Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Программирование на языках высшего уровня

Тема: Метод ветвей и границ

Выполнил
студент гр. 3331506/70401 Соколов Д.А.
Преподаватель Ананьевский М. С.

«____» _____ 2020 г.

Санкт-Петербург
2020

Введение

Задачи дискретной оптимизации имеют конечное множество допустимых решений, которые теоретически можно перебрать и выбрать наилучшее (дающее минимум или максимум целевой функции). Практически же зачастую это бывает неосуществимо даже для задач небольшой размерности.

В методах неявного перебора делается попытка так организовать перебор, используя свойства рассматриваемой задачи, чтобы отбросить часть допустимых решений. Наибольшее распространение среди схем неявного перебора получил метод ветвей и границ, в основе которого лежит идея последовательного разбиения множества допустимых решений. На каждом шаге метода элементы разбиения (подмножества) подвергаются анализу – содержит ли данное подмножество оптимальное решение или нет. Если рассматривается задача на минимум, то проверка осуществляется путем сравнения нижней оценки значения целевой функции на данном подмножестве с верхней оценкой функционала. В качестве оценки сверху используется значение целевой функции на некотором допустимом решении. Допустимое решение, дающее наименьшую верхнюю оценку, называют рекордом. Если оценка снизу целевой функции на данном подмножестве не меньше оценки сверху, то рассматриваемое подмножество не содержит решения лучше рекорда и может быть отброшено. Если значение целевой функции на очередном решении меньше рекордного, то происходит смена рекорда. Будем говорить, что подмножество решений просмотрено, если установлено, что оно не содержит решения лучше рекорда.

Если просмотрены все элементы разбиения, алгоритм завершает работу, а текущий рекорд является оптимальным решением. В противном случае среди непросмотренных элементов разбиения выбирается множество, являющееся в определенном смысле перспективным. Оно подвергается

разбиению (ветвлению). Новые подмножества анализируются по описанной выше схеме. Процесс продолжается до тех пор, пока не будут просмотрены все элементы разбиения.

Формальное описание

Пусть рассматривается задача вида

$$f(x) \rightarrow \min_{x \in D} \quad ,$$

где $f(x)$ – вещественная функция, а D – конечное множество допустимых решений. Пусть $d \subseteq D$. Функцию $b(d)$, ставящую в соответствие множеству d разбиение его на подмножества $d_1, ..., d_N$, $N > 1$, будем называть *ветвлением*.

Вещественная функция $H(d)$ называется *нижней границей* для d, если

1. $H(d) \leq \min_{x \in d} f(x)$;
2. на одноэлементном множестве $\{x\}$ верно равенство $H(\{x\}) = f(x)$.

Алгоритм, реализующий метод ветвей и границ, состоит из последовательности однотипных шагов. На каждом шаге известен рекорд x^0 и подмножества $t_1, t_2, ..., t_{L0}$ не просмотренных решений. В начале работы алгоритма $L = 1$, $t_1 = D$, x^0 – произвольный элемент множества D или пустое множество (на пустом множестве положим значение функционала равным бесконечности).

На каждом шаге алгоритм начинает работу с проверки элементов разбиения. Пусть проверяется множество t_j. Множество t_j отсекается в одном из двух, последовательно проверяемых случаев:

a. если $H(t_j) \geq f(x^0)$;
b. если $H(t_j) < f(x^0)$ и найден такой элемент $y_j \in t_j$, что $f(y_j) = \min_{x \in t_j} f(x) = H(t_j)$.

В случае *b* происходит смена рекорда $x^0 = y_j$.

Пусть $t_1, t_2, \ldots, t_M$ $(M \leq L)$ – неотсеченные множества (будем считать, что отсечены множества с номерами $M + 1, \ldots, L$).

В случае $M = 0$ алгоритм заканчивает работу, и в качестве решения задачи принимается рекорд x^0. При $M \geq 1$ среди множеств $t_1, \ldots, t_M$ выбирается множество для нового ветвления. Пусть таковым является множество t_1. Тогда осуществляется ветвление $b(t_1) = (d_1, \ldots, d_N)$, в результате которого получаем список множеств $d_1, \ldots, d_N, t_2, \ldots, t_M$. Эти множества нумеруются числами от 1 до L, и начинается новый шаг алгоритма.

Нетрудно убедиться в том, что описанный алгоритм находит оптимальное решение за конечное число шагов.

Описанная последовательность действий является общей схемой метода ветвей и границ для решения задач на минимум. При решении конкретной задачи следует указать способы построения нижней и верхней оценок, метод ветвления, а также правило выбора перспективного множества для разбиения.

При определении «перспективного» элемента разбиения в основном применяются две схемы: *одновременного (многостороннего)* и *одностороннего* ветвления. При одновременном ветвлении функция b может быть применена к любому элементу разбиения. Часто в качестве такого элемента выбирается подмножество t_kс минимальной нижней границей

$$H(t_k) = \min_{1 \leq i \leq L} H(t_i).$$

При одностороннем ветвлении номер разбиваемого подмножества известен заранее. В этом случае, не ограничивая общности, можно считать, что «перспективным» является подмножество t_1. Отметим, что при односторонней схеме ветвления нет необходимости запоминать все элементы разбиения, достаточно иметь информацию о первом элементе разбиения и объединении остальных элементов.

Разбиения множеств решений (ветвление) удобно представлять в виде дерева решений. На рис. 1 приведены примеры одновременной (*а*) и

односторонней (*б*) схем ветвления. Каждая вершина дерева соответствует некоторому подмножеству решений. Дуги, исходящие из вершины, означают, что на некотором шаге это подмножество отсечь не удалось и оно было разбито на подмножества. Вершины, в которые входят эти дуги, соответствуют подмножествам, полученным в результате ветвления. Зачеркнутые висячие вершины означают отсеченные подмножества. Незачеркнутые висячие вершины соответствуют непросмотренным множествам, среди которых на следующем шаге алгоритма выбирается подмножество для дальнейшего ветвления.

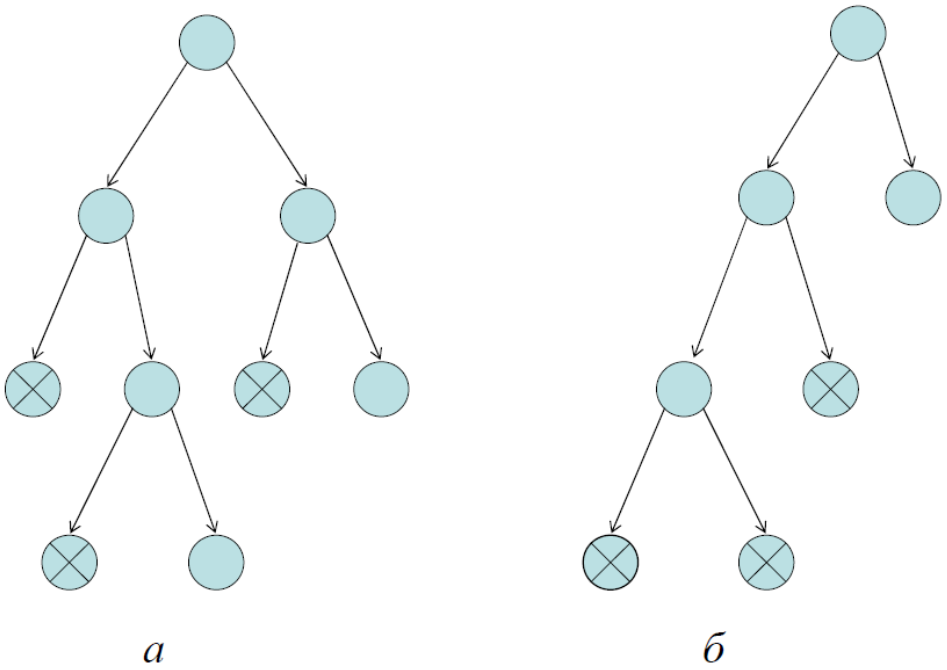

Рисунок 1 — Дерево решений

В схеме одностороннего ветвления выбирается первая (левая) вершина на нижнем уровне, а для схемы одновременного ветвления такой вершиной может быть любая. Алгоритм заканчивает работу, если зачеркнуты все висячие вершины дерева ветвлений.

Графическое представление метода ветвей и границ иллюстрирует его суть – отсечение ветвей дерева поиска, которое осуществляется на основании

сравнения нижней границы и значения функционала на рекорде. Это объясняет название метода.

При использовании описанной выше схемы для решения конкретной задачи необходимо ее уточнение, учитывающее особенности рассматриваемой задачи. В разделе «Практическая реализация» приведен пример реализации метода ветвей и границ для решения задачи коммивояжера.

Псевдокод

Алгоритм состоит из следующих операций:

1. Найти решение `xh` задачи оптимизации. Сохранить его значение, `B = f (xh)`. (Если эвристика недоступна, установить `B` в бесконечность.) `B` будет обозначать лучшее решение, найденное до сих пор, и будет использоваться в качестве верхней границы для возможных решений.
2. Инициализировать очередь для хранения частичного решения задания.
3. Цикл до тех пор, пока очередь не опустеет
    1. Взять узел из очереди
    2. Если `N` представляет собой единственное возможное решение `x` и `f (x) < B`, то `x` является лучшим решением на данный момент. Записать его и установить `B ← f(x).`
    3. В противном случае, ветвь N имеет N_i узлов. Для каждого из них:
        1. Если граница `(Ni) > B`, ничего не делать; так как нижняя граница на этом узле больше верхней границы задачи, она никогда не приведет к оптимальному решению и может быть отброшена.
        2. В противном случае — сохранить `Ni` в очереди.

В листинге 1 представлен псевдокод алгоритма.

```
CombinatorialSolution branch_and_bound_solve(
      CombinatorialProblem problem,
      ObjectiveFunction objective_function /*f*/,
      BoundingFunction lower_bound_function /*g*/)
{
      // Шаг 1
      double problem_upper_bound = std::numeric_limits<double>::infinity;//B
      CombinatorialSolution heuristic_solution = heuristic_solve(problem);//x_h
      problem_upper_bound = objective_function(heuristic_solution);// B = f(x_h)
      CombinatorialSolution current_optimum = heuristic_solution;
      // Шаг 2
      queue<CandidateSolutionTree> candidate_queue;
      // инициализация очереди
      candidate_queue = populate_candidates(problem);
      while (!candidate_queue.empty()) { // Шаг 3
            // Шаг 3.1
            CandidateSolutionTree node = candidate_queue.pop();
            if (node.represents_single_candidate()) { // Шаг 3.2
                  if (objective_function(node.candidate())<problem_upper_bound){
                  current_optimum = node.candidate();
                  problem_upper_bound = objective_function(current_optimum);
                  }
                  // иначе, узел единственный и не является оптимальным
           }
           else { // Шаг 3.3
                  for (auto&& child_branch : node.candidate_nodes) {
                        if(lower_bound_function(child_branch)<=
                        problem_upper_bound) {
                              candidate_queue.enqueue(child_branch);// Шаг 3.3.2
                        }
                        // Шаг 3.3.1
                  }
          }
      }
      return current_optimum;
}
```

Листинг 1 — Псевдокод алгоритма ветвей и границ

Практическая реализация

Формулировка задачи:

Задан полный ориентированный граф G = (V, E) с множеством вер-шин V = {1, …, n} и множеством дуг E. Каждой дуге (i,j)∈ E приписана длина cij ≥ 0. В общем случае задача коммивояжера формулируется на произвольном графе, поэтому длины некоторых (в частности, не сущест-вующих) ребер могут быть сколь угодно большими. Так, считаем, что cii = + ∞.

Требуется найти гамильтонов контур минимальной длины.

Теоретическое описание решения:

Имеется множетво S всех гамильтоновых циклов рафа. На каждом шаге в S ищется ребро (i, j), исключение которого из маршрута максимально увеличит оценку снизу. Далее происходит разбиение множества на два непересекающихся S1 и S2. S1 — все циклы, содержащие ребро (i, j) и не содержащие (j, i). S2 — все циклы, не содержащие (i, j). Далее вычисляется оценка снизу для длины пути каждого множества и, если она превышает длину уже найденного решения, множество отбрасывается. Если нет — множества S1 и S2 обрабатываются так же, как и S.

Алгоритмическое описание

Имеется матрица расстояний M. Диагональ заполняется бесконечными значениями, т.к. не должно возникать преждевременных циклов. Также имеется переменная, хранящая нижнюю границу.

Стоит оговориться, что нужно вести учет двух видов бесконечностей — одна добавляется после удаления строки и столбца из матрицы, чтобы не возникало преждевременных циклов, другая — при отбрасывании ребер. Случаи будут рассмотрены чуть позже. Первую бесконечность обозначим как inf1, вторую — inf2. Диагональ заполнена inf1.

1. Из каждого элемента каждой строки вычитается минимальный элемент данной строки. При этом минимальный элемент строки прибавляется к нижней границе
2. Из каждого столбца аналогично вычитается минимальный элемент и прибавляется к нижней границе.
3. Для каждого нулевого элемента M(i, j) вычисляется коэффициент, равный сумме минимальных элементов строки i и столбца j, исключая сам элемент (i, j). Этот коэффициент показывает, насколько гарантированно увеличится нижняя граница решения, если исключить из него ребро (i, j)

4. Ищется элемент с максимальным коэффициентом. Если их несколько, можно выбрать любой (все равно оставшиеся будут рассмотрены на следующих шагах рекурсии)
5. Рассматриваются 2 матрицы — M1 и M2. M1 равна M с удаленными строкой i и столбцом j. В ней находится столбец k и строка l, в которых не содержится inf1 и элемент M(k, l) приравнивается inf1. Как было сказано ранее, это необходимо во избежание преждевременных циклов (т.е. на первых этапах (k, l) == (j, i)). Матрица M1 соответствует множеству, сожержащему ребро (i, j). Вместе с удалением столбца и строки ребро (i, j) включается в путь.
6. M2 равна матрице M, у которой элемент (i, j) равен inf2. Матрица соответствует множетсву путей, не сожержащих ребро (i, j) (важно понимать, что ребро (j, i) при этом не исключается).
7. Переход к п.1 для матриц M1 и M2.

Эвристика состоит в том, что у матрицы M1 нижняя граница не больше, чем у матрицы M2 и в первую очередь рассматривается ветвь, содержащая ребро (i, j).

Реализация:

Получение нулей в каждой строке и каждом столбце.

```
double LittleSolver::subtractFromMatrix(MatrixD &m) const {
    // сумма всех вычтенных значений
    double subtractSum = 0;
    // массивы с минимальными элементами строк и столбцов
    vector<double> minRow(m.size(), DBL_MAX),
        minColumn(m.size(), DBL_MAX);
    // обход всей матрицы
    for (size_t i = 0; i < m.size(); ++i) {
        for (size_t j = 0; j < m.size(); ++j)
            // поиск минимального элемента в строке
            if (m(i, j) < minRow[i])
                minRow[i] = m(i, j);

        for (size_t j = 0; j < m.size(); ++j) {
            // вычитание минимальных элементов из всех
            // элементов строки, кроме бесконечностей
```

```
            if (m(i, j) < _infinity) {
                m(i, j) -= minRow[i];
            }
            // поиск минимального элемента в столбце после
вычитания строк
            if ((m(i, j) < minColumn[j]))
                minColumn[j] = m(i, j);
        }
    }

    // вычитание минимальных элементов из всех
    // элементов столбца, кроме бесконечностей
    for (size_t j = 0; j < m.size(); ++j)
        for (size_t i = 0; i < m.size(); ++i)
            if (m(i, j) < _infinity) {
                m(i, j) -= minColumn[j];
            }

    // суммирование вычтенных значений
    for (auto i : minRow)
        subtractSum += i;

    for (auto i : minColumn)
        subtractSum += i;

    return subtractSum;
}
```

Увеличение нижней границы и сравнение ее с рекордом.

```
// вычитание минимальных элементов строк и столбцов
// увеличение нижней границы
bottomLimit += subtractFromMatrix(matrix);
// сравнение верхней и нижней границ
if (bottomLimit > _record) {
    return;
}
```

Расчет коэффициентов.

```
double LittleSolver::getCoefficient(const MatrixD &m, size_t r,
size_t c) {
    double rmin, cmin;
    rmin = cmin = DBL_MAX;
    // обход строки и столбца
    for (size_t i = 0; i < m.size(); ++i) {
        if (i != r)
            rmin = std::min(rmin, m(i, c));
        if (i != c)
            cmin = std::min(cmin, m(r, i));
    }
```

```
    return rmin + cmin;
}
```

Поиск всех нулевых элементов и вычисление их коэффициентов.

```
// список координат нулевых элементов
list<pair<size_t, size_t>> zeros;
// список их коэффициентов
list<double> coeffList;

// максимальный коэффициент
double maxCoeff = 0;
// поиск нулевых элементов
for (size_t i = 0; i < matrix.size(); ++i)
    for (size_t j = 0; j < matrix.size(); ++j)
        // если равен нулю
        if (!matrix(i, j)) {
            // добавление в список координат
            zeros.emplace_back(i, j);
            // расчет коэффициена и добавление в список
            coeffList.push_back(getCoefficient(matrix, i, j));
            // сравнение с максимальным
            maxCoeff = std::max(maxCoeff, coeffList.back());
        }
```

Отбрасывание нулевых элементов с немаксимальными коэффициентами.

```
{ // область видимости итераторов
    auto zIter = zeros.begin();
    auto cIter = coeffList.begin();
    while (zIter != zeros.end()) {
        if (*cIter != maxCoeff) {
            // если коэффициент не максимальный, удаление элемента
из списка
            zIter = zeros.erase(zIter);
            cIter = coeffList.erase(cIter);
        }
        else {
            ++zIter;
            ++cIter;
        }
    }
}

return zeros;
```

Переход к множеству, содержащему ребро с максимальным штрафом.

```
auto edge = zeros.front();
// копия матрицы
auto newMatrix(matrix);
```

```
// из матрицы удаляются строка и столбец, соответствующие вершинам
ребра
newMatrix.removeRowColumn(edge.first, edge.second);
// ребро iter добавляется к пути
auto newPath(path);
newPath.emplace_back(matrix.rowIndex(edge.first),
                     matrix.columnIndex(edge.second));
// добавление бесконечности для избежания преждевремнного цикла
addInfinity(newMatrix);
// обработка множества, содержащего ребро edge
handleMatrix(newMatrix, newPath, bottomLimit);
```

Переход к множеству, не содержащему ребро с максимальным штрафом.

```
// переход к множеству, не сожержащему ребро edge
// снова копирование матрицы текущего шага
newMatrix = matrix;
// добавление бесконечности на место iter
newMatrix(edge.first, edge.second) = _infinity + 1;
// обработка множества, не сожержащего ребро edge
handleMatrix(newMatrix, path, bottomLimit);
```

Полную реализацию и весь код можно найти по ссылке:

Ссылка

Несмотря на то, что метод ветвей и границ в худшем случае ничем не лучше полного перебора, в большинстве случаев он значительно выигрывает во времени благодаря эвристике для поиска начального решения и отбрасыванию заведомо плохих множеств.

На рисунке 3 представлены графики сравнения МВиГ с полным перебором и среднее время работы реализованного мной алгоритма на различном количестве городов. Тестировалось на матрицах для случайно сгенерированных точек.

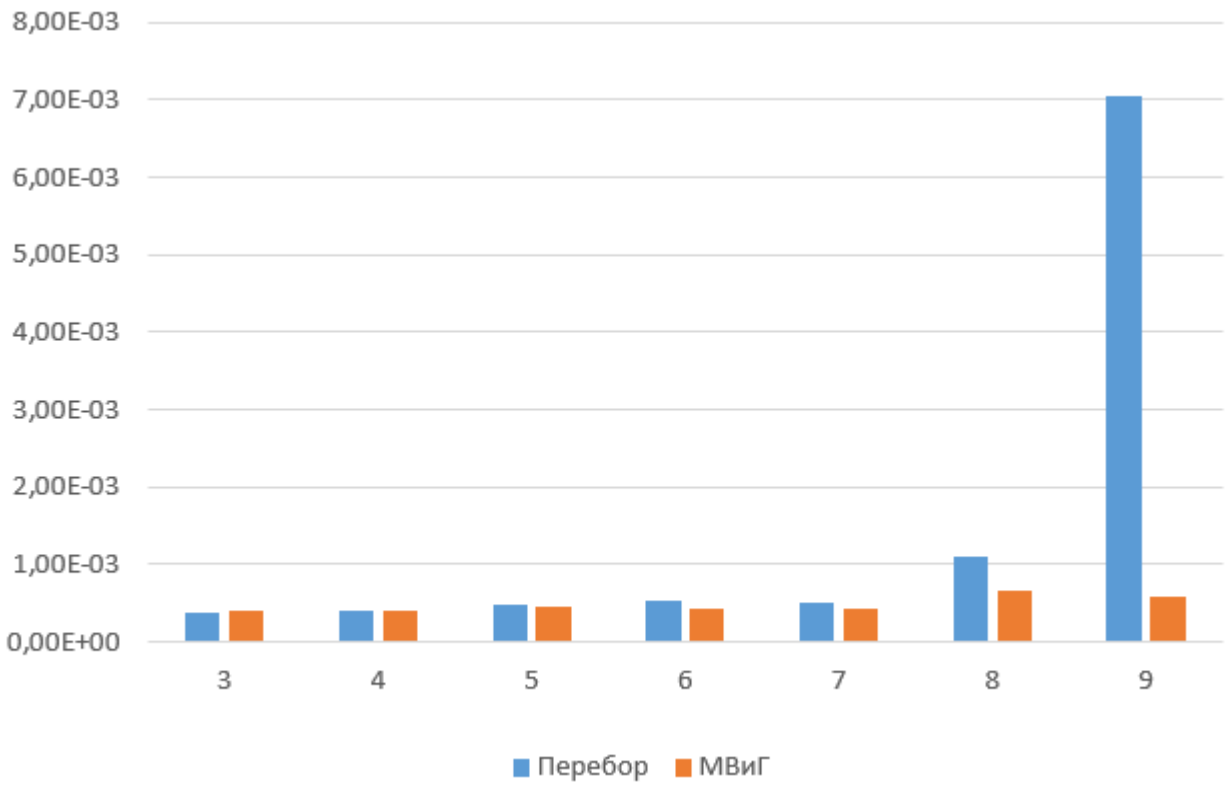

Рисунок 3 — Сравнение метода ветвей и границ с полным перебором

Начиная с 9 городов полный перебор заметно проигрывает МВиГ. Начиная с 13 городов полный перебор занимает больше минуты.

Список литературы

1. Е. Н. Гончаров, А. И. Ерзин, В. В. Залюбовский. Исследование операций. Примеры и задачи. Учебное пособие. Новосибирск. : Новосибирский государственный университет, 2005. 78 с.
2. Решение задачи коммивояжера алгоритмом Литтла с визуализацией на плоскости. Илья @scidev. [habr.com]