УДК 656.2-50: 519.8

РЕОПТИМИЗАЦИЯ КРАТЧАЙШИХ ПУТЕЙ ПРИРАЩЕНИЙ ПРИ РЕШЕНИИ АСИММЕТРИЧНЫХ ЗАДАЧ КОММИВОЯЖЕРА

М.П. РЕВОТЮК, П.М. БАТУРА, А.М. ПОЛОНЕВИЧ

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

Поступила в редакцию 1 апреля 2011

Рассматривается способ ускорения решения асимметричной задачи коммивояжера методом ветвей и границ с ветвлением на задачах о назначении. Предлагаемый способ использует наследование решений порождающих задач, при котором оценка вариантов порожденных задач проводится методом коррекции дерева кратчайших путей приращений. Реоптимизация дерева путей приводит к снижению вычислительной сложности задачи на порядок.

Ключевые слова: задача коммивояжера, разностная схема, вычислительная сложность.

Постановка задачи

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

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

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. Исторически первый способ, предложенный Литтлом, использует операцию приведения матриц. Современный подход базируется на решении линейных задач о назначении (ЛЗН), анализе получающихся замкнутых циклов и, если таких циклов более одного, последующем переборе вариантов разрыва циклов. Установлено, что размер дерева вариантов в этом случае оказывается меньше. Рекурсия обхода дерева ЛЗН строится на матрице расстояний, где разрывы циклов задаются назначением бесконечных значений длин запрещаемых дуг [2].

Основной проблемой использования метода ветвей и границ, как и других процедур исчерпывающего поиска, является повышенные требования к объему памяти. В частности, прямолинейная реализация метода ветвей и границ для асимметричной задачи коммивояжера размерностью n характеризуется потребностью в памяти порядка $n \cdot (n^2 + 2n)$. Использование разностной схемы представления состояний процесса ветвления позволяет получить зависимость $O(n^2)$. Например, реализация такой схемы для случая ветвления на двоичных деревьях методом Литтла характеризуется потребностью в памяти $n^2 + 2n \cdot (n+3)$ [3].

Предмет рассмотрения – способ построения эффективной как по памяти, так и быстродействию разностной схемы ветвления на множестве ЛЗН. 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 пересчета после изменения строки матрицы ЛЗН – $O(n^2)$ [3]. Для простоты изложения далее будут опущены приемы сокращения количества ветвей дерева вариантов [1, 2, 5], не влияющие на общность предлагаемых решений.

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

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

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

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

Шаг 1. Решение ЛЗН $C^k \to R^k$ и оценка целевой функции $Y=\sum_{j=1}^{n} c_{r(j),j}$.

Шаг 2. Если $Y_{\min}<Y$, то выход.

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

Шаг 4. Если $r^k \subset R^k$, т.е. цикл не гамильтонов, то переход к шагу 6.

Шаг 5. Сохранение результата $-Y_{\min}=Y$, $R_{\min}=R^k$, а затем – выход.

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

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

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

В известных реализациях метода SAP чаще всего внимание уделяется эффективности решения независимых ЛЗН вида (2) с квадратной матрицей. Однако алгоритм метода SAP пригоден после модификации для реоптимизации задач с изменяющимися прямоугольными матрицами, требуя сохранения лишь вектора потенциалов.

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 не назначенному столбцу, коррекцию потенциалов столбцов и фиксацию найденного пути приращений от назначенного столбца до текущей строки.

Первый шаг можно опустить, тогда ЛЗН будет фактически решена инкрементной версией алгоритма венгерского метода [3]. Потенциалы столбцов при этом должны быть нулевыми.

Изменение элементов матрицы ЛЗН влечет необходимость пересмотра результата оптимизации, если только меняется позиция нулевого элемента после операции приведения. Действительно, увеличение элемента $c_{ij} \leftarrow c_{ij}^*$ матрицы задачи (2), когда $x_{ij} = 0$, не меняет решения для любых $i, j \in \overline{1,n}$. Аналогично, уменьшение элемента $c_{ij} \leftarrow c_{ij}^*$, когда $x_{ij} = 1$, не нарушает соотношения $c_{ij} = u_i + v_j$ в задаче (2). В общем случае, для решения ЛЗН с измененной матрицей требуется повтор итераций назначения лишь для строк

$$M^* = \left\{ i \middle| (c_{ij}^* > c_{ij}) \wedge (x_{ij} = 0) \vee (c_{ij}^* < c_{ij}) \wedge (x_{ij} = 1), j = \overline{1,n} \right\}. \quad (4)$$

При этом достигается снижение вычислительной сложности пересчета задачи (1) на величину $\left(n - \left|M^*\right|\right) \cdot O(n^2)$. Из выражения (4) следует, что релаксация на множестве ЛЗН при решении задачи коммивояжера методом ветвей и границ соответствует условию $\left|M^*\right| = 1$. Последнее объясняет сокращение времени решения задачи коммивояжера приблизительно в $n-1$ раз.

Из (5) следует, что изменение элементов матриц в любом узле дерева вариантов производится только путем увеличения значения. Учитывая, что потенциал измененной строки будет уточняться на итерации пересчета, предварительное уточнение потенциалов не требуется.

Вектор R^l пригоден как для выявления циклов, не являющихся гамильтоновыми, так и исключает необходимость хранения решения в матричном виде.

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 (для наследования)

$$k^l = \arg\min_k \left\{ \left| r^l(k) \right|, k \in \overline{1,n} \right\}.$$

Правило порождения ЛЗН тривиально – для каждой вершины обнаруженного цикла необходимо запретить посещение других вершин этого цикла. При этом матрица очередной ЛЗН отличается от предыдущей одной строкой, в которой часть элементов заменяются значением, не меньшим значения

$$c_{\max} = \max_{i,j} \left\{ c_{ij} : i \neq j, i = \overline{1,n}, \ j = \overline{1,n} \right\}. \quad (5)$$

Построчное изменение матриц проводится по следующему закону:

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

Так как изменения матрицы C выполняются построчно, то достаточно сохранить изменяемые элементы в буфере размером n. Однако запрет на использование элемента матрицы можно установить смещением его текущего значения на величину $c_{\max}$. В этом случае дополнительный буфер для сохранения элементов строки не требуется. Надежность реализации такого способа очевидна, если выполняется условие размещения в машинном слове, выделяемом для хранения элементов матрицы, значения $c_{\max} \cdot n$.

Фильтрация бесперспективных вариантов

В случае решения задачи коммивояжера естественно использовать взаимосвязь матриц рекурсивно порождаемых ЛЗН. Например, можно учитывать возможности прерывания итераций решения задачи ЛЗН. При решении ЛЗН на основе (3) имеется возможность получения нижних оценок целевой функции. Это позволяет прервать анализ бесперспективного варианта матрицы, используя глобальное значение рекордной оценки среди просмотренных листьев дерева [3]. На основании теории двойственности, нижняя оценка целевой функции $\tilde{Z}^k$ на итерации k решения ЛЗН венгерским методом определяется выражением

$$\tilde{Z}^k = \sum_{i=1}^{n} u_i^k + \sum_{j=1}^{n} v_j^k, \qquad (7)$$

где u_i^k, v_j^k – значения потенциалов строк и столбцов на этой итерации. Если выясняется, что $\tilde{Z}^k \geq Z_{rec}$, где Z_{rec} – значение целевой функции лучшего из просмотренных вариантов, то итерации анализа рассматриваемой ЛЗН можно прекратить. Нижняя оценка целевой функции в этот момент совпадает с оценкой оптимального решения, но выражение (7) эффективнее для вычисления по сравнению с прямолинейным использованием выражения (1).

В алгоритме метода SAP промежуточные итерации проводятся с использованием лишь потенциалов столбцов v_j^k, $j = \overline{1,n}$. Потенциалы строк можно определить после завершения итераций, используя соотношение $u_{r(j)} = c_{r(j),j} - v_j$, $j = \overline{1,n}$. В этом случае $r(j) \in \overline{1,n}$, $j = \overline{1,n}$, что означает назначение каждому столбцу матрицы некоторой ее строки. Однако на промежуточных итерациях алгоритма метода SAP столбцы без назначенных строк можно пометить, например, недействительным значением индекса элемента вектора решения. В результате нижнюю оценку целевой функции на любых итерациях алгоритма метода SAP можно получить следующим образом:

$$\tilde{Z}^k = \sum_{j=1}^{n} c_{r(j),j} \cdot (r(j) > 0).$$

Шаблон класса решения задачи

Рассмотренная схема поиска решения задачи коммивояжера представлена полной версией исходного текста шаблона класса на языке C++(рис. 1–3).

```
template <class cost> struct task { // Дескриптор узла дерева вариантов
 int e,i; // Индексы головного и текущего элемента цикла
 vector<int> r; // Вектор решения - назначенные строки для столбцов
 vector<cost> v; Вектор потенциалов столбцов
 task(int n):v(n),r(n) {}
};
template <class cost> class TSP { Класс решения задачи коммивояжера
 int dots,n,*mark,*cols; // Вектор наилучшего текущего решения
 cost **c, y, c_max, inf;
 cost values(int *r) { // Оценка целевой функции текущего назначения
  cost v=0;
  for (int j=0; j<n; j++) v+=c[r[j]][j];
  return v;
 }
 vector<cost> dtmp;
 vector<int> mark,rows,cols,clst,prnt;
 cost laps(task<cost> *xn); // Решение ЛЗН методом SAP
 void lapi(task<cost> *xn, int e); // Итерация решения ЛЗН
 cost lapn(task<cost> *xn, task<cost> *xo); // Реоптимизация ЛЗН
 void splt(task<cost> *xn); // Порождение подзадач ЛЗН
 void best(task<cost> *xn, cost x); // Анализ структуры решения ЛЗН
public:
 TSP(int N,cost **C): // размерность и указатель массива исходных данных
  n(N),c(C),dots(0),mark(N),dtmp(N),rows(N),cols(N),clst(N),prnt(N) {
  c_max=0;
  for (int i=0; i<n; i++) for (int j=0; j<n; j++)
   if ((i!=j)&&(c_max<c[i][j])) c_max=c[i][j];
  y=inf=(++c_max)*n;
  for (i=0; i<n; clst[i]=i, mark[i++]=dots) c[i][i]=c_max;
 }
 void store(cost x, int *r) { // Сохранение назначения
  y=x;
  for (int j=0; j<n; j++) cols[j]=r[j];
 }
 void operator()() { // Старт решения в корне дерева задач
  task<cost> ar(n);
  best(&ar,laps(&ar));
 }
 int *solution() { return &cols[0]; } // Выборка вектора решения задачи
};
```

Рис. 1. Шаблон класса решения задачи коммивояжера

Атомарной задачей, соответствующей узлу дерева, здесь выступает процесс решения ЛЗН (рис. 2) и анализ результата (рис. 3). При этом проверяется наличие гамильтонова цикла и 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> // Решение ЛЗН методом SAP
inline cost TSP<cost>::laps(task<cost> *xn) {
 vector<int> unchoozen_rows(n);
 cost x,h,*a,*v=&xn->v[0];
 int i,j,k,m,*r=&xn->r[0];
 for (i=0; i<n; i++) rows[i]=-1;
 for (j=n; --j>=0; r[j]=k, v[j]=x) { // Инициализация вектора назначений
  for (x=c[k=0][j], i=1; i<n; i++)
   if ((h=c[i][j])<x) { x=h; k=i; }
  if (rows[k]<0) rows[k]=j; else k=-1;
 }
 for (m=i=0; i<n; i++) // Инициализация вектора потенциалов столбцов
  if ((k=rows[i])<0) unchoozen_rows[m++]=i;
  else {
   for (x=inf, a=c[i], j=0; j<n; j++)
    if ((j!=k)&&((h=a[j]-v[j])<x)) x=h;
   v[k]-=x;
  }
 for (k=m; --k>=0; ) { // Окончательное формирование назначения строк
  lapi(xn,unchoozen_rows[k]);
  for (x=0, j=0; j<n; j++) // Оценка нижней границы целевой функции
   if (((i=r[j])>=0)&&((x+=c[i][j])>y)) return inf;
 }
 return values(r);
}
template <class cost> // Итерация решения ЛЗН методом SAP
 inline void TSP<cost>::lapi(task<cost> *xn, int e) {
 cost x,h,f,*a=c[e],*v=&xn->v[0];
 int i,j,k,*r=&xn->r[0];
 for (j=0; j<n; prnt[j++]=e) dtmp[j]=a[j]-v[j];
 for (int head=0, tail=0, last;;) {
  if (tail==head) {
   last=head; x=dtmp[clst[tail++]];
   for (k=tail; k<n; k++) if ((h=dtmp[j=clst[k]])<=x) {
    if (h<x) { x=h; tail=head; }
    clst[k]=clst[tail]; clst[tail++]=j;
   }
   for (k=head; k<tail; k++) if (r[j=clst[k]]<0) goto backtrack;
  }
  i=r[j=clst[head++]]; a=c[i]; h=a[j]-v[j]-x;
  for (k=tail; k<n; k++) {
   j=clst[k];
   if ((f=a[j]-v[j]-h)<dtmp[j]) {
    prnt[j]=i;
    if (f==x) {
     if (r[j]<0) goto backtrack;
     clst[k]=clst[tail]; clst[tail++]=j;
    }
    dtmp[j]=f;
   }
  }
 }
 backtrack: do { // фиксация кратчайшего пути приращений
  i=r[j]=prnt[j]; k=j; j=rows[i]; rows[i]=k;
 } while(i!=e);
 for (k=0; k<last; v[j]+=dtmp[j]-x) j=clst[k++];
}
```

Рис. 2. Шаблоны основных функций решения задач о назначении

Функции организации ветвления (рис. 3) осуществляют анализ вектора решения и выделяют цикл минимальной длины, если решение не представляет гамильтонов цикл. Ветвление организуется посредством построчного изменения и восстановления текущей матрицы стоимостей. Изменение касается лишь строк и столбцов, соответствующих вершинам разрываемого цикла.

Построенный шаблон класса пригоден для непосредственного использования в однопоточных последовательных 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 памяти – $n^2+(n-1)\cdot(n+1)$, где первое слагаемое – исходная матрица задачи (1), второе – память стека вариантов ветвления.

```
template <class cost> // Анализ структуры решения ЛЗН
inline void TSP<cost>::best(task<cost> *xn, cost x) {
 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) { // Поиск цикла минимальной длины
  for (int i=1; ; ) {
   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); // Переход к ветвлению
 } else store(x,r); // Сохранение лучшего из просмотренных назначений
}
template <class cost> // порождение подзадач ЛЗН
inline void TSP<cost>::splt(task<cost> *xo) {
 task<cost> an(n);
 xo->i=xo->e;
 do {
  cost *b=c[xo->i];
  int j=xo->r[xo->i];
  do { b[j]+=c_max; j=xo->r[j]; } while (j!=xo->i);
  cost x=lapn(&an,xo); // Реоптимизация после изменения строки матрицы
  if (y>x) best(&an,x); // Анализ структуры решения ЛЗН
  j=xo->r[xo->i];
  do { b[j]-=c_max; j=xo->r[j]; } while (j!=xo->i);
 } while ((xo->i=xo->r[xo->i])!=xo->e);
}
template <class cost> // Реоптимизация ЛЗН
inline cost TSP<cost>::lapn(task<cost> *xn, task<cost> *xo) {
 for (int j=0; j<n; xn->v[j]=xo->v[j], j++) rows[xn->r[j]=xo->r[j]]=j;
 rows[xo->i]=xn->r[rows[xo->i]]=-1;
 lapi(xn,xo->i); // Включение строки в дерево приращений
 return values(&xn->r[0]);
}
```

Рис. 3. Функции организации ветвления задач о назначении

Экспериментальная оценка времени решения задач

В таблице приведены результаты экспериментов по оценке среднего времени решения задач коммивояжера процедурами реализации классического алгоритма венгерского метода [5] и алгоритма кратчайшего пути приращений (рис. 1–3). Результаты фиксировались для трех вариантов применения алгоритмов каждого вида: 1) независимое решение ЛЗН; 2) реоптимизация порождаемых ЛЗН в узлах дерева вариантов; 3) реоптимизация порождаемых ЛЗН с прерыванием решения бесперспективных вариантов. Размеры матриц случайных исходных данных, генерируемых по равномерному закону распределения – n=50…150.

Оценка времени решения задач коммивояжера различными процедурами

Размерность задачи (n)	Среднее время решения, сек (Celeron 1,7 Ггц, 512 Мбайт)					
	Венгерский метод			Метод SAP		
	1	2	3	1	2	3
50	0,183	0,028	0,009	0,036	0,011	0,009
60	0,342	0,036	0,012	0,059	0,036	0,013
70	0,576	0,082	0,027	0,126	0,027	0,022
80	0,879	0,122	0,038	0,282	0,047	0,043
90	2,513	0,241	0,081	0,299	0,050	0,046
100	3,234	0,283	0,099	0,481	0,084	0,061
110	9,854	0,344	0,124	1,018	0,188	0,146
120	5,599	0,666	0,238	1,038	0,164	0,112
130	8,048	0,861	0,320	1,545	0,101	0,134
140	14,983	0,607	0,228	2,726	0,281	0,216
150	14,447	0,696	0,260	4,571	0,157	0,293

Результаты эксперимента подтверждают ожидаемое преимущество реализации разностной схемы метода SAP, но эффект от прерывания анализа бесперспективных вариантов для такого метода оказался незначительным.

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. Это приводит к снижению вычислительной сложности задачи коммивояжера в первом приближении на порядок. Дополнительная память для хранения наследуемых значений потенциалов столбцов и вектора решения не превышает объема $O(2n^2)$.

REOPTIMIZATION OF THE SHORTEST AUGMENTING PATHS IN ASYMMETRIC TRAVELING SALESMAN PROBLEM

M.P. REVOTJUK, P.M. BATURA, A.M. POLONEVICH

Abstract

The problem of the solution of asymmetric traveling salesman problem, 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 algorithm of reoptimization, based on a Shortest Augmenting Path method, is offered.

Литература

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. №1, С. 55–62.
4. *Jonker R., Volgenant A.* // Computing. 1987. Vol. 38. P. 325–340.
5. *Zhang W.* // Journal of Artificial Intelligence Research. 2004. Vol. 20. P. 471–497.