \tag{2.8}$$

множество вершин j графа G, из которых вершина i может быть достигнута с использованием взвешенной длины $x_{jl} c_{ji} \leqslant \alpha$.

Области, из которых не достижимы никакие вершины (в пределах заданного расстояния δ_i), описываются соотношением

$$\Phi_{\lambda,l}(0) = \{j|j \in 1,\ldots,P\} - \bigcup_{R_{\lambda,l}(i)} \tag{2.9}$$

где второй член исключает все области графа G, из которых можно достигнуть хотя бы одну вершину i.

Области, из которых можно достигнуть (в пределах заданного расстояния δ_i) ровно t вершин $i_1,i_2,\ldots,i_t$ (для любого$t = 1,\ldots,P$) определяются следующим выражением:

$$\Phi_{\lambda,l}(i_1,i_2,\ldots,i_t) = \bigcap_{q=1,\ldots,t} R_{\lambda,l}(i_q) - \{[\bigcap_{q=1,\ldots,t} R_{\lambda,l}(i_q)] \bigcap [\bigcup_{q=t+1,\ldots,P} R_{\lambda,l}(i_q)]\} \tag{2.10}$$

где второй член исключает такие области, из которых достижимы вершины $i_1,i_2,\ldots,i_t$ и еще хотя бы одна из оставшихся вершин графа.

Более того, расположим все $P(P-1)$ расстояний в матрице расстояний в строку в порядке неубывания $[f_1,f_2,\ldots,f_{P(P-1)}]$.

Описание алгоритма:

а) Выбрать $t = 0$.

б) Установить $t = t + 1$. Выбрать $\lambda = f_t$.

в) Построить множества $R_{\lambda,l}(i)$ для всех $i \in 1,\ldots,P$ и найти области $\Phi_{\lambda,l}$.

г) Образовать двудольный граф $G' = (X' \bigcup X, Y')$, где X' — множество вершин, каждая из которых соответствует некоторой области Φ_λ, и Y' множество дуг, такое, что дуга между областью-вершиной и вершиной i существует тогда и только тогда, когда i может быть достигнута из этой области.

д) Найти наименьшее доминирующее множество графа G' (см. гл. 3) [3].

е) Если число вершин графа содержащихся в приведенном выше множестве больше, чем m_lто вернуться к шагу 2; в противном случае остановиться. Области этого множества образуют абсолютный m_l-центр исходного графа G.

Таким образом можно найти размещение множества m_i заводов, производящих продукцию i.

3 Описание алгоритмов, решающих задачу

Для поиска оптимального решения поставленной задачи, в качестве принципа оптимальности для группы критериев 1.1 был взят утилитарный принцип:

$$\sum_{i=1}^{N} \sum_{l \in A_i} f_{il}(A_i).$$

Так для минимизирования суммы критериев 1.1, был разработан алгоритм, основанный на методе ветвей и границ. А для поиска оптимального решения для группы критериев 1.2 был выбран эгалитарный принцип оптимальности:

$$\max_{x \in X} g_x(A)$$

.

3.1 Метод ветвей и границ для поиска оптимального решения по утилитарному принципу

Стоит заметить, что при переборе методом ветвей и границ, неважно какая группа критериев рассматривается 1.1 или 1.2, различия будут заключаться лишь в подсчете нижней границы. Поэтому будет подробно описан случай для утилитарного принципа оптимальности для критериев 1.1.

Определение. Назовем *заводом* вершину графа, в которой расположен пункт производства.

Построим N матриц $Q^i,\ i = 1, \ldots, N$, с элементами q^i_{kj}. Матрица Q^i размера $P \times P$ описывает размещение заводов, производящих i-ую продукцию. Ее строки соответствуют вершинам, в которых гипотетически могут располагаться заводы, а j-ый столбец содержит вершины графа $q^i_{1j},\ q^i_{2j}, \ldots,$ расположенные в неубывающем порядке по расстоянию до вершины j. При переборе в методе ветвей и границ сперва перебираем все

столбцы первой матрицы, потом второй, и т. д. В i-ой матрице вершина j последовательно прикрепляется к вершинам q^i_{1j}, $q^i_{2j}, \ldots$. Прикрепление j к q^i_{kj} означает, что в вершине q^i_{kj} расположен завод, производящий i-ую продукцию, ближайший к городу j (при этом город q^i_{kj} автоматически прикрепляется к заводу q^i_{kj}). Если к вершине x прикреплена хоть одна вершина, при дальнейшем переборе пропускаем x в i-ой и последующих матрицах.

Замечания для перебора

Чтобы уменьшить множество перебираемых вершин в каждой матрице Q^i, сделаем несколько замечаний из [3].

а) Из матрицы Q^i можно удалить m_i последних строк.

б) Пусть вершина j' прикреплена к заводу $q^i_{k'j'}$. Пусть вершина j' будет k-ой ближайшей вершиной к некоторой еще не прикрепленной вершине j $(j' < j \leqslant P)$, соответствующей столбцу матрицы. Тогда, очевидно, все элементы q^i_{lj} с $l > k$ могут быть при движении вверх в дереве вариантов исключены из дальнейшего рассмотрения (временно отмечены, а при движении по дереву вниз отметки будут сняты).

в) Пусть вершина j' прикреплена к заводу $q^i_{k'j'}$. Тогда можно предположить, что все вершины $q^i_{1j'}$, $q^i_{2j'}, \ldots, q^i_{(k'-1)j'}$ не являются заводами.

г) Если из рассмотрения исключены t верхних элементов j-го столбца, соответствующего нераспределенной вершине j, а $(t+1)$-й элемент $q^i_{(t+1)j'}$ является заводом, то j должна быть прикреплена к вершине $q^i_{(t+1)j'}$.

д) Если на некотором этапе проводимого поиска будет построено множество из m_i заводов, то каждую оставшуюся нераспределенную вершину можно прикрепить к ближайшему заводу.

Вычисление нижней границы

Пусть на определенном шаге метода вычислены прикрепления для матриц $Q^1, \ldots, Q^{i-1}$. Пусть в матрице Q^i выполненные прикрепления дали m' заводов и j' городов прикреплено. Значит осталось прикрепить еще $h = m_i - m'$ заводов. Пусть J — множество индексов еще нераспределенных вершин.

Пусть l_{a_j} и l_{b_j} — первые два неотмеченных элемента в столбце j матрицы Q^i (не исключенные из рассмотрения). Пусть число различных l_{a_j} для $j \in J$ равно h'. Наилучшим прикреплением вершины j будет l_{a_j}.

Если $h = h'$, то, прикрепив вершины j к l_{a_j}, получим m_i заводов, а значит, матрица Q^i заполнена. Если же $h > h'$, для получения нижней границе нужно заменить $h - h'$ минимальных стоимостей на вторые по величине. Для поиска дополнительной стоимости нужно найти на $h - h'$ наименьших значений $d_{ij}(c_{l_{b_j}j} - c_{l_{a_j}j})$ по всем вершинам из J. Нижняя граница будет получаться из суммирования уже выполненных распределений и $h - h'$ наименьших значений. Если же $h < h'$, то наилучшее пополнение частного решения даст меньше заводов, чем m_i. Из [3] известно, что $f_i(A)$ монотонно убывает с увеличением m_i, значит текущее частное решения не является частью оптимального.

Нижняя граница считается в предположении, что элементы матриц $Q^1, \ldots, Q^{i-1}$ уже прикреплены, а значит, стоимость для них суммируется с нижней границей для Q^i-й матрицы.

3.2 Алгоритм поиска решения по эгалитарному принципу для критериев 1.2

При эгалитарном подходе нужно минимизировать данную величину:

$$\max_{x \in X} g_x(A) = \max_{x \in X} \max_{i=1}^{N} E_{xi} = \max_{i=1}^{N} \max_{x \in X} E_{xi}.$$

Если нет ограничений на количество размещений заводов в одном городе, то

$$\min_{A} \max_{i=1}^{N} \max_{x \in X} E_{xi} = \max_{i=1}^{N} \min_{A_i} \max_{x \in X} E_{xi}$$

таким образом необходимо найти решение N задач о m_i–центрах. Однако в нашем случае есть ограничение, и не может в одном городе находиться более одного завода. Поэтому нужно искать другие подходы для решения задачи. Будем искать размещения поочередно для каждого вида продукции, исключая уже занятые вершины графа. Для поиска решения будем пользоваться алгоритмом из предыдущей главы для поиска m_i-центров.

4 Применение алгоритмов на практике

В рамках данной выпускной квалификационной работы осуществлена программная реализация описанных в предыдущем разделе методов на языке C++. Как язык программирования высокого уровня, C++ поддерживает необходимый уровень абстракции для удобной платформонезависимой разработки алгоритмов рассматриваемых методов, при этом предоставляя большое количество стандартных структур данных, дополнительных подключаемых библиотек алгебры матриц и линейного программирования, а также обеспечивая высокое время исполнения скомпилированной программы.

4.1 Программная реализация

Этот параграф посвящен описанию структуры программного кода, реализованных алгоритмов. С частичным листингом кода можно ознакомиться в Приложении.

В основе реализации лежит класс `OptimalAlloc`, предназначенный для создания графа, описывающего систему городов и дорог, а также методов поиска оптимального размещения пунктов производств. Непосредственно методы оптимизации реализованы public-методами `solveByBranchAndBoundMethod()` и `solveByMCenters()`, при этом каждый метод задействует ряд вспомогательных private-методов. Для хранения промежуточных структур(матриц, списков, множеств) используется стандартная библиотека шаблонов в языке программирования C++(Standard Template Library, STL) — это удобные шаблоны, которые поддерживают динамическое выделение памяти, что позволяет более просто и эффективно реализовать данный алгоритм.

Метод `solveByBranchAndBoundMethod()` реализует алгоритм, приведенный в параграфе 3.1. Для отыскания матрицы расстояний меж-

ду произвольными двумя вершинами реализован алгоритм Флойда в качестве private-метода `floyd()`. Также были реализованы private-методы `makeQ()` и `AllocQ()` — создание матриц Q их перебор. Для поиска нижней границы был реализован еще один private-метод `countLB()`.

Метод `solveByMCenters()` задействует один private-метод `AllocCenter()`, который является реализацией алгоритма из 2.4. Для работы эффективной работы `AllocCenter()` необходимо искать наименьшее покрытие множества, поэтому создан дополнительный класс `Covering`. В котором для эффективного поиска наименьшего покрытия используется шаблон `std::bitset` — структура, позволяющая хранить битовые множества, а также производить над ними простейшие побитовые операции.

4.2 Анализ эффективности работы алгоритмов

Благодаря программной реализации мы можем провести более детальный анализ методов оптимизации, применив их к конкретным задачам. Для этого сгенерируем наборы задач размещения производств в сети. В качестве параметров будут выступать такие величины как количество городов P, количество видов продукции N, а также количество пунктов потребления M.

Результаты вычислений для примера

Пусть $P = 25$, $N = 3$, $G(X,Y)$ — граф, моделирующий систему городов, изображенную на рисунке 4.1. Пусть есть три вида продукции A,B,C, каждый из которых должен производиться на двух, трех и четырех заводах соответственно. Пусть также в таблице 4.2 представлены, количества потребления городами продукции A,B,C.

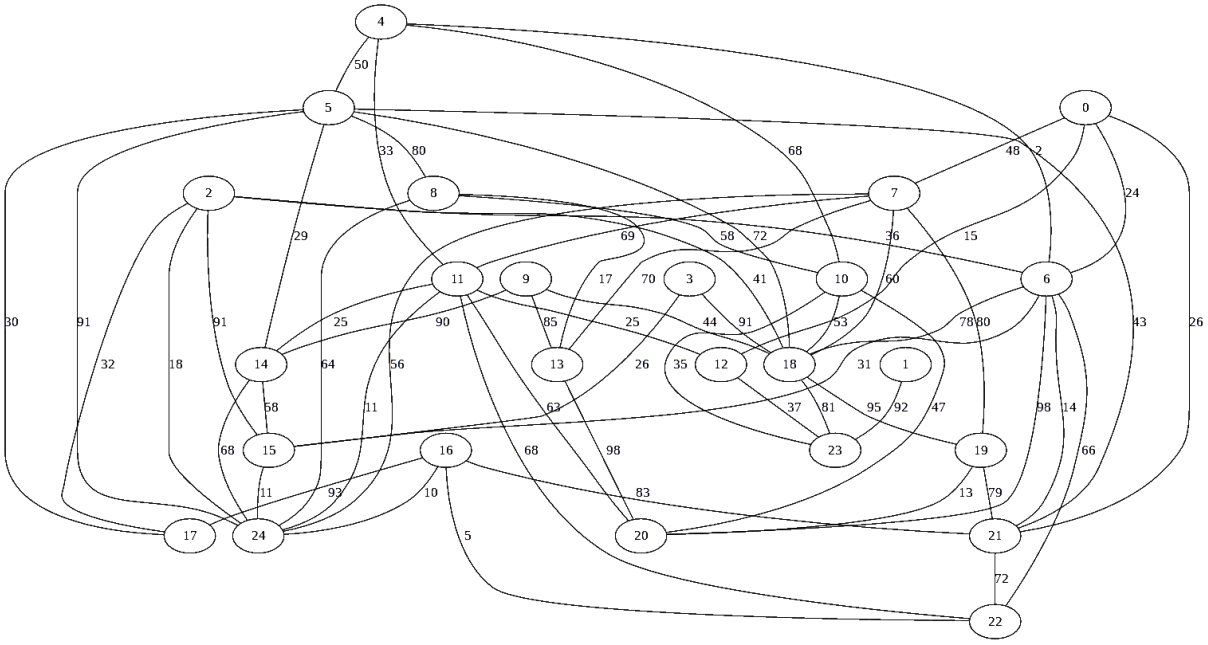

Рисунок 4.1 — $G(X,Y)$

	0	1	2	3	4	5	6	7	8	9	10	11	12
A	6,7342	8,6837	2,6005	2,7608	6,9523	8,8675	4,6362	8,7417	0,1018	3,0381	9,2814	1,402	4,9267
B	8,3781	4,4063	1,2347	2,4525	3,3948	3,5534	4,11	1,6908	1,5886	5,1249	2,1154	1,5014	3,3524
C	2,9349	7,4702	3,2032	1,1425	2,9258	5,3476	4,9461	8,385	2,8409	3,729	5,5729	9,1599	4,4943
	13	14	15	16	17	18	19	20	21	22	23	24	
A	9,0704	9,4354	8,8372	3,6533	5,3358	2,1748	9,118	7,9117	7,7874	4,97	0,7442	2,4627	
B	3,4768	8,9858	4,7417	4,0068	8,4474	2,1574	5,8304	1,9784	8,5181	5,9865	6,7702	8,6146	
C	9,3991	6,4012	2,4691	3,9377	5,0886	1,0072	6,0321	8,5151	2,9342	0,6789	1,2979	9,0464	

Рисунок 4.2 — Таблица потребления товаров A,B,C всеми городами

На рисунках изображены 4.3 и 4.4 — размещения, полученные соответственно алгоритмом, основанным на методе ветвей и границ, а также в методе построенном на алгоритме поиска m_i-центров. Заводы производящие продукцию A отмечены красным цветом, B — зеленом, C — синим.

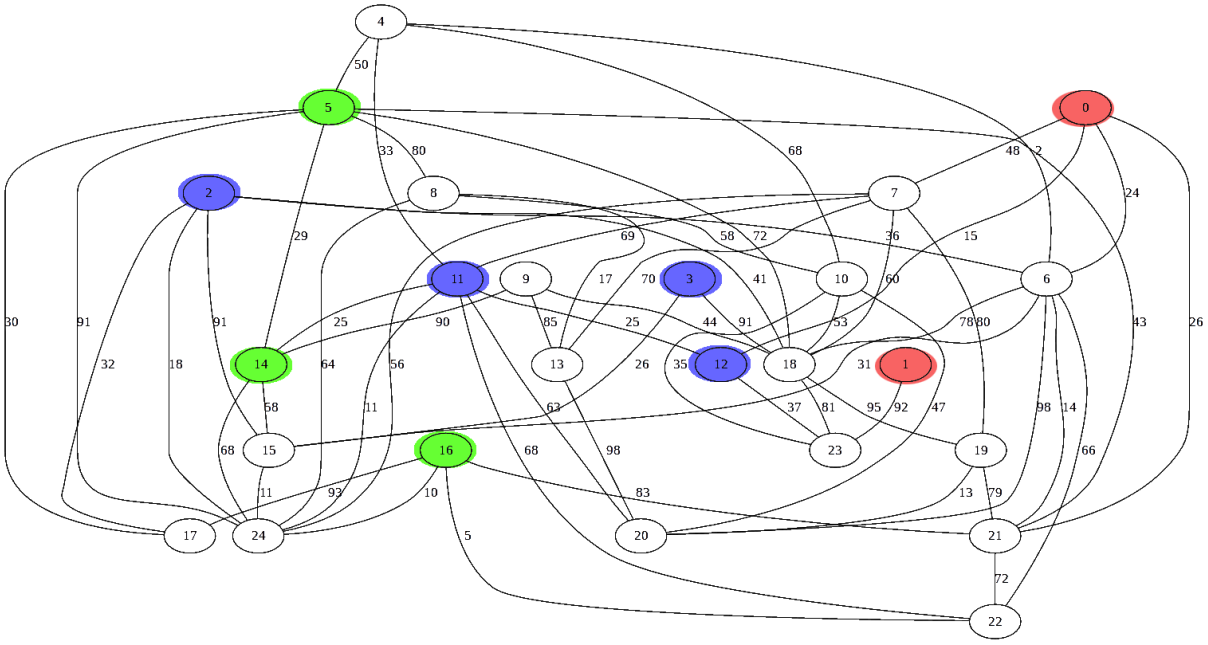

Рисунок 4.3 — Время исполнения программного кода: 3.5 сек

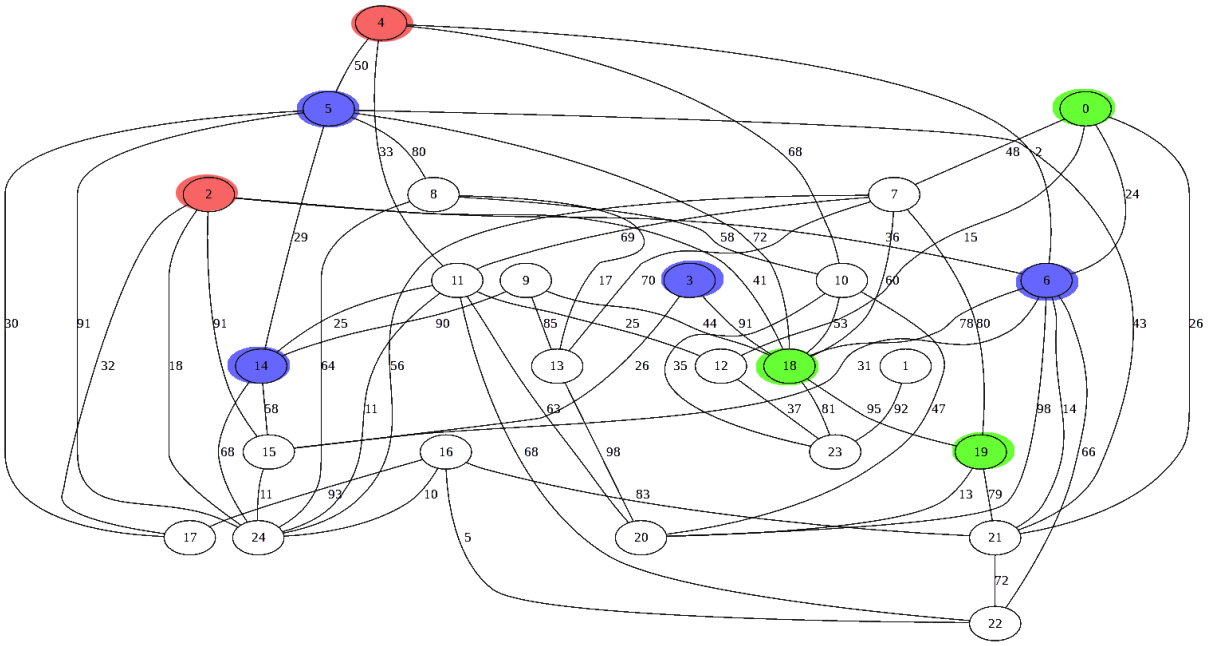

Рисунок 4.4 — Время исполнения программного кода: 3.7 сек

Оценка зависимости времени исполнения реализованных алгоритмов от параметров.

Для тестов программы было написано скриптов на языке Python 3: программа для генерации векторов потребления продукции городами, а также взвешенных связных графов с заданным количеством вершин.

Первая программа использует python-библиотеку [6] для генерации связного графа. Второй скрипт, запускает основную программу со сгенерированными данными, засекает время исполнения, в случае если время исполнения выше заданного параметра, он прерывает исполнение программы, также сохраняет вывод программы в файл. Для тестов было выбрано ограничение на время исполнения программы —60 секунд. Также программы ограничивались по потреблению оперативной памяти — 4000 Mb, превышении данного порога, программы принудительно прекращали свое исполнение.

На рисунках 4.5 и 4.6 изображены диаграммы. На данных диаграммах по вертикальной оси — время исполнения в секундах, по горизонтальной — количество вершин в графе, ломаные — запуск для разного количества типов производимой продукции.

На рисунке 4.5 показаны результаты запуска алгоритма поиска оптимального размещения по критериям 1.1, в качестве принципа берётся утилитарный принцип оптимальности. Алгоритм основан на методе ветвей и границ.

Данный алгоритм имеет высокую скорость исполнения для количества производимых различных товаров $N = \{1,3\}$, при количестве вершин в графе $P < 65$, однако, при $N = 5$ удалось вычислить только для $P = 55$ вершин, при $N = 7$ — $P = 45$ вершин, а при $N = 9$ — только $P = 35$ вершин.

На рисунке 4.5 показаны результаты запуска алгоритма поиска оптимального размещения по критериям 1.2, в качестве принципа берётся эгалитарный принцип оптимальности. Метод основан на алгоритме поиска центров в графе.

Данный алгоритм имеет высокую скорость исполнения для количества производимых различных товаров $N = \{1,3,5,7,9\}$, при количестве вершин в графе $P < 25$, однако, при $N \geqslant 3$ время исполнения растет

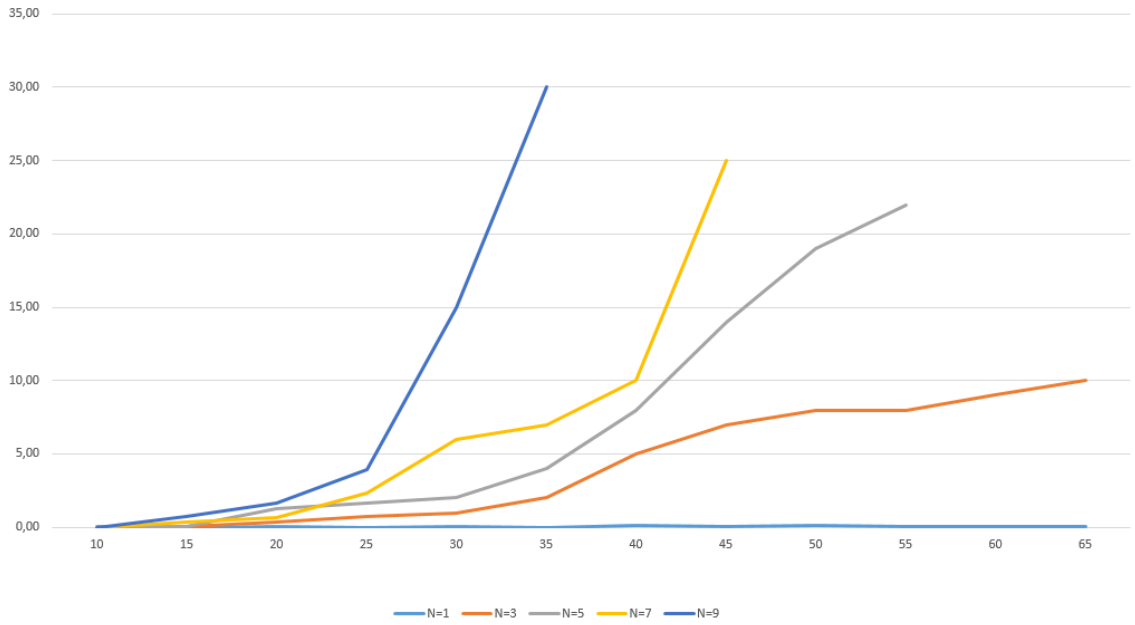

Рисунок 4.5

быстрее чем у прошлого алгоритма. Так ни для одного $N = \{3,5,7,9\}$ за отведенное время не найдено размещение при $P > 45$. Для $N = \{3,5,7,9\}$ найдены размещения только для соответственно $P \leqslant \{45,40,35,30\}$ — вершин в графе.

Итак, оба алгоритма работают быстро для $P \leqslant 25$ вершин в графе. Далее время алгоритмов довольно быстро возрастает. Во втором алгоритме в качестве одной из подзадач используется поиск наименьшего дискриминируемого множества, сложность этой задачи быстро возрастает при росте количества вершин. Поэтому время исполнения программы возрастает очень быстро.

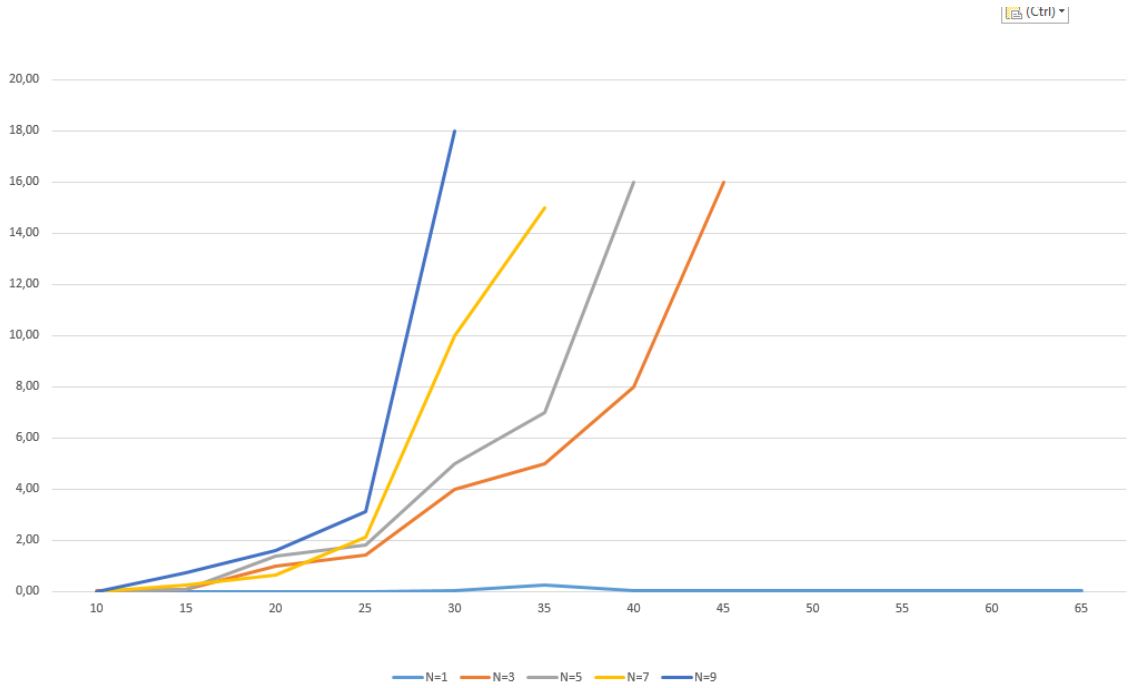
(Ctrl)
20,00
18,00
16,00
14,00
12,00
10,00
8,00
6,00
4,00
2,00
0,00
10
15
20
25
30
35
40
45
50
55
60
65
N=1
N=3
N=5
N=7
N=9

Рисунок 4.6

1

Заключение

В работе рассмотрена задача многокритериальной оптимизации размещения пунктов производства. Данная задача рассматривалась в литературе [1], так например в [2] автор рассматривает задачу размещения производств одного вида продукции в сети. Применяя простейшие принципы оптимальности: эгалитарный и утилитарный, автор сводит задачи размещения к задачам о поиске медиан и центров в графе. В той работе, обобщается данная задача, на размещение пунктов производства, производящих несколько видов продукции. Были выбраны методы для решения, а также разработана программная реализация алгоритмов, решающих задачу. Для утилитарного принципа оптимальности хорошо подходит алгоритм, основанный на методе ветвей и границ, также в ряде случаев задачу можно свести к задаче ЦЛП. Что касается эгалитарного принципа, то для него был предложен алгоритм, основанный на поиске m_i-центров.

Для обоих методов написана программная реализация на языке C++ и проведено тестирование на наборах сгенерированных задач с различными параметрами. Был проведен анализ их работы на различных входных данных, сделаны выводы о скорости и эффективности работы алгоритма. Так оба алгоритма работают быстро при количестве вершин $P \leqslant 25$ и количестве различных видов продукции $N\{1,3,5,7,9\}$. Однако при увеличении P, скорость алгоритмов существенно снижается.

Исследование методов не только показало их применимость к решению задачи, но также и наглядно продемонстрировало различия подходов, определяющих эти методы. Однако подходы к поиску оптимального размещения разнообразны, и даже в рамках различных методов оптимальности методы решения могут быть различны. Поэтому исследование задачи оптимального размещения может быть продолжено. Можно

брать различные принципы оптимальности, искать Парето-оптимальные решения [7] поставленной задачи, оптимизировать использование памяти и скорость выполнения алгоритмов, реализованных в рамках данной работы. Также можно рассмотреть возможность размещения производств на дугах графа.

Список использованных источников

1. Farahani R. Z, SteadieSeifib M., Asgari N. Multiple criteria facility location problems: A survey // Applied Mathematical Modelling, Vol. 34, Iss. 7. 2010. P. 1689–1709.

2. Ogryczak W. Location Problems from the Multiple Criteria Perspective: Effcient Solutions. // Archives of Control Sciences, 7 (XLIII). 1998. P. 161–180.

3. *Н. Кристофидес* Теория графов. Алгоритмический подход. Москва. 1978 год.

4. *А. Схрейвер* Теория линейного и целочисленного программирования. Москва. 1991 год.

5. *Revelle C. S.,Swain R. W.* Central facilites location, Geographical Analysis, 2, стр.30 1970 год

6. Brian Wesley Baugh Generate a randomly connected graph with N nodes and E edges.: [Электронный ресурс]: Github. URL: https://gist.github.com/bwbaugh/4602818 (дата обращения: 9.05.2017).

7. *В.Д. Ногин* Принятие решений в многокритериальной среде. Москва. 2005 год.

Приложение А Программный код реализации алгоритмов

```
class OptimalAlloc{
public:
  OptimalAlloc(const char* file);
  OptimalAlloc(const char* graph,const char* prod);

  vector<set <int> > BranchAndBound();
  vector<set <int> > Center();

private:
  vector<vector<float> > X;
  vector<int> M;
  int P;
  vector<vector<int> > t;
  vector<vector<int> > c;
  void print(vector<vector<int> > vec);
  void print(list<list<int> > l);
  vector<vector<int> >  read(const char *filename,
    int *N, vector<vector<float> > *X,vector<int> *M
    );
  vector<vector<int> >  floyd(int N, vector<vector<
    int> > vec);
  list<list<int> >  makeQ(int N, vector<vector<int> >
     vec);
  float count_ng(list<list<int> > Q,int p,set<int>A,
    vector<float> x,vector<vector<int> > c);
```

```
    set<int> pereborQ(list<list<int> > Q,int p, vector<
        float> x,vector<vector<int> > c, set<int>A_old);
    set <int>  centerAlg(int p,set<int> A_old,list<
        float> L);
    void read(const char *prod, vector<vector<float> >
        *X,vector<int> *M);
    vector<vector<int> >  read(const char *graph,int *N
        );
};


list<list<int> >  OptimalAlloc::makeQ(int N, vector<
    vector<int> > vec){
  list<list<int> > result;

  for ( int i=0;i<N;i++ ) {
    list<int> l;

    l.push_back(i);

    for ( int j=0;j<N;j++){
      if (i!=j){
        bool flag = true;
        for (list<int>::iterator it = l.begin(); it !=
            l.end(); it++){
          if (vec[*it][i]>=vec[j][i]){
            flag=false;
            l.insert(it,j);
            break;
```

```
            }
          }

          if (flag)
            l.push_back(j);
        }

      }
      l.push_front(1);
      result.push_back(l);
    }
    return result;
}


float OptimalAlloc::count_ng(list<list<int> > Q,int p,
    set<int>A, vector<float> x,vector<vector<int> > c){
    float result=0;

    int h=0; // количество не распределенных вершин
    list<vector<int> > m;
    int co=0;
    for (list<list<int> >::iterator lis = Q.begin(); lis
       != Q.end(); lis++){

      if (*((*lis).begin())!=0){
        vector<int> v;
        v.push_back(co);
```

```
        for (list<int>::iterator it = (*lis).begin();it
          != (*lis).end(); it++){
          if ( it == (*lis).begin())
            it++;
          if (A.find(*it)==A.end()){
            v.push_back(*it);
            if (v.size()==3)
              break;
          }
        }
        h++;
        m.push_back(v);
      }
      else{
        for (list<int>::iterator it = (*lis).begin(); it
          != (*lis).end(); it++){
          if ( it == (*lis).begin())
            it++;
          if (*it >99) {
            result += x[co]*c[co][*it-100];
            break;
          }
        }
      }
      co++;

  }
```

```
if (p-A.size()<=h){//прикрепляем к самым верхним элем
    ентам не прикрепленных столбцов
   for (list<vector<int> >::iterator it = m.begin();
       it != m.end(); it++)
       result += x[(*it)[0]]*c[(*it)[0]][(*it)[1]];

   if (p-A.size()<h){//найти h-p+A.size() наименьших р
       азностей 6.20
       list<float> mmm;
       for (list<vector<int> >::iterator it = m.begin();
           it != m.end(); it++)
         mmm.push_back(x[(*it)[0]]*(c[(*it)[0]][(*it)
             [2]]-c[(*it)[0]][(*it)[1]]));
      mmm.sort();
      mmm.reverse();
       co=0;
       for (list<float>::iterator it = mmm.begin();it !=
           mmm.end() and co!=h-p+A.size();it++){
         co++;
         result+=*it;
       }

   }
}
else
   result=1E101;
```

```
  return result;

}

set<int> OptimalAlloc::pereborQ(list<list<int> > Q,int
  p, vector<float> x,vector<vector<int> > c, set<int>
  A_old){//перебирает все вершины которые можно добави
  ть
  list<list<int> >::iterator list_;
  list<int>::iterator it_;
  float ng_=1e100;
  int p_=0;
  set<int>A;
  while(p_<p){
    bool flag = false;

      for (list<list<int> >::iterator lis = Q.begin();
        lis != Q.end(); lis++){

      if (*((*lis).begin())!=0)

        for (list<int>::iterator it = (*lis).begin();
          it != (*lis).end(); it++){
          if ( it == (*lis).begin())
            it++;
          if (A.find(*it)==A.end() and A_old.find(*it)
            ==A_old.end() ){
```

```
            A.insert(*it);
            *it=*it+100;
            *((*lis).begin())=0;

            float ng=count_ng(Q,p,A,x,c);
            *it=*it-100;
            *((*lis).begin())=1;
            A.erase(*it);
            if (ng<ng_){
               flag=true;
               ng_=ng;
               list_=lis;
               it_=it;
            }
          }

        }
      if (flag){
        *((*list_).begin())=0;
        *it_=*it_+100;
        p_++;
        A.insert(*it_-100);
        flag= false;
      }
    }

}
return A;
```

```
}

vector<set <int> > OptimalAlloc::BrachAndBound(){
  vector<set <int> > result;
  list<list<int> > Q= makeQ(P,c);
  set<int> A_old;
  for (int i=0; i<X.size() ; i++){
    set< int> A_ = pereborQ(Q,M[i],X[i],c,A_old);
    A_old.insert(A_.begin(),A_.end());
    result.push_back(A_);
  }
  return result;
}

class Covering
  {
  public:
    std::vector<bitset <100> > bitset_;

    list<pair<bitset<100>, set<int> > > list_;
    int p;
    Covering(std::vector<vector<int> > c,int P,float l)
      {
      p=P;
      for (int i=0;i<c.size();i++){
        pair<bitset<100>, set<int> > temp;
        for (int j=0;j<c.size();j++){
          if (c[i][j] <=l){
```

```
            temp.first[j]=1;
          }
          if (j!=i)
            temp.second.insert(j);
        }

        list_.push_back(temp);
        bitset_.push_back(temp.first);
        }


}
set <int> find_covering(int step,set<int> A_old){
  set <int> result;
  list<pair<bitset<100>, set<int> > > temp;
  for ( pair<bitset<100>  , set<int> >  l : list_){
    for (int k : l.second){

      if (A_old.find(k)==A_old.end()){
        if (p==1 and bitset_[k].count()==bitset_.
            size()){
          set<int> t;
          t.insert(k);
          return t;
        }
        bitset<100> t = l.first|bitset_[k];
        if (t.count()==bitset_.size() and p!=1){
```

```
                return makeset(l.second,k,bitset_.size())
                    ;
              }
              pair<bitset<100>, set<int> > r(l);
              r.second.erase(k);
              r.first=t;
              temp.push_back(r);
            }
          }
        }

        if (step+1 == p or p==1)
          return result;
        else{
          list_ = temp;
          return find_covering(step+1,A_old);
        }
    }
private:
    set <int> makeset(set<int> t, int p,int P){
      set <int> result;
      for (int i=0;i<P;i++)
        if (t.find(i)==t.end())
          result.insert(i);
      result.insert(p);
      return result;
```

```
  }

};

vector<set <int> >  OptimalAlloc::Center(){
  vector<set <int> > result;
  set<int> A_old;
  list<float> L;
  for (int i=0; i<M.size() ; i++){
    for (int j=0; j<M.size() ; j++){
      if (c[i][j]!=0)
        L.push_back(c[i][j]);
  }
  L.sort();
  }
  for (int i=0; i<M.size() ; i++){
    set< int> A_ = centerAlg(M[i],A_old,L);
    A_old.insert(A_.begin(),A_.end());
    result.push_back(A_);
  }
  return result;
}


set <int> OptimalAlloc::centerAlg(int p,set<int> A_old,
   list<float> L){
  set <int> result;
  list<float> ::iterator t =L.begin();
```

```
    do{
        if (t==L.end()){
            result.clear();
            break;
        }

        Covering cov(c,p,*t);
        result = cov.find_covering(1,A_old);
        t++;

    }while(result.size()<p);
    return result;
}
```

Приложение Б Программный код реализации тестов

Скрипт, генерирующий входные данные.

```
import subprocess
import random
import sys
def rand_distanse_adj_list(st,p,e):
  s="p"+str(p)+"e"+str(e)
  res = open(s,"w")
  res.write(str(p)+"␣"+str(e)+"\n")
  list_ = list(st[1:-2].split(",\n"))
  t=False
  for l in list_:
    if not t:
      l_=list(l[1:-1].split(","))
      t = True
    else:
      l_=list(l[2:-1].split(","))
    res.write(str(l_[0])+"␣")
    res.write(str(l_[1])+"␣")

    res.write(str(random.random())+"␣")
    res.write(str(random.random())+"\n")
  #print list_

def rand_potr(N,p):
  s="N"+str(N)+"p"+str(p)
  res = open(s,"w")
```

```
    res.write(str(p)+" "+str(N)+"\n")
    for m in range(1,N+1):
      random.seed()
      #print p/(2*N)+1
      res.write(str(random.randint(1,(p/(2*N))+1))+" ")
      for t in range(p):
        res.write(str(random.random()*10)+ " ")
      res.write("\n")


def main(argv):
    for p in range(10,10+int(argv),5):
      e = p*(p-1)/10
      pr = subprocess.Popen("python
          random_connected_graph.py -p "+str(p)+ " -e " +
          str(e), shell=True, stdout=subprocess.PIPE)
      rand_distanse_adj_list(pr.stdout.read(),p,e)
      for N in range(1,10,2):
        rand_potr(N,p)
    return 0


if __name__ == '__main__':
    print "hi"
    main(sys.argv[1])
```

Скрипт, запускающий тесты.

```
import subprocess
import sys
import os
```

```
import time
import threading, json

class Command(object):
  def __init__(self, cmd):
    self.cmd = cmd
    self.process = None
    self.tim =None
    self.stdout = None
  def tostr(self,obj):
    return "answer"
  def run(self, timeout):
    def target():
      print 'Thread started'
      start = time.time()

      self.process = subprocess.Popen(self.cmd, shell=
        True, stdout=subprocess.PIPE)
      self.stdout = self.process.stdout.read()

      self.process.communicate()
      if 0 == self.process.returncode:
        self.tim=time.time() - start
      else:
        self.tim= None
      print 'Thread finished'
```

```
        thread = threading.Thread(target=target)
        thread.start()


        thread.join(timeout)

        if thread.is_alive():
            print 'Terminating process'
            self.process.terminate()
            thread.join()
            return self.tostr(self.stdout), None
        return self.tostr(self.stdout), self.tim
        print self.process.returncode


def main(argv1,argv2,argv3):
    P = {}
    for p in range(10,10+int(argv1),5):
        e = p*(p-1)/10
        adj_list="p"+str(p)+"e"+str(e)
        N_={}
        for N in range(1,10,2):
            XM="N"+str(N)+"p"+str(p)
            print "test graph\\" +adj_list+" graph\\"+XM
            command = Command("./test graph/" +adj_list+"
                graph/"+XM +" " +argv3)
            stdout_,tim  = command.run(timeout=int (argv2))
```

```
            if tim == None:
                break;
            #pr = subprocess.Popen("./test graph/" +adj_list
                +" graph/"+XM , shell=True, stdout=subprocess.
                PIPE)
            #tim=time.time() - start
            N_[N]=[stdout_,tim]
        P[p]=N_

    print P
    file = open("result.json","w")
    file.write(json.dumps(P, separators=(',', ':')))
    return P

if __name__ == '__main__':
    stime = time.time()
    results = main(sys.argv[1],sys.argv[2],sys.argv[3])
    print time.time() - stime
```