最小费用最大流

分类： 偶滴ACM 2009-09-16 14:57 2209人阅读 评论(0) 收藏 举报

算法 网络 path c

最近为了写过最小费用最大流的模板网上找了很多资料，现将其整理如下：（版权归原作者所有）

Dijkstra版：

【MCMF问题及数学模型】

在介绍最大流问题时，我们列举了一个最大物资输送流问题。如果这个问题的已知条件还包括每条边运送单位物资的费用，那么怎样运送，才能得到最大运输量，并且输送费用最少？这便是所谓最小费用最大流问题。

在最大流的有关定义的基础上，若每条有向边除权数c(e)（表示边容量）外还有另外一个权数w(e)（表示单位流所需费用），并且已求得该网络的最大流值为F， 那么最小费用最大流问题，显然可用以下线性规划模型加以描述：

Min ∑ w(e)f(e)

e∈E

满足 0≤f(e)≤c(e) ，对一切e∈E

f+(v)=f-(v) ，对一切v∈V

f+(x)=F （最大流约束）

（或f-(y)=F ）

【算法思路】

解决最小费用最大流问题，一般有两条途径。一条途径是先用最大流算法算出最大流，然后根据边费用，检查是否有可能在流量平衡的前提下通过调整边流量，使总费用得以减少？只要有这个可能，就进行这样的调整。调整后，得到一个新的最大流。

然后，在这个新流的基础上继续检查，调整。这样迭代下去，直至无调整可能，便得到最小费用最大流。这一思路的特点是保持问题的可行性（始终保持最大流），向最优推进。另一条解决途径和前面介绍的最大流算法思路相类似，一般首先给出零流作为初始流。这个流的费用为零，当然是最小费用的。然后寻找一条源点至汇点的增流链，但要求这条增流链必须是所有增流链中费用最小的一条。如果能找出增流链，则在增流链上增流，得出新流。将这个流做为初始流看待，继续寻找增流链增流。这样迭代下去，直至找不出增流链，这时的流即为最小费用最大流。这一算法思路的特点是保持解的最优性（每次得到的新流都是费用最小的流），而逐渐向可行解靠近（直至最大流时才是一个可行解）。

由于第二种算法和已介绍的最大流算法接近，且算法中寻找最小费用增流链，可以转化为一个寻求源点至汇点的最短路径问题，所以这里介绍这一算法。

在这一算法中，为了寻求最小费用的增流链，对每一当前流，需建立伴随这一网络流的增流网络。 按以下原则建 立增流网络的边：若G中边(u，v)流量未饱，即f(u,v) ＜ e(u,v)，则G ' 中建边(u,v)，赋权w ' (u,v)=w(u,v)；若G中边(u, v)已有流量，即f(u,v)〉0，则G′中建边(v,u)，赋权w′(v,u) =-w(u,v)。建立增流网络后，即可在此网络上求源点至汇点的最短路径，以此决定增流路径，然后在原网络上循此路径增流。这里，运用的仍然是最大流算法的增流原理，唯必须选定最小费用的增流链增流。

计算中有一个问题需要解决。这就是增流网络G ′中有负权边，因而不能直接应用标号法来寻找x至y的最短路径，采用其它计算有负权边的网络最短路径的方法来寻找x至y的最短路径，将 大大降低计算效率。为了仍然采用标号法计算最短路径，在每次建立增流网络求得最短路径后,可将网络G的权w(e)做一次修正，使再建的增流网

络不会出现负权边，并保证最短路径不至于因此而改变。下面介绍这种修改方法。

当流值为零，第一次建增流网络求最短路径时，因无负权边，当然可以采用标号法进行计算。为了使以后建立增流网络时不出现负权边，采取的办法是将 G中有流边（f(e)＞0）的权w(e)修正为0。为此， 每次在增流网络上求得最短路径后，以下式计算G中新的边权w " (u,v)：

w " (u,v)=L(u)-L(v)+w(u,v) (*)

式中 L(u),L(v) -- 计算G′的x至y最短路径时u和v的标号值。第一次求最短径时如果(u,v)是增流路径上的边，则据最短 路径算法一定有 L(v)=L(u)+w ' (u,v)=L(u)+w(u,v)， 代入(*)式必有

w″(u,v)=0。

如果(u,v)不是增流路径上的边，则一定有：

L(v)≤L(u)+w(u,v)，

代入(*)式则有 w(u,v)≥0。

可见第一次修正w(e)后，对任一边，皆有w(e)≥0， 且有流 的边（增流链上的边），一定有w(e)=0。以后每次迭代计算，若 f(u,v)＞0，增流网络需建立(v,u)边，边权数w ' (v,u)=-w(u,v) =0，即不会再出现负权边。

此外，每次迭代计算用(*)式修正一切w(e)， 不难证明对每一条x至y的路径而言，其路径长度都同样增加L(x)-L(y)。因此，x至y的最短路径不会因对w(e)的修正而发生变化。

【计算步骤】

1．对网络G=[V,E,C,W]，给出流值为零的初始流。

2．作伴随这个流的增流网络G′=[V′,E′,W′]。

G′的顶点同G：V′=V。

若G中f(u,v)＜c(u,v)，则G′中建边(u,v)，w(u,v)=w(u,v)。

若G中f(u,v)＞0，则G′中建边(v,u)，w′(v,u)=-w(u,v)。

3．若G′不存在x至y的路径，则G的流即为最小费用最大流，

停止计算；否则用标号法找出x至y的最短路径P。

4．根据P，在G上增流： 对P的每条边(u,v)，若G存在(u,v)，则(u,v)增流；若G存在(v,u)，则(v,u)减流。增（减）流后，应保证对任一边有c(e)≥ f(e)≥0。

5．根据计算最短路径时的各顶点的标号值L(v)，按下式修 改G一切边的权数w(e):

L(u)-L(v)+w(e)→w(e)。

6．将新流视为初始流，转２。

最小费用最大流 修改的dijkstra + Ford-Fulksonff算法

修改的dijkstra其实和Johnson算法的思想是一致的。

一个求最小费用最大流的朴素算法是这样的：

1 求最小费用增广路

2 判断是否存在增广路，否的话算法终止。

3 增加增广路上边的流量

4 在增广路上添加必要的逆向负权边

5 goto 1

因为负权边的存在，求最小费用增广路就不可以用dijkstra算法。当然，我们可以用bellman-ford算法，可是这样的话求一次最短路的时间代价就是O(e*n)，e是边数，n是顶点数。代价大了点，如果能用dijkstra算法就好了。利用Johnson算法的思想，这是可以做到的。

第一次求最短路可以用dijkstra算法（如果一开始就有负权边，那就用bellman-ford算法，这没关系），求出源点到所有点的距离，嗯，我说的距离是指路径上边的费用之和的最小值。注意，要求出到所有点的距离，而不是求出到汇点的距离就完事了。

假设有一条边u->v，源点到u的距离是d[u]，到v的距离是d[v]，边的费用（权值）是w(u，v)。很显然，d[u]+w(u,v)>=d[v]，不然的话，你会发现一条更好的路径从源点到v。问题是，什么时候取等呢？当u->v在v的最优路径上，范围说小一点，当u->v在从源点到汇点的最优路径，即最小费用增广路上。

好的，如果u->v被你增载了，你要开始添负权边v->u了，权值取负，就是-w(u,v)。负权就是讨厌，是正的就好了，dijkstra算法就可以再用了。怎么办呢，把负权边加个权值，让它非负。要加多少呢，d[v]-d[u]。当然不能只加一条边，对所有边，无论原有的还是新添的，按这个规则加，构造一个新的图：

对边a->b，新的边权w′(a,b)=w(a,b)+d[a]-d[b]

现在来看看你的杰作：

对原来的边u->v, w'(u,v)=w(u,v)+d[u]-d[v]: 记得么d[u]+w(u,v)>=d[v], 所以 w'(u,v) >= 0

对新加的负权边v->u, w'(v,u)=w(v,u)+d[v]-d[u]=-w(u,v)+d[v]-d[u]: 记得么d[u]+w(u,v)==d[v]，这里可是取等号的，所以w'(v,u) == 0

哈哈，这下所有边又是非负的了。

可是，问题是，为啥不每个边加个足够大的正数，这样不是所有边也都是正的了么。仔细想想，边权为啥要为正，不就是为了求源点到汇点的最短路方便么，可是，都加大正数的话，你求出的最短路和原来图的最短路能一致么，不能，为啥，画个三角形，自己想想。可是，我的方法就能一致么，能。我证明给你看。

假设从源点s到汇点t有一条路径s->a->b->c->d….->t,在原图中的路径长度为

w(s,a)+w(a,b)+w(b,c)+…+w(x,t)

在新图中的路径为

w'(s,a)+w'(a,b)+w'(b,c)+…w'(x,t)

展开来就是

w(s,a)+d[a]-d[s]+w(a,b)+d[b]-d[a]+w(c,d)+d[d]-d[b]+….+w(x,t)+d[t]-d[x]

消阿消，d[a]和-d[a]，d[b]和-d[b]…d[x]和-d[x]，剩下什么呢：

w(s,a)+w(a,b)+w(b,c)+…+w(x,t)+d[t]-d[s]

噢，不就比原图中多d[t]-d[s]么（其实d[s]==0）。这可是对所有s到t的路径都成立的，既然所有路径，在新图中的权值都比在原图中的权值多了d[t]，那么，新图的最短路，也就对应原图的最短路，只不过路径长度多了d[t]，这不仅对t成立，对所有节点u都成立，只不过新图中到u的最短路长度比原图多了d[u]。

好，用dijkstra算法，第二次求出最短路。然后求出新的d'[u]，然后添加新的边，然后准备第三次的dijkstra算法。。。为什么第二次可以这样做，第三次还可以这样做，第三次的原图可能有很多负权边啊？我可没说过w(u,v)>=0这样的限制，所以，即使原图有负权边还是可以这样做的。

好了，第一次dijkstra算法（或者bellman-ford算法，如果有负权边的话，只用一次，不会成为瓶颈的），然后每次求最小增广路用一次修改的dijkstra算法。这个算法求最小费用最大流复杂度是O(m*n*n), m是最大流量，或者是求增广路次数的上界。最后，如果用这个算法来求最优匹配问题，复杂度是O(n^3)的。

Bellman-ford版:

```
网络中最小费用最大流
|参数含义:   n代表网络中的总节点数
|          net[][]代表剩余网络
|          cost[][]代表单位费用
|          path[]保存增广路径
|          ecost[]源点到各点的最短路
|算法:初始最小费用和最大流均为，寻找单位费用最短路
|在最短路中求出最大流，即为增广路，再修改剩余网络，直到无可增广路为止
|返回值:      最小费用，最大流量
└**** **** **** **** **** ****/
 const int NMAX = 210;
 int net[NMAX][NMAX], cost[NMAX][NMAX];
 int path[NMAX], ecost[NMAX];
 int n;
 bool bellman_ford()
⊟…{
 |  int i,j;
 |  memset(path,-1,sizeof(path));
 |  fill(ecost, ecost+NMAX, INT_MAX);
 |  ecost[0] = 0;
 |
 |  bool flag = true;
   while(flag) …{
```

```
        flag = false;
      for(i=0;i<=n;i++) …{
         if(ecost[i] == INT_MAX) …{
              continue ;
         }
         for(j=0;j<=n;j++) …{
            if(net[i][j] > 0 && ecost[i]+cost[i][j] < ecost[j]) …{
                 flag = true;
                 ecost[j] = ecost[i]+cost[i][j];
                 path[j] = i;
            }
         }
      }
   }
   return ecost[n] != INT_MAX;
}

int min_cost_max_flow()
…{
   int i,j;
   int mincost = 0, maxflow = 0;
   while( bellman_ford() ) …{
      int now = n;
      int neck = INT_MAX;
      while(now != 0) …{
         int pre = path[now];
         neck = min(neck, net[pre][now]);
         now = pre;
      }
      maxflow += neck;
      now = n;
      while(now != 0) …{
         int pre = path[now];
         net[pre][now] -= neck;
         net[now][pre] += neck;
         cost[now][pre] = - cost[pre][now];
         mincost += cost[pre][now] * neck;
         now = pre;
      }
   }
   return mincost;
}
```


上一篇　宁波赛区网络预选赛 C题 Code Merging 求最小字典序的LCS

下一篇　数论中的一些公式（转）