УДК 656.2-50: 519.8

РЕАЛИЗАЦИЯ МЕТОДА ВЕТВЕЙ И ГРАНИЦ ДЛЯ РЕШЕНИЯ ЗАДАЧ КОММИВОЯЖЕРА С РАЗРЕЖЕННЫМИ МАТРИЦАМИ

М.П. РЕВОТЮК, М.К. КАРОЛИ, П.М. БАТУРА

Белорусский государственный университет информатики и радиоэлектроники
П. Бровки, 6, Минск, 220013, Беларусь

Поступила в редакцию 4 сентября 2013

Рассматривается способ ускорения процедуры метода ветвей и границ для решения асимметричных задач коммивояжера с разреженными матрицами. Предлагаемый способ использует наследование решений порождающих задач, при котором оценка вариантов порожденных задач о назначении проводится методом коррекции дерева кратчайших путей приращений. Реоптимизация дерева путей приращений на структурах смежности приводит к гарантированному снижению вычислительной сложности задачи на порядок.

Ключевые слова: задача коммивояжера, метод ветвей и границ, вычислительная сложность.

Постановка задачи

Задача коммивояжера, как известно, возникает во многих случаях оптимизации управления дискретными процессами, легко формулируется, но трудно решается. Формальная модель такой задачи в классической постановке имеет вид:

$$Z_{\min} = \min\left\{ \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \;\middle|\; \begin{array}{l} \sum_{i=1}^{n} x_{ij} = \sum_{j=1}^{n} x_{ij} = 1;\ x_{ij} \geq 0,\ i, j = \overline{1,n}; \\ u_i - v_j + nx_{ij} \leq n-1,\ i = \overline{2,n},\ \ j = \overline{2,n},\ i \neq j \end{array} \right\}. \quad (1)$$

Для решения так называемых асимметричных задач вида (1), когда $c_{ij} \neq c_{ji}, i, j \in \overline{1,n}$, наиболее эффективным из точных методов считается метод ветвей и границ [1]. Схема алгоритма метода ветвей и границ может использовать разные способы порождения дерева вариантов. Один из лучших среди известных подходов базируется на решении линейных задач о назначении (ЛЗН), анализе получающихся замкнутых циклов и, если таких циклов более одного, последующем переборе вариантов разрыва циклов. Размер дерева вариантов в этом случае оказывается наименьшим. Рекурсия обхода дерева ЛЗН строится на матрице расстояний, где разрывы циклов задаются назначением бесконечных значений длин запрещаемых дуг [1, 2].

Основной проблемой использования метода ветвей и границ, как и других процедур исчерпывающего поиска, являются повышенные требования к объему памяти. Прямолинейная реализация метода ветвей и границ для асимметричной задачи коммивояжера с квадратными матрицами размерностью n характеризуется потребностью в памяти порядка $O(n\cdot(n^2+2n))$. Реализация для таких задач разностной схемы организации процесса ветвления позволяет получить зависимость $O(n^2)$ [3].

Предмет рассмотрения – развитие способа построения эффективной как по памяти, так и быстродействию разностной схемы ветвления на множестве ЛЗН [3] для задач (1) с разреженными матрицами. Алгоритм решения ЛЗН будет явно учитывать возможность частичного наследования результатов решения порождающей задачи. Потенциальный эффект

реализации такого приема объясняется тем, что вычислительная сложность решения независимых ЛЗН – $O(n^3)$, а сложность пересчета после изменения строки матрицы ЛЗН – $O(n^2)$ [3]. В случае разреженных матриц вычислительная сложность решения отдельных ЛЗН снижается пропорционально доле значимых элементов матрицы [3,4]. Далее это используется как основа эффективной реализации метода ветвей и границ на структурах смежности неполных графов транспортных сетей.

Алгоритм ветвления и его переменные состояния

При прямолинейной реализации метода ветвей и границ время решения (1) в основном определяется временем решения в каждом узле дерева ЛЗН фиксированной размерности

$$\min\left\{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\cdot x_{ij} \;\middle|\; \sum_{i=1}^{n} x_{ij}=\sum_{j=1}^{n} x_{ij}=1;\; x_{ij}\ge 0;\; i,j=\overline{1,n}\right\}. \qquad (2)$$

С целью определения переменных состояния процесса поиска оптимального решения (1) рассмотрим процесс ветвления более детально. При этом на разреженность матрицы исходных данных пока не будем обращать внимания.

Ветвление дерева вариантов задачи коммивояжера удобно проводить, используя стратегию DFS (Depth First Search) на векторе решения

$$R=\left\{ i \mid x_{ij}=1,\; j=\overline{1,n}\right\}=\left\{ r(j), j=\overline{1,n}\right\}. \qquad (3)$$

Поиск оптимального решения начинается с создания глобальных объектов – вектора лучшего текущего назначения $R_{\min}$ и его оценки $Z_{\min}$ с начальными значениями $R_{\min}=\varnothing$ и $Z_{\min}=\infty$. Далее следует вызов процедуры анализа корневого узла дерева для матрицы C.

Процедура анализа узла на уровне l, получив матрицу C^l, включает следующие шаги.

Шаг 1. Решение ЛЗН $C^l \to R^l$ и оценка целевой функции

$$Z=\sum_{j=1}^{n} c_{r(j),j}\,. \qquad (4)$$

Шаг 2. Если $Z_{\min}<Z$, то выход.

Шаг 3. Поиск в решении цикла обхода вершин минимальной длины $r^l \subseteq R^l$.

Шаг 4. Если $r^l \subset R^l$, то есть цикл не гамильтонов, то переход к шагу 6.

Шаг 5. Сохранение результата – $Z_{\min}=Z$, $R_{\min}=R^l$, а затем – выход.

Шаг 6. Для всех вершин цикла поочередно создать копию матрицы $C^l \to C_i^{l+1}$, установить запрет посещения других вершин цикла и рекурсивно вызвать процедуру анализа узла, соответствующего матрице C_i^{l+1}, $i\in r^l$.

В настоящее время лучшим по времени решения ЛЗН (2) является метод кратчайшего пути приращений SAP (Shortest Augmenting Path)[4]. Развивая идею классического венгерского метода, метод SAP базируется на двойственной задаче линейного программирования

$$\max\left\{\sum_{i=1}^{n} u_i+\sum_{j=1}^{n} v_j \;\middle|\; c_{ij}-u_i-v_j\ge 0,\;\; i,j=\overline{1,n}\right\}. \qquad (5)$$

Для случая квадратных разреженных матриц известна версия алгоритма метода SAP, где графы транспортных сетей заданы структурами смежности вида FSF (Forward Star Form) [4]. Однако алгоритм метода SAP пригоден после модификации для реоптимизации ЛЗН с изменяющимися прямоугольными матрицами (включая разреженные), требуя сохранения лишь вектора потенциалов [3]. Таким образом, лишь в корневом узле дерева ЛЗН придется выполнить n итераций алгоритма метода SAP, тогда как в других узлах – по одной итерации.

Алгоритм решения ЛЗН методом SAP включает следующие шаги:

1) формирование начального назначения и потенциалов различающихся столбцов по результатам поиска минимальных элементов в строках;

2) для всех строк без назначенных столбцов выполнить поиск кратчайшего пути приращений к любому столбцу без назначения, коррекцию потенциалов столбцов и фиксацию найденного пути приращений от назначенного столбца до текущей строки.

Первый шаг можно опустить, решая ЛЗН инкрементной версией алгоритма венгерского метода [3, 5]. Потенциалы столбцов при этом вначале могут быть нулевыми: $v_j = 0,\ j = \overline{1,n}$.

Вектор решения R^l пригоден для выявления циклов, не являющихся гамильтоновыми. Можно заметить, что если k – некоторая вершина цикла в решении задачи (2), то последовательность $r^l(k) = \left\{ r(0) = k, \left\langle r(i) = R^l_{r(i-1)} \middle| r(i) \neq k \right\rangle \right\} \subseteq R^l$ только тогда соответствует гамильтонову циклу, когда условием остановки является $r(n-1) = k$, $k \in \overline{1,n}$. Если цикл не гамильтонов, т.е. $r^l(k) \subset R^l$, то необходимо породить множество задач уровня $l+1$. Для этого следует указать цикл минимальной длины, выбрав вершину входа в цикл: $k^l = \arg\min_k \left\{ \left| r^l(k) \right|, k \in \overline{1,n} \right\}$.

Правило порождения ЛЗН тривиально – для каждой вершины обнаруженного цикла необходимо запретить посещение других вершин этого цикла. При этом структура смежности очередной ЛЗН отличается от предыдущей одной строкой, в которой часть элементов заменяются значением, не меньшим значения $c_{\max} = \max_{i,j} \left\{ c_{ij} : i = \overline{1,n},\ j \in L_i \right\}$, где $L_i = \left\{ j \mid c_{ij} < \infty,\ j = \overline{1,n} \right\}$ – список потенциально назначаемых элементов строки, $i = \overline{1,n}$.

Построчное изменение структур смежности проводится по следующему закону:

$$c_{ij}^{l+1} = c_{ij}^l + c_{\max}\left(i \in r^l\left(k^l\right) \wedge\ j \in r^l\left(k^l\right) \right), i = \overline{1,n},\ j \in L_i. \qquad (6)$$

В этом случае дополнительный буфер для сохранения и восстановления элементов строк не требуется. Реализуемость такого способа очевидна, если в машинном слове, выделяемом для хранения весовых элементов c, возможно хранение значения $c_{\max} \cdot n$.

Шаблон класса решения задачи

Рассмотренная схема поиска решения задачи коммивояжера представлена полной версией исходного текста шаблонов классов на языке C++ (рис. 1–4).

```
template <class cost> struct Arc { // Представление элемента строки
 int j; // Индекс столбца матрицы (номер конечной вершины дуги графа)
 cost c; // Значение элемента матрицы (вес дуги графа)
};
template <class cost> struct FSF { // Структура смежности
 int n; // Количество строк
 valarray<int> in; // Индексы списков элементов строк
 valarray<Arc<cost>> ar; // Списки элементов строк
 FSF<cost>(valarray<int> &i,valarray<Arc<cost>> &a):
 in(i),ar(a) { n=in.size()-1; }
};
template <class cost> struct Iter { // Итератор списка элементов строки
 Arc<cost> *s, *f, *p;
 void reset() { p=s; }
 Iter<cost>(FSF<cost> &o, int i):
 s(&o.ar[o.in[i]]),f(&o.ar[o.in[i+1]]) { reset(); }
 Arc<cost> *operator()() { return (p<f)? p++:0; }
};
```

Рис. 1. Шаблоны представления разреженных матриц структурами смежности вида FSF

Исходные данные и текущее состояние процесса решения представлены структурой смежности FSF (рис. 1).

Атомарной задачей, соответствующей узлу дерева, здесь выступает процесс решения ЛЗН (рис. 2–3) и анализ результата (рис. 4). В каждом узле дерева проверяется наличие гамильтонова цикла, после чего порождаются новые узлы, если цикл не гамильтонов.

После этапа решения ЛЗН в случае необходимости ветвления имеем список порождаемых задач следующего уровня. Список задач полностью представлен элементами вектора решения (3) текущей ЛЗН.

```
template <class cost> struct Alts { // Представление узла дерева задач
 int e,i; // Индексы головного и текущего элемента цикла
 cost z; // Оценка текущего назначения
 valarray<Arc<cost> *> utmp;
 valarray<int> q,r,s; // Вектор решения и его инверсия
 valarray<cost> v; // Вектор потенциалов столбцов
 Alts(int n):v(n),utmp(n),q(n),r(n),s(n) { s=n; }
 void operator=(Alts &obj) {
 utmp=obj.utmp; z=obj.z; q=obj.q; r=obj.r; v=obj.v; }
};

template <class cost> class STSP { // Класс решения задачи коммивояжера
 Arc<cost> *p;
 int n,dots,tail; // Вектор наилучшего текущего решения
 FSF<cost> fdx;
 valarray<cost> dtmp;
 valarray<Arc<cost> *> stmp;
 valarray<int> q,r,mark,clst,prnt;
 cost z, c_max, inf;
 void lapi(Alts<cost> &xn, int e); // Итерация решения ЛЗН
 void splt(Alts<cost> &xn); // Порождение подзадач ЛЗН
 void best(Alts<cost> &xn); // Анализ структуры решения ЛЗН
 bool final(int *r, int &j);
 void expans(int j,cost &x);
 Arc<cost> *px(int i,int j) {
 for (Iter<cost> next(fdx,i); (p=next())&&(p->j!=j); );
 return p;
 }

public:

 STSP(FSF<cost> &fd): fdx(fd),n(fd.n),r(fd.n),dtmp(fd.n),stmp(fd.n),
 mark(fd.n),clst(fd.n),prnt(fd.n),q(fd.n),dots(0), mark(0) {
 c_max=0;
 for (int i=0; i<n; i++)
 for (Iter<cost> next(fdx,i); p=next(); ) if (c_max<p->c) c_max=p->c;
 ++c_max; z=inf=numeric_limits<cost>::max();
 }
 void store(Alts<cost> &xn) { // Сохранение назначения
 z=xn.z; r=xn.r;
 }

 void operator()() { // Старт решения в корне дерева задач
 Alts<cost> ar(n);
 ar.r=n; ar.v=0; // Решение ЛЗН методом SAP
 for (int i=0; i<n; i++) lapi(ar,i);
 ar.z=0;
 for (int i=0; i<n; i++) ar.z+=ar.utmp[i]->c;
 best(ar); // Анализ структуры решения ЛЗН
 }
};
```

Рис. 2. Шаблон класса решения задачи коммивояжера с разреженными матрицами

Итерации решения ЛЗН методом SAP для всех строк выполняются лишь в корне дерева вариантов. Функция реализации итераций (рис. 3) для других вариантов ЛЗН вызывается один раз, так как очередной вариант отличается от предыдущего всего одной строкой. При этом учитывается тот факт, что изменяемые элементы строки (согласно (6)) формально соответствуют запрету на назначение столбцов. Вычислительная сложность поиска оптимального назначения для нового варианта ЛЗН снижается на порядок.

```
template <class cost> void STSP<cost>::lapi(Alts<cost> &xn, int e){
 cost *v=&xn.v[0];
 cost x=inf,h,f; int i,j,k,last=n, *q=&xn.q[0], *r=&xn.r[0];
 ++dots; tail=0; dtmp=inf;
 for (Iter<cost> next(fdx,e); p=next(); ) {
 dtmp[j=p->j]=p->c-v[j]; prnt[j]=e; stmp[j]=p;
 expans(j,x);
 }
 if (tail) {
 if (!final(r,j)) {
 for (int head=n; tail--; ) {
 k=clst[tail]; i=r[k]; clst[--head]=k; h=px(i,k)->c-v[k]-x;
 for (Iter<cost> next(fdx,i); p=next(); )
 if ((mark[j=p->j]!=dots)&&((f=p->c-v[j]-h)<dtmp[j])) {
 dtmp[j]=f; prnt[j]=i; stmp[j]=p;
 if (f==x) {
 if (r[j]==n) goto fin;
 mark[j]=dots; clst[tail++]=j;
 }
 }
 if (tail==0) {
 x=inf; last=head;
 for (j=0; j<n; j++) if (mark[j]!=dots) expans(j,x);
 if (final(r,j)) break;
 }
 }
 fin: while (last<n) { k=clst[last++]; v[k]+=dtmp[k]-x; }
 }
 do { // Фиксация кратчайшего пути приращений
 i=prnt[j]; xn.utmp[i]=stmp[j]; r[j]=i; k=j; j=q[i]; q[i]=k;
 } while(i!=e);
 }
}
template <class cost> void STSP<cost>::expans(int j,cost &x) {
 cost f=dtmp[j];
 if (f<=x) {
 if (f<x) { tail=0; x=f; }
 clst[tail++]=j;
 }
}
template <class cost> bool STSP<cost>::final(int *r, int &j) {
 for (int k=0; k<tail; k++) {
 if (r[j=clst[k]]==n) return true;
 mark[j]=dots;
 }
 return false;
}
```

Рис. 3. Функции реализации основных итераций алгоритма метода SAP

Функции организации ветвления (рис. 4) осуществляют анализ вектора решения и выделение цикла минимальной длины, если решение не представляет гамильтонов цикл.

Ветвление организуется посредством построчного изменения и восстановления текущей структуры смежности. Изменения касаются лишь строк и столбцов, соответствующих вершинам разрываемого цикла. Реализация ветвления проведена на основе принципа ленивой инициализации рабочих массивов, используемых функциями итераций метода SAP.

Итерации предлагаемого подхода ассоциируются со строками матриц. Известно, что изменение оценок решения ЛЗН после изменения строки i экономно оценивается выражением $u_i^{k+1} - u_i^k = Z^{k+1} - Z^k$, $i \in \overline{1,n}$, где u_i^l – потенциалы строк, определяемые (5)[5]. В этом случае вычислительная сложность оценки результата решения ЛЗН – $O(1)$. Таким образом, прямолинейное вычисление выражения (4), сложность которого $O(n)$, не требуется.

```
template <class cost>
void STSP<cost>::best(Alts<cost> &xn) { // Анализ структуры решения ЛЗН
 dots++; xn.e=0;
 int k=0, m=0, *r=&xn.r[0];
 do { m++; mark[k]=dots; k=r[k]; } while (mark[k]!=dots);
 if (m<n) {
 for (int i=1; ; ) { // Поиск цикла минимальной длины
 while ((i<n)&&(mark[i]==dots)) i++;
 if (i==n) break;
 int k=i, t=0;
 do { t++; mark[k]=dots; k=r[k]; } while (mark[k]!=dots);
 if (m>t) m=t, xn.e=i;
 }
 splt(xn); // Порождение подзадач ЛЗН
 } else store(xn); // Сохранение лучшего из назначений
}

template <class cost>
void STSP<cost>::splt(Alts<cost> &xn) { // Порождение подзадач ЛЗН
 Alts<cost> an(n);
 int e=xn.e, &i=xn.i;
 do { an.s[e]=xn.e; e=xn.r[e]; } while (e!=xn.e);
 i=e;
 do {
 int k=xn.q[i];
 an=xn; an.r[k]=n; an.z=xn.z-xn.utmp[i]->c+xn.v[k];
 Iter<cost> next(fdx,i);
 while (p=next()) if (an.s[p->j]==e) p->c+=c_max;
 lapi(an,i); // Реоптимизация после изменения строки матрицы
 an.z+=an.utmp[i]->c-an.v[an.q[i]];
 if (z>an.z) best(an); // Анализ структуры решения ЛЗН
 next.reset();
 while (p=next()) if (an.s[p->j]==e) p->c-=c_max;
 } while ((i=xn.r[i])!=e);
}
```

Рис. 4. Функции реализации ветвления метода ветвей и границ

Построенный шаблон класса пригоден для непосредственного использования в однопоточных вычислениях. При распараллеливании шаблон применим для анализа отдельной ветви дерева вариантов. Основным достоинством рассмотренной здесь разностной схемы реализации ветвления на множестве ЛЗН является экономное использование памяти. Оценка потребности в дополнительной памяти стека вариантов ветвления – $(n-1)\cdot(n+1)$.

Экспериментальная оценка времени решения задач

В таблице приведены результаты экспериментов по оценке среднего времени решения задач коммивояжера предлагаемыми процедурами реализации метода ветвей и границ. Размеры матриц случайных исходных данных, генерируемых по равномерному закону распределения – n=50…150. Разреженные матрицы порождаемых задач фактически были представлены структурами смежности исходящих вершин (в форме FSF) соответствующих графов транспортных сетей. С целью оценки влияния разреженности матрицы на время решения фиксировались результаты для различных значений полустепени исхода вершин графов.

Оценки времени решения задач коммивояжера

Размерность (n) задачи	Среднее время решения, сек (Celeron 1,7 Ггц, 512 Мбайт)					
	Полустепень исхода вершин графа					
	$n/10$	$n/4$	$n/2$	$2n/3$	$3n/4$	$n-1$
50	0,003	0,008	0,019	0,024	0,032	0,036
60	0,004	0,012	0,028	0,040	0,048	0,052
70	0,005	0,013	0,047	0,063	0,078	0,088
80	0,011	0,016	0,102	0,128	0,160	0,172
90	0,012	0,017	0,117	0,137	0,169	0,186
100	0,019	0,024	0,136	0,192	0,224	0,244
110	0,040	0,048	0,180	0,338	0,512	0,596
120	0,055	0,067	0,253	0,358	0,400	0,448
130	0,061	0,080	0,279	0,385	0,444	0,534
140	0,083	0,116	0,488	0,621	0,724	0,862
150	0,122	0,144	0,548	0,769	1,027	1,193

Результаты эксперимента подтверждают ожидаемое сокращение времени решения задачи коммивояжера при увеличении степени разреженности матриц исходных данных.

Заключение

Реализация метода ветвей и границ предложенным открытым для расширения шаблоном класса для решения асимметричных задач коммивояжера базируется на наследовании решений порождающих задач о назначении, при котором оценка вариантов порожденных задач проводится методом коррекции дерева кратчайших путей приращений. Отображение дерева реализуется на реальные альтернативы развития путей, выделяемых на структуре смежности графа транспортной сети. Это приводит к гарантированному снижению вычислительной сложности решения задачи коммивояжера с разреженными матрицами в первом приближении на порядок. Дополнительная память для хранения наследуемых значений потенциалов столбцов и вектора решения не превышает объема $O(2n^2)$.

IMPLEMENTATION THE BRANCH AND BOUND METHOD FOR SOLVING THE TRAVELING SALESMAN PROBLEM WITH SPARSE MATRIX

M.P. REVOTJUK, M.K. QARALEH, P.M. BATURA

Abstract

The problem of the solution of asymmetric traveling salesman problem with sparse matrix, based on branch and bound techniques with linear assignment problems relaxation, is considered. Inheritance of the result's data of previous problems and its reoptimization allows to decreasing time of reception of the new solution on branch's tree path. The reoptimization algorithm, based on a Shortest Augmenting Path method, is offered.

Список литературы

1. *Miller D., Pekny J.* // Science. 1991. Vol. 251. P. 754–761.
2. *Mahshid A.F., Rosnah M.Y.* // European Journal of Scientific Research. 2009. Vol. 29. № 3. P. 349–359.
3. *Ревотюк М.П., Батура П.М., Полоневич А.М.* // Докл. БГУИР. 2011. № 3 (57). С. 56–62.
4. *Jonker R., Volgenant A.* // Computing. 1987. Vol. 38. P. 325–340.
5. *Ревотюк М.П., Кароли М.К., Батура П.М.* //Докл. БГУИР. 2013. № 5 (75). С. 30–36.