Отчет по программе

Гончаров Сергей, ИУ8-54, 31 вариант задания

Оглавление

Задание

Вы устроились на работу специалистом по ИБ. Руководство поставило задачу обеспечения ИБ в предприятии. Известно, что при повышении безопасности страдает удобство пользователей. У вас есть набор средств и мер ИБ, каждая из которых защищает от одной или нескольких угроз. При этом каждое средство снижает удобство пользователя. Постройте алгоритм выбора средств, которые с одной стороны защищают от всех известных угроз, а с другой — наименее неудобны для пользователя.

Теоретическая часть

Данная задача очевидным образом к задаче булева программирования.

Задача булева программирования

Постановка задачи. Найти среди n-мерных булевых векторов $x = (x_1, x_2, ..., x_n)$, ($x_i \in \{0,1\}$ для $i = 1, ..., n$), удовлетворяющих системе

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \le b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \le b_2 \\ ... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \le b_m \end{cases}$$

такой, для которого достигается минимум ЦФ

$$\min F = c_1x_1 + c_2x_2 + ... + c_nx_n.$$

Задача булевого программирования может быть сформулирована, как «задача о рюкзаке», а также как «задача о покрытии».

Данная задача является NP-трудной, значит решить ее за полиномиальное время не представляется возможной.

Сведение нашей задачи к БП

Сведение осуществляется следующим образом:

- коэффициент a_{ij} – защищает ли j-ый метод от i-ой угрозы
- переменная x_i – используем ли мы i-ую защиту
- $\sum a_{ni} * x_i \ge 1$ – гарантирует, что хотя бы один метод защищает от n-ой угрозы
- коэффициенты c_i – ухудшение пользовательских условий i-ой защитой
- цель задачи следующая – минимизировать ухудшение условий при полной покрытии угроз, т.е. F→min

Методы решения

Распространенные методы решения задачи БП следующие:

- полный перебор;
- «жадные» алгоритмы;
- улучшенный перебор (метод Фора–Мальгранжа, метод Балаша) (аддитивные методы);
- эвристические методы (генетические алгоритмы, нейронные сети и др.).

В данном случае будем решать задачу методом улучшенного перебора: ветвей и границ с использованием аддитивных улучшений метода Балаша. Почему именно так? Полный перебор очень долго работает, жадные алгоритмы дают только приближенное значение (не всегда оптимальное), нейронные сети программировать не умеем (. Метод улучшенного перебора хоть в худших случаях дает сложность $O(2^n)$ (полный перебор), обычно работает быстрее и позволяет найти оптимальное решение достаточно быстро.

Метод ветвей и границ

Метод ветвей и границ (англ. branch and bound) — общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации. По существу, метод является вариацией полного перебора с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Метод ветвей и границ впервые предложили в 1960 году Ленд и Дойг для решения задач целочисленного программирования.

Общая идея метода может быть описана на примере поиска минимума функции f(x) на множестве допустимых значений переменной x. Функция f и переменная x могут быть произвольной природы. Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ).

Процедура ветвления состоит в разбиении множества допустимых значений переменной x на подобласти (подмножества) меньших размеров. Процедуру можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево, называемое деревом поиска или деревом ветвей и границ. Узлами этого дерева являются построенные подобласти (подмножества множества значений переменной x).

Процедура нахождения оценок заключается в поиске верхних и нижних границ для решения задачи на подобласти допустимых значений переменной x.

В основе метода ветвей и границ лежит следующая идея: если нижняя граница значений функции на подобласти A дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B, то A может быть исключена из дальнейшего рассмотрения (правило отсева). Обычно минимальную из полученных верхних оценок записывают в глобальную переменную m; любой узел дерева поиска, нижняя граница которого больше значения m, может быть исключён из дальнейшего рассмотрения.

Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции и достигается на соответствующей подобласти.

Аддитивные улучшения Балаша

Предположим, что выполнены неравенства $0 \le c1 \le c2 \le \dots \le cn$.(если нет, необходимо отсортировать переменные, чтобы неравенство выполнялось)

Допустимое решение x доминирует допустимое решение y, если $F(x) < F(y)$. (Для недопустимого решения x положим $F(x) = +\infty$.) Если решения доминируются лучшим найденным допустимым решением (рекордом), то их можно отбросить.

Алгоритм начинает работу с вершины x = (0, … ,0). Затем просматривает следующие за ней вершины. При этом перебор вершин может быть сокращен на основе различных правил, некоторые из которых приведены ниже.

- Правило 1. Так как имеют место неравенства $0 \le c1 \le c2 \le \dots \le cn$, то значение целевой функции при переходе к следующему решению может только возрасти. Значит, если некоторая вершина соответствует допустимому решению, то не надо просматривать следующие за ней вершины. Они исключаются.
- Правило 2. Пусть F^* − минимальное (рекордное) значение целевой функции на найденных допустимых решениях, Fq – значение функционала в вершине Q, где j – номер вершины по которой ветвили. Если для некоторого $i > j$ имеет место неравенство $Fq + ci > F^*$, то достаточно ветвить только те следующие за Q вершины, в которых номер меньше i (в силу неравенств $0 \le c1 \le c2 \le \dots \le cn$).
- Правило 3. Пусть вершина Q задает решение xj (номер переменной ветвления j). Предположим, что мы включим в решение все последующие переменные (x_{j+1}=1, …). Тогда, должны выполняться все условия в матрице ограничений. Вершина Q может оказаться недопустимой из- за некоторых неравенств.

Инструкция по использованию программы

Для использования программы необходимо собрать проект, следуя инструкциям, приведенным на официальном сайте Microsoft (или не Microsoft) (для конкретно вашего оборудования и окружения) (или используя документацию вашего программного обеспечения). Основной файл – algdz31.cpp

К примеру:

- ***-cl /EHsc algdz31.cpp matrix.cpp***
- ***g++ -o main algdz31.cpp matrix.cpp***

Описание логики работы программы

Программа работает следующим образом:

1. При запуске программы проверяется наличие файла с входными данными и их наличие. Если все в порядке, данные считываются, строится матрица условий, производится сортировка по ухудшения удобства, управление передается основной функции.
2. Решение программы осуществляется следующим образом.
    - Изначально вектор использованных мер равен нулю (не используем ни одну меру, вектор заполнен нулями)
    - Осуществляется ветвление по первой переменной (берем ее равную 1(lvetv) и 0 (rvetv)).
    - В функции lvetv вычисляется удобство пользователей относительно текущего вектора, проверяется выполнение всех условий (защищено от всех угроз). Если защищено, то значение функции удобства сравнивается, если текущее меньше, то данная позиция запоминается, все дальнейшие ветвления опускаются. Если же не все условия выполнены, то по 2 и 3 условиям балаша проверяется, имеет ли смысл считать дальше, и если да – то до какой переменной ветвить (по умолчанию до последней). Дальше происходит ветвление по следующей переменной.
    - В функции rvetv просто осуществляется ветвление по следующей перемееной.

    Таким образом, осуществляется ветвление до тех пор, пока мы не дойдем до крайних положений (дальнейшее отброшенно или это последняя переменная).
3. В конце работы алгоритма решения получаем выходные данные, их записываем в выходной файл.

Используемые структуры данных

Для построения матрицы условий, хранения вектора ограничений, и названий защит используется контейнер std::valarray(в случае матрицы valarray*valarray). Это массив заданной длины. Почему используем именно его? Потому что valarray поддерживает операции поэлементного сложения, умножения и т.д. Также он позволяет находить сумму хранящихся в нем элементов. Данные операции используются для нахождения оптимального решения – поэлементное умножение текущего массива решения на массив ограничений и последующее сложение, проверки ограничений (так же). Нам не нужно изменять размер матрицы и массивов, значит нам не нужен динамический массив. Следовательно, под наши задачи valarray подходит идеально. Подробнее о нем тут.

Формат входных и выходных данных

Входные данные

Во входном файле записаны 2 числа: первое(N) – количество угроз безопасности, второе(M) – количество способов их решения.

Каждая последующие M строк содержат в себе:

1. N чисел (0 или 1) – защищает ли от i угрозы данная защита или нет.
2. Число C – на сколько снижается удобство пользователя
3. Строка S – название данного способа защиты

Программа рассчитана на корректный пользовательский ввод, но содержит защиту от дурака – на некоторые неправильные значения она ругается на пользователя.

Выходные данные

На выходе в первой строке выдаются названия способов, которые необходимо использовать.

Во второй строке выдается число – на сколько будет снижено удобство пользователя

Тесты

К программе приложены тесты. Для их запуска необходимо их скомпилировать (почти так же, как и основную программу, только основной файл test.cpp)

Файл

- 0.out – отсутствие входного файла
- 1.in – проверка на пустом файле
- 2.in – проверка на некорректных значениях
- 3.in – проверка на неправильном формате входных данных
- 4.in – проверка на защиту от одной угрозы (выбор наименьшего защищающего) (жадный алгоитм работает)
- 5.in – проверка на защиту от двух угроз (ЖА работает)
- 6.in – проверка на защиту от трех угроз (предположим, что если работает на первых трех, то и дальше работает) (ЖА работает)
- 7.in – проверка на большем числе угроз и защит
- 8.in – проверка, когда жадный алгоритм дает не оптимальное решение
- 9.in – совмещённые условия 7 и 8

К каждому входному тесту прилагается выходной файл *.out который содержит предполагаемый вывод программы.

Оценка программы

Оценка сложности

Так как данный алгоритм представляет собой частный случай метода ветвей и границ (то есть частичного перебора) то сложность в худшем случае $O(2^n)$.

Оценка используемой памяти

Программа хранит в себе - матрицу N*M (матрица защиты от угроз), 2 массива из M элементов (название защит от угроз и ухудшение пользовательских условий). Так как

вспомогательные переменные и данные не превышают N*M, то наша программа использует O(N*M) памяти.