УДК 519.852(075.8)+519.87(075.8)

ТЕХНИКА ПРОГРАММНОЙ РЕАЛИЗАЦИИ ПОТОКОВЫХ АЛГОРИТМОВ

А.В. Панюков, В.А. Телегин

SOFTWARE ENGINEERING OF THE FLOW ALGORITHMS

A.V. Panyukov, V.A. Teleghin

Рассмотрены способы реализации ведущего преобразования в схеме симплекс-алгоритма для задач транспортного типа, позволяющие осуществлять перестройку базисного дерева за время линейное от числа вершин сети, существенно сократив при этом число проверок условия оптимальности. Техника программной реализации указанных процедур проиллюстрирована в исходном тексте абстрактного класса `transport` и классов `Transshipment` и `Transportation`, предназначенных для решения и постоптимизационного анализа транспортных задач соответственно в сетевой и матричной постановках.

Ключевые слова: транспортная задача, задача об оптимальном потоке, алгоритмы. структуры данных, объектно-ориентированное программирование, техника программной реализации

The authors consider the ways of the software engineering of the pivot procedures in the scheme of the primal simplex algorithm for flow problems, which enable to rearrange the redix tree within the linear time of the vortex of network number, considerably decrease the number of checks of the optimality condition. The technique of the software implementation of these procedures is given in the source text of the abstract class transport and classes Transshipment and Transportation which are destined for solving and post-organizational analysis of the transportation problems in the cross network and matrix definitions respectively.

Keywords: transportation problem, transhipment problem, algorithms data structures, object-oriented programming, software engineering

Введение

Потоковые задачи: построение оптимального потока в конечной сети, задача о максимальном потоке, транспортная задача, задача о назначении и задача о кратчайшем пути, – были одними из первых, давших полезные для практики сетевые оптимизационные модели. В работах 50-х годов [1] рассматривались прямые алгоритмы (метод потенциалов) решения данных задач. В последующих работах был предложен ряд алгоритмов другого типа, в частности, алгоритм беспорядка [3, 4] и двойственный симплекс-метод [5, 6]. Опыт эксплуатации и вычислительные эксперименты показали, что новые алгоритмы требуют меньшего числа изменений текущего базисного решения (ведущих преобразований) при выполнении симплекс-алгоритма. Это привело в середине 60-х годов к мнению, что наиболее жизнеспособным является алгоритм беспорядка [3, 4]. Из-за убежденности в неэффективности прямого алгоритма оставались незамеченными предложения [2, 7] о

способах увеличения скорости выполнения ведущего преобразования прямого симплекс-метода. Большинство из этих предложений включены в современные реализации прямых алгоритмов построения оптимального потока в конечной сети [16, 18].

Развитие современного математического обеспечения решения задач об оптимальном потоке началось в 70-х годах [8, 9, 10, 11, 12]. Широкое сравнение алгоритмов решения задачи об оптимальном потоке было проведено в работах [8, 13] и на Всесоюзном конкурсе «Транспорт-81». Эти исследования показали, что прямой симплекс-метод значительно превосходит остальные известные методы. Появившиеся полиномиальные алгоритмы для задачи об оптимальном потоке [19, 20, 21, 23] также не составили конкуренцию прямому симплекс-алгоритму при решении практических задач.

Заметим, что вычислительные эксперименты [8, 13] проводились для «реальных»сетей, которые задаются разреженным графом. Однако, построение оптимального потока требуют алгоритмы для многих задач размещения и синтеза сетей, в частности, для задач размещения прямоугольных объектов с минимальной длиной связывающей их сети [15, 25]. Отличительными особенностями данных задач являются: 1) в определении сети используется полный граф; 2) необходимо решать последовательность локально возмущенных задач, отличающихся параметрами пары вершин и инцидентных им дуг. Вычислительная сложность прямого симплекс-алгоритма существенно зависит от стратегии проверки условия оптимальности для небазисных дуг. Использование известных стратегий приводит на заключительных итерациях алгоритма к проверке условия оптимальности для всех небазисных дуг, хотя большинство из них удовлетворяет данному условию. Следствие – неоправданные затраты временных ресурсов при решении задач указанного класса. Например, сложность определения решения возмущенной задачи по решению исходной задачи оказывается почти такой же, как и сложность решения исходной задачи. Для устранения этого парадокса в работе [13] предлагают использовать двойственный симплекс-алгоритм.

В данной работе изложен способ учета дуг, заведомо удовлетворяющих условию оптимальности, основанный на *упорядочении изучения базисного дерева* [17, 22, 24]. При использовании данного способа из поиска очередного кандидата для ввода в базис исключаются дуги, для которых установлено выполнение условия оптимальности. Предложенные в работе структуры данных и процедуры для них позволяют уменьшить вычислительную сложность всей итерации прямого симплекс-метода, а не только ведущего преобразования как в известных алгоритмах. Рассмотренный способ существенно повышает эффективность процедур, использующих прямой алгоритм решения задачи об оптимальном потоке в схеме метода ветвей и границ, при пост-оптимизационном анализе и т.п. [15]. Кроме того, данный способ открывает пути распараллеливания вычислений при решении задачи об оптимальном потоке.

Предлагаемый способ использован при разработке класса `Transport`, предоставляющего средства для решения и постоптимизационного анализа транспортных задач в сетевой и матричной постановках, а также задачи о назначении. Исходный текст данного класса на языке `C++` приведен в статье для иллюстрации используемых структур данных, алгоритмов и техники их программной реализации.

1. Постановка задачи

Пусть (N, A) – связный граф, N – множество вершин, A – множество дуг. В каждой вершине $n \in N$ задана конечная величина $q(n) \in \mathbb{Z}$, которая сопоставляется либо величине произведенного продукта ($q(n) \geq 0$), либо величине потребляемого продукта ($q(n) \leq 0$). Для каждой дуги $(n, m) \in A$ заданы стоимость $c(n, m)$ транспортировки единицы

продукта от вершины n до вершины m и величина $k(n,m)$ пропускной способности данной транспортной коммуникации. Рассматриваемая задача состоит в определении потока $x(n,m)$ вдоль каждой дуги $(n,m) \in A$, удовлетворяющего требованиям

$$\sum_{(n,m)\in A} c(n,m)x(n,m) \quad \to \max_x, \tag{1}$$

$$(\forall n \in N)\left(\sum_{m:(m,n)\in A} x(m,n) - \sum_{m:(n,m)\in A} x(n,m) = q(n)\right), \tag{2}$$

$$(\forall (n,m) \in A)\,(0 \le x(n,m) \le k(n,m))\,. \tag{3}$$

Задача (1) – (3), известная как задача об оптимальном потоке (или как транспортная задача в сетевой постановке), является наиболее общей формой линейных однопродуктовых потоковых задач. Такие задачи, как задача о максимальном потоке, транспортная задача, задача о назначении и задача о кратчайшем пути, являются частными случаями задачи (1) – (3). Однако, учет особенностей частных задач позволяет строить для них более эффективные алгоритмы. В частности, особенностью транспортной задачи в матричной постановке является двудольность графа (N, A).

2. Прямой симплекс-алгоритм

Прямой критерий оптимальности задачи (1) – (3) дает [1]

Теорема 1. *Необходимым и достаточным условием оптимальности допустимого решения $x(n,m)$, $(n,m) \in A$ задачи (1) – (3) является существование величин $u(n)$, $n \in N$ (потенциалов вершин), таких, что*

$$0 < x(n,m) < k(n,m) \quad \Leftrightarrow \quad u(m) - u(n) = c(n,m), \tag{4}$$

$$x(n,m) = 0 \quad \Leftrightarrow \quad u(m) - u(n) \ge c(n,m), \tag{5}$$

$$x(n,m) = k((n,m) \quad \Leftrightarrow \quad u(m) - u(n) \le c(n,m). \tag{6}$$

Переменные допустимого базисного решения задачи (1) – (3), удовлетворяющие условию 4, соответствуют дугам, образующим остовное дерево T графа (N, A). Общепринято дерево T отождествлять с базисом, а образующие его дуги называть базисными.

Алгоритм, построенный по схеме прямого симплекс-метода, состоит в выполнении следующих основных шагов.

- Шаг 0. Инициализация. Построить базис (возможно содержащий искусственные дуги с высокой стоимостью) и соответствующий допустимый базисный поток x. Пометить одну из вершин как корень. Определить потенциалы $u(i)$ вершин $i \in N$ из уравнений (4) для базисных дуг.

- Шаг 1. Найти дугу для ввода в базис. Текущее решение является оптимальным, если выполнены условия (5) – (6), иначе небазисная дуга, не удовлетворяющая данным условиям, выбирается для ввода в базис.

- Шаг 2. Выполнить ведущее преобразование.

  – Шаг 2.1. Найти выводимую из базиса дугу. Дуга для вывода из базиса определяется при обходе единственного цикла, образованного при добавлении вводимой в базис дуги. Изменение потока вдоль вводимой в базис дуги от его

текущей границы вызовет изменение потока по дугам цикла. При максимальном допустимом изменении поток на одной или нескольких дугах цикла достигнет своей верхней или нижней границы. Из таких дуг для вывода из базиса выбирается дуга, ближайшая к корневой вершине в текущем базисном дереве.

– Шаг 2.2. Выводимая и вводимая дуги обмениваются своими базисным/небазисным статусами. Вычисляются обновленные базисный поток и потенциалы вершин. После чего осуществляется переход на Шаг 1.

Результативность и конечность данного алгоритма общеизвестны и в дальнейшем не обсуждаются.

В известных реализациях прямого симплекс-метода трудоемкость выполнения шага 2 линейна относительно числа вершин сети. В то же время трудоемкость шага 1 изменяется от итерации к итерации, достигая на последних итерациях величины, пропорциональной числу дуг сети.

3. Структуры данных. Интерфейс с программами пользователя

Для представления базисного дерева в компьютере одна из его вершин $r \in N$ выбирается в качестве корня. Если вершина j лежит на пути от вершины i к корню r, то вершину j называют *предшественником* i, а вершину i – *преемником вершины* j. Очевидно, что непосредственные предшественники и преемники являются вершинами, инцидентными базисным дугам. Следовательно, дерево T может быть представлено функцией $P: N \to N$, где $P(i)$ есть непосредственный предшественник вершины i, $P(r) = r$. Следует отметить, что $P(i)$ только указывает вершину, смежную вершине i, но не определяет ориентацию дуги, инцидентной вершинам i и $P(i)$. Для определения ориентации базисных дуг определим булеву функцию

$$Pos: N \to \{\texttt{True}, \texttt{False}\}: \begin{cases} Pos(i) = \texttt{True} \Rightarrow (i, P(i)) \in AT, \\ Pos(i) = \texttt{False} \Rightarrow (P(i), i) \in AT, \end{cases}$$

где AT – множество базисных дуг (дуг дерева T).

На рис. 1(a) показаны дерево с корнем в вершине 1, определены значения функции P, а также дополнительных функций $D: N \to N$, $B: N \to N$ и $L: N \to N$, которые будут использованы в программной реализации процедур, рассматриваемых в статье. Первая из дополнительных функций $D: N \to N$ задает прямой порядок обхода дерева T (т.е. порядок, при котором посещение предшественников осуществляется до посещения преемников). Следующая функция $B: N \to N$ является функцией обратной D, т.е. если $D(i) = j$ то $B(j) = i$. Функция B определяет концевой порядок обхода дерева T, согласованный с прямым порядком обхода, заданным функцией D. Наконец, функция $L: N \to N$ определяет последнего преемника $L(i)$ вершины i при прямом обходе. Данные функции использованы, например, в работе [11], для построения эффективных алгоритмов ведущего преобразования базиса (шага 2 в общей схеме). В данной работе эти же функции используются в модифицированном алгоритме ведущего преобразования, позволяющем существенно сократить количество проверок выполнения условия оптимальности, т.е. среднюю вычислительную сложность шага 1 в общей схеме. Основные этапы изменения базисного дерева, показанного на рис. 1(a), при выполнении модифицированного ведущего преобразования показаны на рис. 1(b – d). Все процедуры модифицированного ведущего преобразования реализованы как методы абстрактного класса `Transport`.

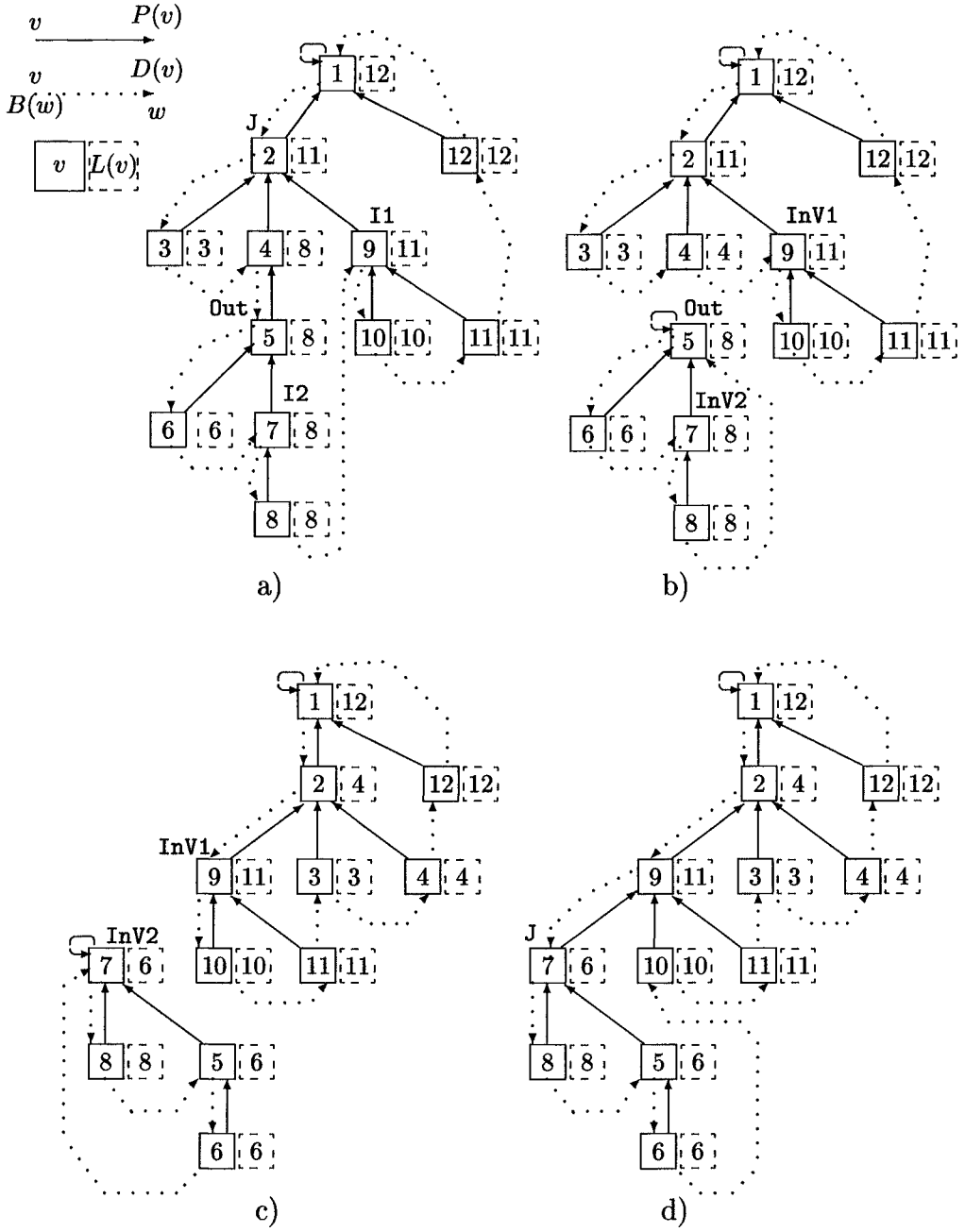


Рис. 1. Базисное дерево и определенные на нем функции

Стоимости $c(\cdot)$ и пропускные способности $k(\cdot)$ дуг сети будем кодировать как недиагональные элементы квадратных матриц C и K размера $|N|$. Далее считаем $N = \{i \in \mathbf{N} : 1 \le i \le |N|\}$. Строки матриц соответствуют началам дуг, а столбцы – концам. Достижение потоком в небазисной дуге $(i,j) : \ i \ne P(j), j \ne P(i)$ величины пропускной способности будем кодировать изменением знака величины $K(i,j)$. Таким образом, далее приняты следующие соглашения:

$$c(i,j) = C(i,j),\ k(i,j) = |K(i,j)|\,, \quad i,j \in N :\ i \ne j, \tag{7}$$

$$x(i,j) = \begin{cases} 0, & \text{если } K(i,j) \ge 0, \\ k(i,j), & \text{если } K(i,j) < 0, \end{cases} \quad i,j \in N :\ i \ne P(j), j \ne P(i). \tag{8}$$

В диагональных элементах матриц C и K равны будем кодировать соответственно потенциалы вершин и потоки (с учетом ориентации) в базисных дугах, т.е.

$$C(i,i) = u(i),\ K(i,i) = \begin{cases} x(i,P(i)), & \text{если } Pos(i) = \texttt{True}, \\ -x(P(i),i), & \text{если } Pos(i) = \texttt{False}. \end{cases} \tag{9}$$

На рис. 2 определены типы и структуры данных, используемые для представления базисного дерева и определения класса `Transport` и его потомков. Константа `Large` задает недостижимую верхнюю границу для значения целевой функции, пропускных способностей и стоимостей дуг. Тип `PowerOfX` будет использован для определения пользователем функции, возвращающей мощность заданной вершины сети. Тип `XOfArc` будет использован для определения пользователем функций на дуге: стоимости, пропускной способности или потока. Значения определенных выше функций D, B, L, P и Pos на множестве N вершин задаются как поля структуры типа `TNode`, а функции C и K на множестве A дуг сети как поля записи типа `TArc`. Поле `A` структуры `TNode` определяет инцидентную дугу на пути от соответствующей вершины к корню, а поле `Y` – поток по ней. Наконец, поле `Im` содержит порядковый номер вершины.

```
#define  Large 0xFFFFFFFL
typedef long (*PowerOfX)(unsigned);
typedef long (*XOfArc)(unsigned, unsigned);
struct TArc {long C;            // price
             long K;            // capacity
             };
struct TNode { TNode *D /* direct next */, *B /* back next*/,
               TNode *L /* last */, *P /* predecessor */;
               TArc  *A;             // incidence basic arc (root path)
               bool Pos;          // is the basic arc direction to the root?
               long *U /* potential */, *Y /* incidence basic arc flow */
               int unsigned Im;   // image (index)
              };
```

Рис. 2. Структуры для представления базисного дерева

На рис. 3 приведено описание интерфейса класса `Transport`. Описание полей и методов класса `Transport` из раздела `Protected` будет дано далее при описании их реализации. Здесь остановимся на описании полей из раздела `Public` данного класса. Поля `NPivots` и `NChecks` содержат соответственно количество выполненных ведущих преобразований и количество проверок условий оптимальности при решении задачи. Поле `Ir` содержит состояние созданного объекта: `Ir`=0 – оптимальное решение найдено, `Ir`=1 – стоимость потока неограничена сверху, `Ir`=2 – не существует допустимого потока. В случае `Ir`=0 поле

FlowCost содержит стоимость оптимального потока, в противном случае поле BottleNeck содержит одну из дуг, параметры которой являются причиной отсутствия решения задачи.

Как отмечалось выше, класс Transport является абстрактным классом, предоставляющем средства решения и постоптимизапционного анализа задач транспортного типа как в матричной, так и в сетевой постановках. На рис. 4 приведено описание интерфейса классов Transshipment (4(a)) и Transportation (4(b)), предоставляющих средства решения транспортных задач в сетевой постановке и матричной постановках соответственно. Данные классы наследуют все поля и методы класса Transport, добавляют собственные и модифицируют конструктор и деструктор.

В разделе public класса Transshipment поле N определяет число вершин сети, функции PriceOfArc, CapacityOfArc и PowerOfSource составляются пользователем. Они должны возвращать конструктору значения C(Tail,Head), K(Tail,Head) и Q(Node) исходных данных решаемой задачи (1) – (3) для вершин и дуг, передаваемых им в качестве параметров. Здесь и далее значения переменных Tail и Head – вершины, соответствующие началу и концу выбранной дуги. Функции ArcFlow и NodePotential возвращают оптимальный поток по соответствующей дуге и потенциал соответствующей вершины. Наконец, с помощью метода Perturb модифицируются мощности вершин V1, V2, передаваемых в качестве параметров, и параметры инцидентных им дуг, а также отыскивается оптимальное решение возмущенной таким образом задачи. Наконец, с помощью метода Perturb модифицируются мощности вершин V1, V2, передаваемых в качестве параметров, и параметры инцидентных им дуг, а также отыскивается оптимальное решение возмущенной таким образом задачи. В Классе Transportation поля NSup, NCons определяют число поставщиков и потребителей соответственно. Остальные функции аналогичны соответствующим функциям класса Transshipment, но учитывают двудольность сети.

На рис. 5 приведены примеры использования классов Transshipment (a) и Transportation (b) в программах пользователя. В приведенном примере объект Prb класса Transhipment имеет восемь вершин, все исходные данные задаются с помощью датчика случайных чисел. После создания объекта его состояние отображается с помощью функции void ArcsPrint(Transportation), написанной пользователем. После этого решается возмущенная задача, в которой изменены мощности вершин 1 и 2 на величину потока между ними в оптимальном базисном решении. Объект Prb класса Transportation имеет по восемь поставщиков и потребителей, все исходные данные задаются алгоритмически. После создания объекта его состояние отображается с помощью функции void ArcsPrint(Transportation), написанной пользователем. После этого решается возмущенная задача, в которой могут быть изменены мощности вершин 1 и 2, а также стоимости и пропускные способности инцидентной им дуги.

4. Описание реализации классов

4.1. Конструкторы и деструкторы объектов

Функциональное назначение конструктора – построение оптимального базисного решения соответствующей задачи по импортируемым исходным данным. Впоследствии построенный объект, т.е. оптимальное базисное решение, может подвергаться перестройке и постоптимизационному анализу. Методы класса transport дают средства построения оптимального базисного решения из любого допустимого базисного решения.

Дерево, соответствующее начальному базисному решению, а также функции, определенные на множестве его вершин, показаны на рис. 6. В идейном отношении структура начального базисного решения для классов Transshipment (рис. 6(a)) и

```
class Transport {                  // Base class
   public:
      Transport(){Tree=0; PArc=0; FictArc.K=Large; FictArc.C=-Large;};
      ~Transport(){};
      long FlowCost;
      long unsigned NPivots, NChecks;
      int unsigned Ir, BottleNeckTail, BottleNeckHead;
   protected:
      long FIncr,                  // Flow incriment
           CIncr;                  // Potential incriment
      TNode *J,                    // Join point
            *Head,*Tail,           // Nodes of Incoming arc
            *Out,*POut,            // Nodes of Outgoing arc
            *Root, *Tree;
      TArc *InArc,                 // Incoming arc
            **PArc, FictArc;
      void Pivot(), Decompose(TNode*), Compose(TNode*,TNode*);
      TNode* LookForOutgoingArc(bool,TNode*);
      void Reroot(TNode*), OrderStudiedTree(TNode*,TNode*);
      void ModifyOfPotentials(TNode*);
};    // End of Transport
```

Рис. 3. Интерфейс класса `Transport`

```
class Transshipment: public Transport {
   public:
    Transshipment(
            unsigned, // Nnodes
            PowerOfX, // PowerOfSource
            XOfArc,   // CapacityOfArc
            XOfArc    // PriceOfArc
            );
      ~Transshipment();
      long ArcFlow(unsigned,unsigned);
      long NodePotential(unsigned);
      void Perturb(unsigned,unsigned,long);
      unsigned N;
      PowerOfX PowerOfSource;
      XOfArc CapacityOfArc, PriceOfArc;
   protected:
      bool ThereIsDefectArc();
      void Solve();
};     // End of Transshipment
```

(a)

```
class Transportation: public Transport {
public:
    Transportation( unsigned, unsigned,
                    PowerOfX, PowerOfX,
                    XOfArc, XOfArc
                    );
   ~Transportation();
   long ArcFlow(unsigned,unsigned);
   long SupPotential(unsigned);
   long ConsPotential(unsigned);
   void Perturb(unsigned,unsigned);
   PowerOfX PowerOfSource, PowerOfSewer;
   XOfArc CapacityOfArc,PriceOfArc;
   unsigned NSup, NCons;
protected:
   bool ThereIsDefectArc();
   void Solve();
   TArc *PSup, *PCons;
};  // End of Transportation
```

(b)

Рис. 4. Интерфейс классов `Transshipment` (a) и `Transportation` (b)

```
long random( long D ) {
return(D*((double)rand()/(double)RAND_MAX)
);
long PowerOfSource(unsigned Node){
 return(random(0));
};
long CapacityOfArc( unsigned Tail,
                    unsigned Head){
return(1+random(5)); };
long PriceOfArc(unsigned Tail,
                unsigned Head){
return(-5+random(10)); };
void ArcsPrint(Transshipment*P){
for(unsigned I=0; I<(P->N); I++){
     cout<<'\n'<<I<<'\t';
     for(unsigned J=0; J<(P->N); J++)
        cout<<setw(4)<<P->ArcFlow(I,J);
        };
     };
};
int _tmain(void){ srand(666);
int N=8; Transshipment*Prb=new
Transshipment(
                        N,
                        PowerOfSource,
                        CapacityOfArc,
                        PriceOfArc
                        );
   ArcsPrint(Prb);
   Prb->Perturb(1,2,0);
   cout<<"\n";
   ArcsPrint(Prb);
   delete Prb;
return 0; };
```

(a)

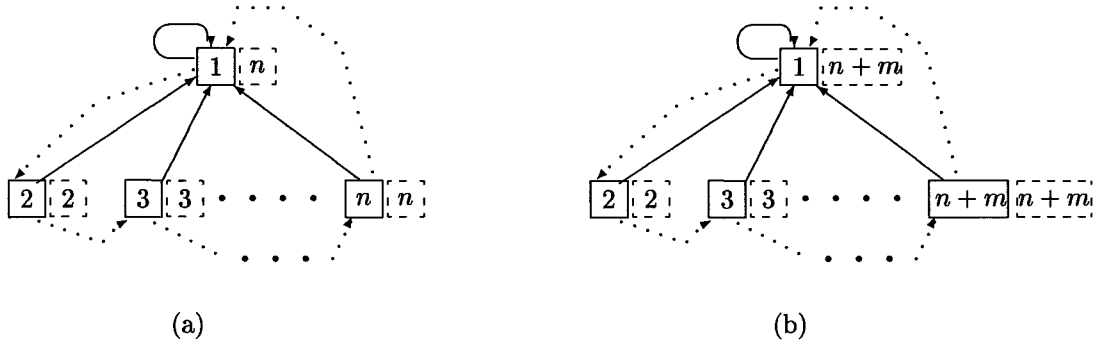
```
long PowerOfSource(unsigned Node){
return(1+Node); }; long
PowerOfSewer(unsigned Node){
return(1+Node); }; long
CapacityOfArc(unsigned Tail,
                    unsigned Head){
    int d=(long)Tail-(long)Head;
    d=(d>0)? d:-d; return(3+d);
}; long PriceOfArc(unsigned Tail,
                unsigned Head){
    int d=(long)Tail-(long)Head;
    d=(d>0)? d:-d; return(d);
}; void ArcsPrint(Transportation*P){
 for(unsignedI=0;I<(P->NSup);I++){
    cout<<'\n'<<I<<'\t';
    for(unsigned J=0; J<(P->NCons); J++)
        cout<<setw(4)<<P->ArcFlow(I,J);
    };
}; int _tmain(){ srand(666);
int NSup=8, NCons=8; Transportation
*Prb=new Transportation(
            NSup,NCons,
            PowerOfSource, PowerOfSewer,
            CapacityOfArc,
            PriceOfArc
            );
ArcsPrint(Prb);
if(Prb->Ir!=0)cout<<"Ir="<<Prb->Ir
    <<" Tail="<<Prb->BottleNeckTail
    <<"  Head="<<Prb->BottleNeckHead;
Prb->Perturb(1,2); cout<<"\n";
ArcsPrint(Prb); delete Prb; };
```

(b)

Рис. 5. Примеры использования классов `Transshipment` (a) и `Transportation` (b)

`Transportation` (рис. 6(b)) одинакова. Все дуги начального базисного решения являются: 1) инцидентными вершине 1, представляющей корень базисного дерева; 2) искусственными с неограниченными стоимостями и пропускными способностями. В соответствии с принятыми соглашениями величина константы `Large` равна верхней недостижимой границе величин $c(i,j), k(i,j)$, $(i,j) \in A$ и стоимости любого допустимого решения, поэтому при определении потенциалов вершин сети стоимости дефектных дуг полагаются равными величине `Large`. Поток по дугам начального базисного решения и их ориентация определяются с помощью метода `PowerOfSource`. Прямой порядок обхода вершин совпадает с их естественной нумерацией. Завершается создание объектов применением метода `Solve` для нахождения оптимального базисного решения.

Конструктор и деструктор объектов класса `Transshipment` приведены на рис. 7. Функция `Transshipment` имеет четыре параметра. Первый параметр – число вершин сети. Оставшиеся три параметра – функции `PriceOfArc`, `CapacityOfArc` и `PowerOfSource`, которые конструктор использует для получения значений C(`Tail,Head`), K(`Tail,Head`), и Q(`Node`) исходных данных решаемой задачи.



(a) (b)

Рис. 6. Дерево начального базисного решения для класса `Transshipment` (a) и для класса `Transportation` (b)

Конструктор и деструктор объектов класса `Transportation` приведены на рис. 8. Функция `Transportation` имеет шесть параметров. Первыей два параметра – число вершин в каждой доле грфа (т.е. число поставщиков и число потребителей). Оставшиеся четыре параметра – функции `Price`, `Cap`, которые конструктор использует для получения значений C(`Tail,Head`), K(`Tail,Head`), и функции `Source`, `Sewer`, используемые для передачи в программу значений мощностей вершин.

При выполнении обоих конструкторов сначала выделяется память под массив `Arc` и с помощью функций `Cap` и `Price` определяет недиагональные элементы данного массива. Затем выделяется память под массив `Tree` и строится начальное базисное решение. При выполнении деструктора освобождается память, выделенная объекту при его создании.

4.2. Метод `Solve`

Метод `Solve` реализован в виде одноименной функции (см. рис. 9). Функция `Solve` использует метод `ThereIsDefectArc` для выявления в текущем базисном решении дуг, которые не удовлетворяют условию оптимальности (4) – (6), и метод `Pivot` для выполнения ведущего преобразования. Применение данных методов производится до тех пор, пока метод `ThereIsDefectArc` находит дефектные дуги (т.е. дуги, не удовлетворяющие условию оптимальности) или метод `Pivot` не установит `Ir=1`, что соответствует существованию цикла из дуг с неограниченной пропускной способностью и положительной суммой их стоимостей. В последнем случае оптимальное решение прямой задачи неограничено, а двойственная задача не имеет решения.

Если же методу `ThereIsDefectArc` не удалось обнаружить дефектных дуг, то, как далее будет показано, в построенном решении все дуги удовлетворяют условиям оптимальности. В данном случае осталось проверить, имеются ли среди базисных дуг искусственные дуги, использованные при построении начального базисного решения. При положительном ответе мы имеем задачу, не имеющую допустимых решений. Оптимальное решение соответствующей двойственной задачи в этом случае неограничено.

В противном случае построены оптимальные решения прямой и двойственной задач. Поскольку двойственная задача имеет множество оптимальных решений, то в рассматриваемой процедуре оно приводится к нормальной форме, в которой минимальное значение двойственных переменных равно нулю.

Последняя группа вложенных циклов предназначена для подсчета оптимального значения целевой функции.

4.3. Изучение базисного дерева

Будем через $T(i)$ обозначать минимальное поддерево текущего базисного дерева T, стягивающего всех преемников вершины i. Вершину i будем называть корнем

```
//Constructor
Transshipment::Transshipment(
    unsigned NCard, PowerOfX Power,
    XOfArc Cap, XOfArc Price):Transport(){
PowerOfSource=Power; CapacityOfArc=Cap;
PriceOfArc=Price; N=NCard;
PArc=new TArc* [N];
for(unsigned i=0; i<N; i++){
   PArc[i]=new TArc [N];
   for(unsigned  m=0; m<i; m++){
      PArc[i][m].K=CapacityOfArc(i,m);
      PArc[i][m].C=PriceOfArc(i,m);
      };
   for(unsigned m=i+1; m<N; m++){
      PArc[i][m].K=CapacityOfArc(i,m);
      PArc[i][m].C=PriceOfArc(i,m);
      };
    };
Tree=new TNode [N]; Root=Tree;
for(unsigned  i=0; i<N; i++){
   long K=PowerOfSource(i);
   if(K>=0){  // (i,0) is the basic arc
     Tree[i].Pos=true;
     if(PArc[i][0].K>=K){
                // (i,0) is the real arc
        PArc[i][i].K=K;
        PArc[i][i].C=-PArc[i][0].C;
        Tree[i].A=PArc[i];
        }
        else { // (i,0) is the fiction arc
            PArc[i][i].K=K-PArc[i][0].K;
            PArc[i][0].K=-PArc[i][0].K;
            PArc[i][i].C=Large;
            Tree[i].A=&FictArc;
            };
         }else { // (0,i) is the basic arc
        Tree[i].Pos=false;
        if(PArc[0][i].K>=-K){
                   //(0,i) is a real arc
            PArc[i][i].K=K;
            PArc[i][i].C=PArc[0][i].C;
            Tree[i].A=PArc[0]+i;
            }
        else { // (0,i) is the fiction arc
            PArc[i][i].K=K+PArc[0][i].K;
            PArc[0][i].K=-PArc[0][i].K;
            PArc[i][i].C=-Large;
            Tree[i].A=&FictArc;
            }
      }
}
TNode* T=Tree+1;
for(unsigned  i=1; i<N; i++, T++) {
   T->P=Root; T->L=T;
   T->D=T+1; T->B=T-1;
   T->U=&(PArc[i][i].C);
   T->Y=&(PArc[i][i].K);
   PArc[0][0].K+=PArc[i][i].K;
   T->Im=i;
   }
Tree->U=&(PArc[0][0].C);
Tree->Y=&(PArc[0][0].K);
*(Tree->U)=0; Tree[N-1].D=Root;
Tree->Im=0; Tree->P=Root;
Tree->D=Root+1; Tree->B=Tree+(N-1);
Tree->L=Tree+(N-1); Tree->Pos=true;
J=Tree->L; Solve();
}; //End of Transshipment



//Destructor
Transshipment::~Transshipment(){
if(Tree) delete[] Tree ;
for(unsigned  i=0; i<N; i++)
    if(PArc[i]) delete[] PArc [i];
if(PArc) delete[] PArc;
}; //End of ~Transshipment
```

Рис. 7. Конструктор и деструктор класса `Transshipment`

поддерева $T(i)$. Поддерево $T(i)$ будем называть *изученным*, если установлено выполнение условий оптимальности для всех дуг, инцидентных вершинам данного поддерева. Если базисное дерево содержит изученные поддеревья, то проверка условий оптимальности необходима только для тех дуг, которые не стягивают вершины изученного поддерева. Попытаемся за счет надлежащего выбора порядка изучения поддеревьев базисного дерева и процедур преобразования информации при его перестройке уменьшить количество проверок условия оптимальности. Естественным требованием к искомому порядку является его согласованность с отношением включения: из $T(x) \subset T(y)$ должно следовать, что изучение поддерева $T(x)$ предшествует изучению $T(y)$.

```
Transportation::Transportation(
            unsigned N,
            unsigned M,
            PowerOfX Source,
            PowerOfX Sewer,
            XOfArc Cap,
            XOfArc Price
            ) : Transport(){
PowerOfSource=Source; PowerOfSewer=Sewer;
CapacityOfArc=Cap; PriceOfArc=Price;
NSup=N; NCons=M; PArc=new TArc* [N];
for(unsigned i=0; i<N; i++){
   PArc[i]=new TArc [M];
   for(unsigned  m=0; m<M; m++){
      PArc[i][m].K=CapacityOfArc(i,m);
      PArc[i][m].C=PriceOfArc(i,m);
      };
    };
PSup=new TArc [N]; PSup->K=PowerOfSource(0);
PSup->C=0; PCons=new TArc [M];
Tree=new TNode [N+M]; Root=Tree;
for(unsigned i=1; i<N; i++){
// All initial arc from Suppers are fiction
    TNode* T=Tree+i;
    (PSup->K)+=(PSup[i].K=PowerOfSource(i));
    T->Pos=true;
    PSup[i].C=Large;
    T->A=&FictArc;
    T->P=Tree; T->L=T; T->D=T+1; T->B=T-1;
    T->U=&(PSup[i].C); T->Y=&(PSup[i].K);
    T->Im=i;
    };
for(unsigned i=0; i<M; i++){
      long K=PowerOfSewer(i);
        (PSup->K)-=K;
      TNode* T=Tree+N+i;
      T->Pos=false;
      if(PArc[0][i].K>=K){
            //(0,i) is the real arc
        PCons[i].K=-K;
        PCons[i].C=PArc[0][i].C;
        T->A=PArc[0]+i;
        }
     else {
         // (0,i) is the fiction arc
        PCons[i].K=-K+PArc[0][i].K;
        PArc[0][i].K=-PArc[0][i].K;
        PCons[i].C=Large;
        T->A=&FictArc;
        }
      T->P=Tree; T->L=T; T->D=T+1;
      T->B=T-1; T->U=&(PCons[i].C);
      T->Y=&(PCons[i].K); T->Im=i;
      };
Tree->Pos=true; Tree->Im=0;
Tree->P=Tree; Tree->D=Tree+1;
(Tree+1)->B=Tree;
Tree->B=Tree->L=Tree+N+M-1;
(Tree+N+M-1)->D=Tree;
Tree->U=&(PSup->C);
Tree->Y=&(PSup->K);
PCons->C=0; Tree->A=0;
J=Tree->L; Solve();
}; //End of Transportation



//Destructor
Transportation::~Transportation(){
if(Tree) delete[] Tree;
if(PCons) delete  PCons;
if(PSup) delete[] PSup;
for(unsigned  i=0; i<NSup; i++)
    if(PArc[i]) delete[] PArc [i];
if(PArc) delete[] PArc;
}; //End of ~Transsportation
```

Рис. 8. Конструктор и деструктор класса `Transportation`

Рассмотрим концевой порядок обхода базисного дерева, определяемый функцией B. Данная функция индуцирует отношение $\prec$ линейного порядка на множестве всех поддеревьев базисного дерева: $T(i) \prec T(j)$, если $i = B^{s_i}(r)$, $j = B^{s_j}(r)$, $s_i < s_j$, где B^s – суперпозиция s функций B. Очевидно, что из $T(x) \subset T(y)$ следует $T(x) \prec T(y)$. Изучение поддеревьев базисного дерева в порядке, определяемом отношением $\prec$, использовано в методе `ThereIsDefectArc`. Данный метод реализован в виде одноименной булевой функции (рис. 10) без формальных параметров.

Данная функция возвращает значение `False` при отсутствии в базисном дереве дефектных дуг. В противном случае функция возвращает значение `True` и модифицирует значения следующих глобальных переменных: 1) `J` для указания корня поддерева, при изучении которого была найдена дефектная дуга; 2) `Tail` и `Head` для указания начала и

```
void Transshipment::Solve(){
NPivots=0; NChecks=0; Ir=0;
while ((Ir!=1) &&
    ThereIsDefectArc())Pivot();
if(Ir==1)return;
long MinU=*(Root->U);
TNode*V=Root->D;
do{
   TNode* PV=V->P;
   CIncr=*(V->U)-*(PV->U);
   if(CIncr>=Large){
         BottleNeckTail=PV->Im;
         BottleNeckHead=V->Im;
         Ir=2; return;
        };
   if(CIncr<=-Large){
         BottleNeckTail=V->Im;
         BottleNeckHead=PV->Im;
         Ir=2; return;
         };
   if(MinU>*(V->U))MinU=*(V->U);
   V=V->D;
}while(V!=Root); // End of do V=Root;
do{
    (*(V->U))-=MinU;
    V=V->D;
    } while(V!=Root);
long S=0; TNode* U=Root;
do {
  V=Root;
  do {
     S+=ArcFlow(U->Im,V->Im)*
           PArc[U->Im][V->Im].C;
     V=V->D;
     }while(V!=Root);
   U=U->D;
  }while(U!=Root);
FlowCost=S;
};   // End of Transshipment::Solve
```

(a)

```
void Transportation::Solve(){
NPivots=0; NChecks=0; Ir=0;
while ((Ir!=1) &&
    ThereIsDefectArc()) Pivot();
if(Ir==1)return;
long MinU=*(Root->U);
TNode* V=Root->D;
do{
   TNode* PV=V->P;
   CIncr=*(V->U)-*(PV->U);
   if(CIncr>=Large){
         BottleNeckTail=PV->Im;
         BottleNeckHead=V->Im;
         Ir=2; return;
        };
   if(CIncr<=-Large){
         BottleNeckTail=V->Im;
         BottleNeckHead=PV->Im;
         Ir=2; return;
         };
   if(MinU>*(V->U))MinU=*(V->U);
   V=V->D;
}while(V!=Root); // End of do V=Root;
do{ (
    *(V->U))-=MinU;
    V=V->D;
    } while(V!=Root);
long S=0;
for(unsigned i=0; i<NSup; i++)
    for(unsigned j=0; j<NCons; j++)
    S+=ArcFlow(i,j)*PArc[i][j].C;
FlowCost=S;
};   // End of Transportation::Solve
```

(b)

Рис. 9. Метод `Solve` для классов `Transshipment` (a) и `Transportation` (b)

конца дефектной дуги, которую следует ввести в базис; 3) `Cincr` для указания невязки в ограничении двойственной задачи, соответствующем дуге (`Tail, Head`).

До начала действия метода и после каждого выполнения тела внешнего цикла значение переменной J указывает корень последнего изученного дерева. По определению, все вырожденные поддеревья (т.е. представляющие единственную вершину) являются изученными. В первом вложенном цикле значение переменной J устанавливается равным корню следующего поддерева, которое следует изучить, а у переменной I – корню изученного поддерева, предшествующего $T(\mathtt{J})$.

```
bool Transshipment::ThereIsDefectArc(){
TNode *I, *I1, *I2, *LI, *LJ, *DLJ;
CIncr=0;
do {
   do {I=J; J=J->B;}while(J!=I->P);
   LJ=J->L; DLJ=LJ->D; LI=I->L; I2=J;
   do {
      do { I1=I;
         do {              // UINCR
            TArc *Arc1=PArc[I2->Im]+(I1->Im);
            TArc *Arc2=PArc[I1->Im]+(I2->Im);
            long RArc1=0; long RArc2=0;
            NChecks++;
            long Incr=(*(I1->U))-(*(I2->U));
            if((Arc1->K)<0)
                RArc1=-Incr+(Arc1->C);
            else if((Arc1->K)>0)
                RArc1=Incr-(Arc1->C);
            if(Arc2->K<0)RArc2=Incr+(Arc2->C);
                else if(Arc2->K>0)
                    RArc2=-Incr-(Arc2->C);
            if(RArc1<=RArc2) {
                if(RArc1<CIncr){
                    InArc=Arc1; CIncr=RArc1;
                    Head=I1; Tail=I2;
                    };
               }
                else if(RArc2<CIncr){
                    InArc=Arc2; CIncr=RArc2;
                    Head=I2; Tail=I1;
                    };      // End of UINCR
            if(CIncr<0)return true;
            I1=I1->D;
            }while(I1!=DLJ);
            I2=I2->D;
        }while(I2!=I);
         I=LI->D; LI=I->L;
    }while(I!=DLJ);
}while(J!=Root); return false;
};//End of Transshipment::ThereIsDefectArc
```

(a)

```
bool Transportation::ThereIsDefectArc(){
CIncr=0;
do { TNode* I;
   do {I=J; J=J->B;}while(J!=I->P);
   TNode* LJ=J->L; TNode* DLJ=LJ->D;
   TNode* LI=I->L; TNode* I2=J;
   do {
      do {
         bool I2Pos=I2->Pos;
         TNode* I1=I;
         do {              // UINCR
            bool I1Pos=I1->Pos;
            TNode *H, *T;
            if(I2Pos!=I1Pos){
                if(I2Pos){
                    H=I1;
                    T=I2;
                    }else{H=I2; T=I1;};
             TArc* Arc=PArc[T->Im]+(H->Im);
             long Incr=(*(T->U))-(*(H->U));
             long RArc=0; NChecks++;
                if((Arc->K)<0)
                    RArc=Incr-(Arc->C);
                else if((Arc->K)>0)
                    RArc=-Incr+(Arc->C);
                if(RArc<CIncr){
                    InArc=Arc;
                    CIncr=RArc;
                    Head=H; Tail=T;
                    };
              }      // End of UINCR
            if(CIncr<0)return true;
            I1=I1->D;
            }while(I1!=DLJ);
            I2=I2->D;
        }while(I2!=I);
         I=LI->D; LI=I->L;
    }while(I!=DLJ);
}while(J!=Root); return false;
};//End of Transshipment::ThereIsDefectArc
```

(b)

Рис. 10. Метод `ThereIsDefectArc` для классов `Transshipment` (a) и `Transportation` (b)

Следующая далее группа вложенных циклов обеспечивает проверку выполнения условия оптимальности для всех дуг, стягивающих в изучаемом дереве $T(\mathtt{J})$ вершины `I1` и `I2`, принадлежащие разным максимальным по включению изученным поддеревьям, т.е. из просмотра исключаются дуги, стягивающие вершины каждого изученного поддерева. Внутренний блок операторов `Uincr` вычисляет невязки `RArc1` `RArc2` дуг инцидентных вершинам `I1` и `I2` и модифицирует значения глобальных переменных `Tail` и `Head`, определяющих начало и конец дефектной дуги с наибольшей невязкой, а также переменной `Cincr`, определяющей величину этой невязки.

Если при изучении поддерева встречались дефектные дуги, то метод завершается,

возвращая значение `True`. В противном случае метод переходит к изучению следующего неизученного поддерева. Если изученным оказалось все базисное дерево, то метод завершается, возвращая значение `False`.

4.4. Ведущее преобразование

Если найдена дуга для ввода в базис, то процесс изучения текущего базисного дерева останавливается и вызывается метод `Pivot` выполнения ведущего преобразования. Данный метод использует модифицированную перестройку функций D, B и L на множестве вершин базисного дерева [17, 22, 24], при которой наследуется предшествование всех не изменившихся изученных поддеревьев относительно неизученных. На рис. 11 приведен текст метода `Pivot`. Для определения выводимой дуги величина `FIncr` предварительно устанавливается равной пропускной способности вводимой дуги. Далее определяются вершины `InV1` и `InV2`, такие, что: 1) вдоль пути в базисном дереве от вершины `InV1` до вершины `J` поток должен возрасти на прямых дугах и уменьшиться на встречных; 2) вдоль пути от вершины `InV2` до вершины `J` поток должен наоборот уменьшиться на прямых дугах и увеличиться на встречных. Для определения максимально возможного допустимого изменения потока и нахождения ближайшей к вершине `J` дуги, определяющей это изменение дважды, используется метод `LookForOutgoingArc`. Текст процедуры, реализующей данный метод, приведен на рис. 12(а). Входными данными являются переменная `V`, задающая вершину поддерева $T($`J`$)$, и логическая переменная `Decr`, определяющая характер изменения потока в дугах базисного дерева, принадлежащих пути от вершины `V` до вершины `J`: 1) если `Decr=True`, то поток уменьшается на прямых дугах и увеличивается на обратных; 2) если `Decr=False`, то поток увеличивается на прямых дугах и уменьшается на обратных. Выходным параметром является переменная `Out`, указывающая ближайшую к `J` вершину, такую, что дуга $($`Out`$, P($`Out`$))$ определяет максимально возможное допустимое изменение потока. Кроме того, метод уточняет значение глобальной переменной `Fincr`, указывающей максимально возможное допустимое изменение потока, установленное ранее.

Если максимально возможное допустимое изменение `FIncr` потока после просмотра всех дуг полученного цикла осталось неограниченным (т.е. по определению равным `Large`), то выполнение завершается с кодом `Ir=1`. В противном случае производится изменение потока вдоль дуг образованного цикла. Если же в образованном цикле величина `FIncr` определяется только пропускной способностью введенной на данной итерации дуги, то эту дугу вновь следует вывести из базиса, т.е. завершить выполнение метода `Pivot` и продолжить изучение базисного дерева $T($`J`$)$. Признаком отсутствия кандидатов для вывода из базиса является равенств `Out1=0` и tt Out2=0. В противном случае введенная дуга остается в базисе. Последняя из дуг, при просмотре которой было изменение величины `Out1` или `Out2`, выбирается для вывода из базиса. Для идентификации данной дуги используется переменная `Out`, указывающая инцидентную ей вершину, дальнюю от вершины `J`. Затем переменным `InV1` и `InV2` присваиваются значения, равные вершинам, инцидентным вводимой в базис дуге, так чтобы `InV1` $\in T \setminus T($`Out`$)$, `InV2` $\in T($`Out`$)$.

Базисное дерево при удалении из него выводимой дуги преобразуется в лес из двух компонент связности $T($`Out`$)$ и $T \setminus T($`Out`$)$. Очевидно, что условия выполнения критерия оптимальности изменяются только для дуг, инцидентных вершинам из разных компонент связности. Так как вводимая в базис дуга была определена методом `ThereIsDefectArc`, то для нового базисного решения будут справедливы утверждения: 1) компонента связности, не содержащая вершину `J`, представляющую корень изучаемого поддерева, является изученным поддеревом; 2) в компоненте связности, содержащей вершину `J`, изученными останутся все ранее изученные поддеревья, не содержащие вершины `InV1`.

```
void Transport::Pivot(){
TNode *InV1, *InV2;
Ir=0; NPivots+=1;
 // IncomingArc
long KInArc=InArc->K;
if (KInArc>0){
        FIncr=KInArc;
        InV1=Head;
        InV2=Tail;
        } else {
            FIncr=-KInArc;
            InV2=Head;
            InV1=Tail;
            };
TNode* Out1=LookForOutgoingArc(false,InV1);
TNode* Out2=LookForOutgoingArc(true,InV2);
  //ModifyFlow
if(FIncr>=Large) Ir=1;
   else {
        TNode* V=InV1;
        while(V!=J){
            (*(V->Y))+=FIncr;
            V=V->P;
            };
        V=InV2;
        while(V!=J){
            (*(V->Y))-=FIncr;
            V=V->P;
            };
        if((Out1==0)&&(Out2==0)) {
            (InArc->K)=-KInArc;
            Ir=2;
            }else{
                Ir=0;
                (InArc->K)=(KInArc<0)?
                    -KInArc : KInArc;
                };
}; // End of if(FIncr>=Large)
switch(Ir){
 case 0: {
  //OutgoingArc
  TNode* Out=(Out2==0)? Out1 : Out2;
  if(( *(Out->Y)!=0)&&((Out->A)!=&FictArc))
                        Out->A->K=-(Out->A->K);
  if(Out==Out1){
      Out1=InV1;
      InV1=InV2;
      InV2=Out1;
      };
  Decompose(Out);
  OrderStudiedTree(InV1,J);
  Reroot(InV2);
  Compose(InV1,InV2);
  InV2->A=InArc; (InV2->Pos)=(InV2==Tail);
  if(KInArc<0)FIncr=-FIncr-KInArc;
  *(InV2->Y)=(InV2->Pos)? FIncr : -FIncr;
  if((KInArc<0)==(InV2->Pos))CIncr=-CIncr;
  ModifyOfPotentials(InV2);
  J=InV2; break;
  };  // End of case 0
 case 1: {
  BottleNeckTail=Tail->Im;
  BottleNeckHead=Head->Im;
  return;
  };
 case 2: {Ir=0; J=J->D;};
  };
};  // End of Pivot
```

Рис. 11. Метод `Pivot`

Для определения на деревьях $T(\texttt{Out})$ и $T \setminus T(\texttt{Out})$ значений функций P, D, B и L, сохраняющих порядок предшествования ранее изученных поддеревьев неизученным, используется метод `Decompose`. Текст процедуры, реализующей данный метод, приведен на рис. 12(b). Единственный формальный параметр `Out` определяет дальнюю от корня вершину дуги, выводимой из базиса. Следующий шаг – построение упорядочения (за счет переопределения функций D, B и L), изученных поддеревья дерева $T(\texttt{J}) \setminus T(\texttt{Out})$, при котором все его изученные поддеревья, не содержащие вершины `InV1`, предшествуют поддеревьям, содержащим эту вершину. Данный шаг выполняется методом `OrderStudiedTree`. Текст процедуры, реализующей метод `OrderStudiedTree`, приведен на рис. 13(a). Формальными параметрами процедуры являются вершины `I` и `J` дерева, удовлетворяющие условию $T(\texttt{I}) \subset T(\texttt{J})$. Предполагается, что все поддеревья, предшествующие $T(\texttt{J})$, являются изученными. В результате применения метода

```
TNode* Transport::LookForOutgoingArc(
            bool Decr,
            TNode* V){
long Flow; TNode *SV, *Out=0; while(V!=J){
   SV=V; V=V->P;
   if(SV->Pos){
         Flow=*(SV->Y);
         if(!Decr)Flow=(SV->A->K)-Flow;
        }
    else {
         Flow=-(*(SV->Y));
         if(Decr)Flow=(SV->A->K)-Flow;
        };
   if(Flow<FIncr){FIncr=Flow; Out=SV;};
    }; // End of while(V!=J)
return Out; }; // End of LookForOutgoingArc
```

(a)

```
void Transport::Decompose(TNode* Out){
TNode* LV=Out->L;
TNode* BV=Out->B;
Out->B=LV;
TNode* V=Out->P;
Out->P=Out;
TNode* DLV=LV->D;
LV->D=Out;
BV->D=DLV; DLV->B=BV;
TNode* PDLV=DLV->P;
while(V!=PDLV) {V->L=BV; V=V->P;};
if(DLV==V) V->L=BV;
}; // End of Decompose
```

(b)

Рис. 12. Методы `LookForOutgoingArc` (a) и `Decompose` (b)

определяются значения функций D, B и L такие, что 1) сохраняется предшествование изученных поддеревьев неизученным; 2) все изученные поддеревья, не содержащие вершины `I`, предшествуют поддеревьям, содержащим вершину `I`.

Далее с помощью метода `Reroot` переопределяются функции P, D, B и L на дереве T(`Out`), так, чтобы вершина `InV2` стала корнем. На рис. 13(b) приведен текст процедуры, реализующей метод `Reroot`. Формальный параметр `NewRoot` определяет требуемое размещение корня, для которого определяются новые значения функций P, D, B и L.

Затем необходимо сделать композицию деревьев, добавив вводимую дугу в лес, образованный при удалении выводимой дуги. Измененные значения функций P, D, B и L должны сохранять предшествование изученных поддеревьев неизученным. При этом в перестроенном дереве последним изученным поддеревом будет T(`InV2`). Данный шаг осуществляется с помощью метода `Compose`.

Текст процедуры, реализующей метод `Compose`, приведен на рис. 14(a). Формальные параметры `V` и `W` процедуры определяют вершины, инцидентные вводимой дуге. При этом предполагается, что вершины `V` и `W` принадлежат разным деревьям, а вершина `W` является корнем дерева. В результате применения метода значения функций P, D, B и L модифицируются таким образом, что сохраняется порядок, имевший ранее место.

Завершается метод `Pivot` фиксацией ориентации потока во вводимой дуге и соответствующей модификацией двойственного решения, которая осуществляется методом `ModifyPotentials`. Текст соответствующей процедуры приведен на рис. 14(b). Формальным параметром процедуры является корень поддерева, к потенциалам вершин которого следует добавить величину глобальной переменной `CIncr`.

На рис. 1 дана иллюстрация выполнения ведущего преобразования. Текущее базисное дерево и определенные на нем функции P, D, B и L показаны на рис. 1(a). Пусть при изучении дерева T(`J`), `J`=2 метод `ThereIsDefectArc` определил, что в базис следует ввести дугу, инцидентную вершинам `I1`=9, `I2`=7. Пусть метод `Pivot` определил, что из базиса следует вывести дугу, инцидентную вершинам `Out`=5 и P(`Out`) = 4. На рис. 1(b) показан результат декомпозиции. На рис. 1(c) – результат переупорядочения изученных поддеревьев и переноса корня. Результат композиции показан на рис. 1(d).

```
void Transport::OrderStudiedTree(
            TNode* I,
            TNode* Join
            ){
if(I==Join)return;
TNode* V=I; TNode* LV=V->L;
TNode* DLV=LV->D;
TNode* PDLV=DLV->P;
do{
   TNode* PV=V->P; TNode* DPV=PV->D;
   if(DPV==V){
        if(PDLV!=PV) PV->L=LV;
        else if(PV!=DLV) {
            LV=PV->L; DLV=LV->D;
            PDLV=DLV->P;
            }
            else PV->L=LV;
        }
   else {
      TNode* BV=V->B; LV->D=DPV;
      DPV->B=LV; PV->D=V; V->B=PV;
      BV->D=DLV; DLV->B=BV;
      if(PDLV!=PV){
        PV->L=BV;
        LV=BV;
        }else if(PV!=DLV){
            LV=PV->L; DLV=LV->D;
            PDLV=DLV->P;
            } else {
                PV->L=BV;
                LV=BV;
                }
      }
   V=PV;
    }while(V!=Join);
while(V!=PDLV) {
    V->L=LV;
    V=V->P;
    };
if(V==DLV)V->L=LV;
}; //  End of OrderStudiedTree
```

(a)

```
void Transport::Reroot(
            TNode* NewRoot
            ){
long VY, PVY; TArc *VA, *PVA;
TNode *V,*W,*BV,*DV,*PV,*BPV,
                *LV,*DLV,*PDLV;
bool PosOrV, PosOrPV;
V=NewRoot;
PosOrV=V->Pos; VY=*(V->Y);
VA=V->A; PV=V->P; V->P=V;
if (V==PV) return;
LV=V->L; DLV=LV->D; PDLV=DLV->P;
BV=V->B; BPV=PV->B; DV=V;
if(V!=LV){
    DV=V->D; LV->D=V;
    V->B=LV;
    };
V->D=PV; PV->B=V;
while(true){
    PosOrPV=PV->Pos; PV->Pos=!PosOrV;
    PosOrV=PosOrPV;
    PVY=*(PV->Y);
    *(PV->Y)=-VY; VY=PVY;
    PVA=PV->A; PV->A=VA; VA=PVA;
    if(PV==PDLV){
      W=PDLV->L;
      if(LV==W){LV=BV; break;};
      LV=W; W=DLV; DLV=LV->D;
      PDLV=DLV->P;
      LV->D=DV; DV->B=LV; DV=W;
      if(PV==PDLV)break;
      };
   W=V; V=PV; PV=V->P;
   V->P=W; W=BV; BV=BPV;
   BPV=PV->B; PV->B=W; W->D=PV;
   };
BV->D=DV; DV->B=BV; PV->L=LV;
DLV=LV->D; PDLV=DLV->P; PV->P=V;
while(true){
    PV=PV->P;
   if(PV!=PDLV) PV->L=LV;
    else if(DLV==PDLV)break;
              else {
                LV=PDLV->L; DLV=LV->D;
                PDLV=DLV->P;
                 };
};
PV->L=LV;
};  //  End of Reroot
```

(b)

Рис. 13. Методы `OrderStudiedTree` (a) и `Reroot` (b)

```
void Transport::Compose(
                    TNode *V,
                    TNode *W){
W->P=V;
TNode* LW=W->B; W->B=V;
TNode* DV=V->D; V->D=W;
TNode* LV=V->L; LW->D=DV;
DV->B=LW;
if(V==LV) {
      TNode* PDV=DV->P;
      do{ V->L=LW; V=V->P; }while(V!=PDV);
      if(V==DV)V->L=LW;
        };
}; //  End of Compose
```

(a)

```
void Transport::ModifyOfPotentials(
                                    TNode *V){
TNode* I=V->L;
while(I!=V){
    (*(I->U))+=CIncr;
    I=I->B; };
    (*(V->U))+=CIncr;
}; //  End of ModifyOfPotentials
```

(b)

Рис. 14. Методы `Compose` (a) и `ModifyOfPotentials` (b)

4.5. Метод `Perturb`

Постоптимизационный анализ и многие алгоритмы для задач размещения и синтеза сетей требуют решения последовательности локально возмущенных задач, отличающихся от решенной значениями функций `PowerOfSource` на двух вершинах `V1`, `V2`, а также значениями функций `PriceOfArc` и `CapacityOfArc` на дугах, инцидентных указанным вершинам. В данном случае при построении начального базисного решения возмущенной задачи разумно использовать оптимальное базисное решение невозмущенной задачи. Функции на базисном дереве, следует модифицировать так, чтобы изученными остались все поддеревья, не содержащие одновременно вершины `V1`, `V2`. Для решения указанной задачи в классы `Transshipment` и `Transportation` включен метод `Perturb`.

Текст функции, реализующей данный метод для класса `Transshipment` приведен на рис. 15. Формальными параметрами процедуры являются вершины `V1`, `V2` и величина `FlowExch`. При этом предполагается, 1) значения функции `PowerOfSource` на вершине `V1` увеличенно на величину `FlowExch`, а на вершине `V2` – уменьшенно на эту величину; 2) модифицированные значения функций `PriceOfArc` и `CapacityOfArc` на дугах, инцидентных вершинам `V1`, `V2` должны быть определены пользователем.

Текст функции, реализующей метод `Transportation::Perturb` приведен на рис. 16. Формальными параметрами процедуры являются вершины `V1`, `V2`. При этом предполагается, что пользователем переопределены значения функций `PowerOfSource` на вершинах `V1` и `V2`, и значения функций `PriceOfArc` и `CapacityOfArc` на дугах, инцидентных вершинам `V1`, `V2`.

В обеих функциях базисное решение строится следующим образом. Корень оптимального базисного дерева переносится в вершину `V1`. Если оптимальное базисное дерево не содержит дуги инцидентной вершинам `V1`, `V2`, то с помощью локальной процедуры `CorrectBasicTree` строится базисное решение, близкое к оптимальному, но содержащее такую дугу. Для вывода из оптимального базиса выбирается дуга, определяющая минимальное из максимально возможных допустимых изменений потока (увеличение или уменьшение) вдоль пути от вершины `V2` до корня, т.е. вершины `V1`. Далее локальная процедура `CorrectData` запрашивает измененные значения стоимостей и пропускных способностей и заменяет базисную дугу, инцидентную вершинам `V1` и `V2`, искусственной дугой стоимостью `-Large`. В построенном базисном решении все собственные поддеревья

```
Transshipment::Perturb(unsigned IV1,
                       unsigned IV2,
                       long FlowExch
                       ){
TNode* V1=Tree+IV1;
TNode* V2=Tree+IV2;
Reroot(V1);
Root=V1; J=V1;
if(V1!=V2->P) { //  CorrectBasisTree
  TNode *V, *Out, *Out1, *Out2;
  FIncr=Large;
  Out1=LookForOutgoingArc(false,V2);
  Out2=LookForOutgoingArc(true,V2);
  V=V2;
  if(Out2!=0) Out=Out2;
     else { Out=Out1; FIncr=-FIncr;};
  while(V!=J){
    (*(V->Y))-=FIncr;
    V=V->P;
    };
  if(*(Out->Y)!=0)
    (Out->A->K)=-(Out->A->K);
  Decompose(Out);
  Reroot(V2);
  Compose(V1,V2);
  V2->Pos=(FIncr>0);
  *(V2->Y)=FIncr; V2->A=&FictArc;
   }; //  end of CorrectBasisTree

// CorrectData
long Flow=*(V2->Y)-FlowExch;
TArc* A=PArc[V1->Im]+(V2->Im);
if((A->K)<0)Flow+=(A->K);
A->K=CapacityOfArc(IV1,IV2);
A->C=PriceOfArc(IV1,IV2);
A=PArc[V2->Im]+(V1->Im);
if((A->K)<0)Flow-=(A->K);
A->K=CapacityOfArc(IV2,IV1);
A->C=PriceOfArc(IV2,IV1);
V2->Pos=(Flow>0);
CIncr=(V2->Pos)?
    Large-(*(V2->U)) : -Large-(*(V2->U));
(*(V2->Y))=Flow;
ModifyOfPotentials(V2);
OrderStudiedTree(V2,J);
J=V2;
Solve();

}; // End ofTransshipment::Perturb
```

Рис. 15. Метод `Transshipment::Perturb`

```
void Transportation::Perturb(
                    unsigned IV1,
                    unsigned IV2){
TNode* V1=Tree+IV1;
TNode* V2=Tree+NSup+IV2;
Reroot(V1); Root=V1;
J=V1;
if(V1!=V2->P) { //  CorrectBasisTree
    FIncr=Large;
    TNode* Out=LookForOutgoingArc(false,V2);
    TNode* V=V2;
    while(V!=J){ (*(V->Y))+=FIncr; V=V->P; };
    if(*(Out->Y)!=0)(Out->A->K)=-(Out->A->K);
    Decompose(Out);
    Reroot(V2);
    Compose(V1,V2);
    *(V2->Y)=-FIncr;
    V2->A=&FictArc;
  }; //  end of CorrectBasisTree

  // CorrectData
  TArc* A=PArc[IV1]+IV2;
  if((A->K)<0)(*(V2->Y))+=(A->K);
  A->K=CapacityOfArc(IV1,IV2);
  A->C=PriceOfArc(IV1,IV2);
  CIncr=-Large-(*(V2->U));
  ModifyOfPotentials(V2);
  OrderStudiedTree(V2,J);
  J=V2;
  Solve();
}; //  End of  Transsportation::Perturb
```

Рис. 16. Метод `Transportation::Perturb`

являются изученными, а неизученным является только само базисное дерево. Поэтому переменная J устанавливается равной V2, т.е. последнему собственному поддереву текущего базисного дерева.

Заключение

Рассмотреные процедуры ведущего преобразования и упорядоченного изучения базисного дерева в схеме симплекс-алгоритма для задач транспортного типа существенно сокращают число проверок условия оптимальности. Приведенный в работе исходный текст абстрактного класса `transport` и классов `Transshipment` и `Transportation`, предназначенных для решения и постоптимизационного анализа транспортных задач соответственно в сетевой и матричной постановках, дает эффективную технику программной реализации потоковых алгоритмов.

Литература

1. Dantzig, G. Application of the Simplex Method to a Transportation Problem / G. Dantzig // Activity Analysis of Production and Allocation. – 1951. – P. 196 – 218.
2. Glickman, S. Coding in Transportation Problem / S. Glickman, J. Jonson, L. Eselson // Naval Research Logistics Quar. – 1960. – V. 7, № 2. – P. 169.
3. Fulkerson, D. An Out-of-Kilter Method for Minimal-Cost Flow Problems / D. Fulkerson // SIAM J. of Applied Mathematics – 1961. – V. 9, № 1. – P. 1 – 18.
4. Форд, Л. Потоки в сетях / Л. Форд, Д. Фалкерсон. – М.: Мир, 1962. – 276 с.
5. Dantzig, G. Linear Programming and Extensions / G. Dantzig. – Princeton: University, 1963. – 215 p.
6. Гольштейн, Е.Г. Новые направления в линейном программировании / Е.Г. Гольштейн, Д.Б. Юдин. – М.: Сов. радио, 1966. – 524 с.
7. Jonson, J. Networks and Basic Solutions / J. Jonson // Operations. Res. – 1966. – V.14. – P. 619 – 623.
8. Glover, F. Implementation and Computational Comparisions of Primal, Dual and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems/ F. Glover, D. Karney, D. Klingman // Networks. – 1974. – V. 4, № 3. – P. 191 – 212.
9. A Computational Study on start procedures basis change criteria, and solution algorithms for transportation problems / F. Glover, D. Karney, D. Klingman, A. Napier // Manage. Sci. – 1974. – Vol.20, № 5.– P. 793 – 813.
10. Bradley, G.H. Design and Implemetation of Large Scale Primal Transshipment Algorithms / G.H. Bradley, G.G. Brown, G.W. Graves // Manage. Sci. – 1977. – Vol.24, № 1. – P. 1 – 34.
11. Barr, R. Enhancements of Spanning Tree Labeling Procedures for Network 0ptimization / R. Barr, F. Glover , D. Klingman // INFOR. – 1979. – V. 17, № 1. – P. 16 – 34.
12. Ahrens, J.H. Primal Transportation and Transshipment Algorithms / J.H. Ahrens, G. Finke // Z. Oper. Res. – 1980. – V.24, № 1. – P. 1 – 32.
13. Armstrong, R. D. Implementation and Analisis of a Variant of Dual Method for the Capacitated transshipment Problem / R.D. Armstrong, D. Klingman, D. Whitman // European J. Oper. Res. – 1980. – V.4, № 6. – P. 403 – 420.
14. Панюков, А.В. Алгоритм локальной оптимизации для задачи размещения прямоугольных объектов с минимальной длиной связывающей их сети / А.В. Панюков // Изв. АН СССР. Техн. кибернетика. – 1981. – № 6. – С. 180 – 184.

15. Панюков, А.В. Метод решения возмущенной транспортной задачи на сети / А.В. Панюков // Методы и программы решения оптимизационных задач на графах и сетях. Часть 2: Теория, алгоритмы: тез. докл. П Всес. совещания;. Улан-Удэ, август, 1982. – Новосибирск, 1982. – С. 113 – 114.

16. Гловер, Ф. Последние достижения в технике реализации сетевых потоковых алгоритмов / Ф. Гловер, Д. Клингман // Экономико-оптимизационные задачи большой размерности: труды сов.-американ. семинара. США, 1980. – М., 1983. – С. 180 – 209.

17. Панюков, А.В. Повышение эффективности прямых алгоритмов построения потока минимальной стоимости в насыщенной сети / А.В. Панюков // Системы программного обеспечения задач оптимального планирования: VШ Всес. симп: тез. докл. Нарва-Йыесуу, апрель, 1984. – М., 1984. – С. 153 – 154.

18. Йенсен, П. Потоковое программирование / П. Йенсен, Д. Барнес – М.: Радио и связь, 1984. – 391 с.

19. Galil, Z. An $O(n^2(m + n \log n) \log n)$ min-cost flow algorithm / Z. Galil, E. Tardos // 27th Annu. Symp. Found. Comput. Sci., Toronto, Oct. 27 – 29, 1986. – P. 1 – 9.

20. Goldberg, A.V. Combinatorial algorithms for the generalized circulation problem / A.V. Goldberg, , S.A. Plotkin, E. Tardos // Math. Oper. Res. – 1991. – Vol.16, № 2. – P. 351 – 381.

21. Orlin, J. B. Polynomial dual network simplex algorithms / J.B. Orlin, S.A. Plotkin , E. Tardos // Math. Program. – 1993. Vol. 60A, № 3. P. 255 – 276.

22. Panyukov, A.V. The Study of Basis Tree for Primal Transshipment Algorithms / A.V. Panyukov // CO94, Amsterdam, the Netherlands, April 5 – 8, 1994. Program&Abstracts.

23. Kleinschmidt, P. A Strongly Polynomial Algorithm for the Transportation Problem / P. Kleinschmidt, H. Schannath // Mathematical Programming. – 1995. – Vol. 68, № 1. – P. 1 – 13.

24. Панюков, А.В. Упорядоченное изучение базисного дерева в прямых алгоритмах для транспортной задачи / А.В. Панюков // Международная Сибирская конференция по исследованию операций: материалы конф. – Новосибирск, 1998. – С. 44.

25. Панюков, А.В. Задача размещения прямоугольных объектов с минимальной стоимостью связывающей сети / А.В. Панюков // Дискретный анализ и исследование операций. Серия 2. – Том 8, № 1. – 2001. – С. 70 – 87.

26. Панюков, А.В. Способ генерации должностных инструкций и положений о подразделениях // А.В. Панюков, В.А. Телегин // III Всероссийская конференция «Проблемы оптимизации и экономические приложения»: материалы конф. (Омск, 11 – 15 июля 2006 г.) / Омский филиал Ин-та математики им. С.Л. Соболева СО РАН. – Омск, 2006. – С. 185.

Кафедра экономико-математических методов и статистики,
Южно-Уральский государственный университет
pav@susu.ac.ru

Поступила в редакцию 20 июня 2008 г.