基于点割集的最短路径算法的改进与应用

吴漫 白明丽 曾咏欣 蒋峰 利叶斌
(湖南科技大学数学与计算科学学院,湘潭,411100)

摘 要 本文通过介绍图论中的重要内容——割点与点割集的概念,将寻找割点与点割集的算法,与经典的 Dijkstra 算法结合,形成改进的并行算法并予以实现与应用,为寻找无向图的最短路径提供了理论依据,并用其改进了路由协议 OSPF 中的路由选择算法,降低了算法的时间复杂度.

关键词 割点 点割集 Dijkstra 算法 路由选择算法

Improvement and Application of Shortest Path Algorithm Based on Point Cut Set

Wu Man Bai Mingli Zeng Yongxin Jiang Feng Li Yebin
(Hunan University of Science and Technology, Xiangtan 411100, China)

Abstract By introducing the concept of cut point and point cut set, which are the important part of graph theory, this paper combines the algorithm of finding cut point and point cut set with the classical Dijkstra algorithm to form an improved parallel algorithm. The application of the improved parallel algorithm is also given. It provides a theoretical basis for finding the shortest path of undirected graph, and improves the routing algorithm in the routing protocol OSPF, which reduces the time complexity of the algorithm.

Key words Cut point Point cut set Dijkstra algorithm Routing selection algorithm

1 引言

图论是组合数学的一个分支,与其他的数学分支,如群论、矩阵论等有着密切的联系.凡有二元关系的系统,图论均可提供一种数学模型,因而它在许多科学领域中具有越来越重要的地位.最短路问题是图论中的一个重要研究问题,在图论中占有非常重要的地位,它不仅广泛应用于工程中的诸多问题,如信息的通讯、线路的规划、管道的铺设等,也常作为一种基本工具,用于解决其他的最优化问题,以及预测与决策问题等.

国内外许多专家学者对此问题进行了深入研究.经典的图论与不断发展完善的计算机数

湖南省大学生研究性学习和创新性试验计划项目(201810534042)资助
收稿日期:2018 年 11 月 20 日

据结构及算法的有效结合使得新的最短路径算法不断涌现.1962 年由弗洛伊德(Floyd)提出的一个算法,又称传递闭包方法,为求每一对节点之间的最短路径提供新的思想与策略[1];1982 年 João Carlos Namorado Climaco 与 Ernesto Queirós Vieira Martins 研究得一种双列的最短路径算法[2].它们在空间复杂度、时间复杂度、易实现性及应用范围等方面各具特色.

本论文考虑一些特殊图中的最短路径算法问题,通过引入将大对象分割为若干小对象的粒计算思想,提出利用点割集将大图分为互不相关的小块,以提高求最短路径问题效率的算法.我们先根据割点判定定理对一般图进行判定,在具有割点的图中进行 Dijkstra 最短路径算法的改进,并运用计算机程序实现.然后在含一般点割集的图中,对分块利用 Dijkstra 算法求最短路径的问题进行了更加深入的探讨与研究.最后将改进的算法应用到路由协议 OSPF 中,改进了路由选择算法,降低了算法的时间复杂度.

2 预备知识

2.1 相关定义

定义 1(连通分支)[3] 在无向图 G 中,任何两个顶点是连通的,则称 G 为连通图,否则称 G 为非连通图或分离图. G 中的任意划分块称为 G 的一个连通分支,图 G 的连通分支数记为 $\rho(G)$.

定义 2(点割集、割点)[3] 设 $G=<V,E>$ 为无向图,若存在 $V'\subset V$,使 G 的子图 $G-V'$ 的连通分支数大于 G 的连通分支数,即 $\rho(G-V')>\rho(G)$,而删除 V' 的任何真子集 V'' 时,$\rho(G-V'')=\rho(G)$,则称 V' 为 G 的点割集.若点割集中只有一个顶点,则称此点为割点.

定义 3(连通度)[3] 设 G 是无向图,称 $k(G)=\min\{card(V')\mid V'$ 是 G 的点割集或 $G-V'$ 是平凡图$\}$,为 G 的点连通度,简称连通度,即为了产生一个不连通图需要删去的 G 的点的最少数目.

定义 4(块)[4] 没有割点的非平凡连通图称为块.无向图 G 中不含割点的极大连通子图称为图 G 的块.

定义 5(最短路径)[3] 设带权图 $G=<V,E,W>$, G 中每条边带的权均大于等于 0. $u,v\in V$ 为 G 中任意两个顶点,从 u 到 v 的所有通路中带权最小的通路称为 u 到 v 的最短路径.求给定两顶点之间的最短路径问题称为最短路径问题.

定理 1[3] 图 G 中的一个顶点 v 是割点,当且仅当存在两个不同于 v 的顶点之间的每条通路都经过 v .

性质 1(最短路径的子路径是最短路径)[5] 对于一给定的带权有向图 $G=(V,E)$,所定义的权函数为 $w:E\to R$.设 $p=<v_1,v_2,\cdots,v_k>$ 是从 v_1 到 v_k 的最短路径,对于任意 i,j ,

其中 $1 \leqslant i \leqslant j \leqslant k$，设 $p_{ij} = < v_i, v_{i+1}, \cdots, v_j >$ 为 p 中从顶点 v_i 到 v_j 的子路径，那么，p_{ij} 是从 v_i 到 v_j 的最短路径.

2.2 算法 1：经典 Dijkstra 算法

Dijkstra 算法是荷兰计算机科学家 Edsger Wybe Dijkstra 于 1956 年提出的解单源最短路径问题的贪心算法.其基本思想为，设置顶点集合 S 并不断地作贪心选择来扩充这个集合.它是通过为每个顶点 v 保留目前为止所找到的从 s 到 v 的最短路径来工作的，其具体步骤如下：

输入：带权无向图 G 的邻接矩阵 M，确定源点 s.

输出：源点 s 到图中其他顶点的最短路径及各自的长度.

步骤 1：对于图 $G=(V,E)$，初始时数组 S 中仅含有源点 s，即 $S=\{s\}$，而 T 集包含除源点 s 以外的图 G 中所有的点，即 $T=\{u \mid u \in V$ 且 $u \neq s\}$；

步骤 2：初始化数组 $dist$，对于 S 中的源点 s，如果 T 中的点有到源点的弧(直接路径)，则 $dist[i]$ 为弧值，否则 $dist[i]$ 为无穷(表示无直接路径)；

步骤 3：从 T 中选取从目前的 $dist$ 来看，到 S 内的点距离最短的一个 T 中的点加入到 S 中；

步骤 4：更新 $dist$，更新集合 S 与 T；

步骤 5：重复执行步骤 3、步骤 4 直到 $S=V$ 结束.

该算法时间复杂度为 $O(n^2)$，其中 n 为顶点个数.

3 寻找割点、点割集的算法研究

3.1 寻找割点与点割集的已有算法

通过分析割点与点割集的定义与性质，结合所查阅的资料，我们查找到以下两种算法，为快速寻找给定图中所存在的割点与点割集提供了可行的方法.

Tarjan 算法是由 Robert Tarjan 提出的求解有向图强连通分量的线性时间的、基于对图深度优先搜索的算法.每个连通分支为搜索树中的一棵子树，搜索时，把当前搜索树中未处理的节点加入一个堆栈，回溯时可以判断栈顶到栈中的节点是否为一个连通分支.可以用于解决求无向图的割点问题.

Dinic 算法是由以色列计算机科学家 Yefim(Chaim) A. Dinic 于 1970 年提出的利用最大流定理求图的最小点割集的算法.它是一种对于网络流问题的增广路算法，通过对残量网络进行分层，并在层次图上寻找增广路的方式，实现了在 $O(n^2m)$ 的时间内求出网络的最大流，从而确定图的最小点割集.

通过对两种算法的学习与研究，我们将两种算法结合在一起，减少了输入条件、输入次数

以及算法调用次数，形成了一种改进的寻找割点与点割集的算法，由于割点是点割集的特殊情况，故综合起来我们命名该改进算法为点割集搜寻算法.

3.2 算法 2：点割集搜寻算法

通过图的基本信息，判断有无割点，若存在割点，由于我们以割点优先对图进行划分，此时该图是否具有点割集对图的划分并无影响，故在原图存在割点的前提下，将不再继续判断是否具有点割集.

从而我们设计的算法的思想为：输入图的基本信息，判断是否存在割点，若存在割点，则输出割点信息；若不存在割点，则判断是否存在点割集.其具体步骤如下：

输入：输入无向图 G 的顶点个数 n 、边的信息，设置源点 s ，汇点 t .

输出：图的割点或点割集.

步骤 1：为每个节点 i 设置以下两个变量；

时间戳 $num[i]$ ：深度优先搜索遍历时节点 i 被搜索的次序，一旦某个点被搜索到，这个时间戳就不再改变；

最小时间戳 $low[i]$ ：节点 i 及其子树中的点，通过非父子边，能够回溯到的最早位于栈中的节点；

步骤 2：选定一个根节点，初始化 $num[]=low[]=++count$ ($count$ 为计数器)，进行深度遍历；

步骤 3：判断与当前访问节点连通的点是否已经被访问过，若没有，则进行访问.若该点访问过则已经入栈，说明形成了环，则更新 $low[i]$ ；

步骤 4：在不断深搜的过程中如果出边遍历完，那么就进行回溯，回溯时不断比较 $low[i]$ ，取最小的 low 值.对于节点 u 及其父节点 v ，如果 $low[v]\geqslant num[u]$ ，此时 u 就是割点，将 u 存入数组 $array$ ；

步骤 5：判断数组 $array$ 是否为空，若非空，则输出数组元素即为割点，程序结束；否则，执行步骤 6；

步骤 6：判断源点 s 与汇点 t 之间是否直接有边相连，若有，则结束；若无边相连，执行步骤 7；

步骤 7：构造网络图，求初始化后图的最大流 $flow$ ；

步骤 8：从编号为 1 的顶点开始遍历，即 $i=1$ 开始，判断 $i\leqslant n$ 是否成立，若不成立，则表示已经对图中的顶点遍历完全，结束程序；若成立，则执行步骤 9；

步骤 9：判断结点 i 是否为源点或结点，若是，则 $i=i+1$，转入步骤 8；否则，将求出去掉结点 i 后重新构造的网络图的最大流 $flow_{new}$ 执行步骤 10；

步骤 10:判断 $flow_{new}$ 与 $flow$ 是否相同,若相同则表示结点 i 在最小点割集里,输出 i ;否则,将结点 i 放回, $i=i+1$,转入步骤 8.

4 基于割点与点割集的 Dijkstra 并行算法研究

通过对上述算法的编程实现,将割点、点割集与最短路径相结合,对传统的求最短路径算法 Dijkstra 算法进行改进.

4.1 基于割点的 Dijkstra 算法研究

4.1.1 改进后的并行算法的描述

首先,仅考虑无向图中割点的情形.由割点性质知,一个割点(设为 A)可以把一个连通图分为 $n(n\geqslant 2)$ 个连通分支.经分析,在无向连通图中寻找任意给定(设为 s,t)两点之间的最短路径可分为以下几种情况:

(1)当 $n=2$ 时,即割点把该图分为两个连通分支

1)若分割之后,顶点 s,t 处于不同的连通分支(如图 1 所示),则从割点 A 出发,利用 Dijkstra 算法,分别找到割点 A 到连通分支一中点 s 的最短路径 $(A,\cdots,s)$,及割点 A 到连通分支二中点 t 的最短路径 $(A,\cdots,t)$.由定理 1 得,路径 $(a,\cdots,A,\cdots,t)$ 是顶点 s,t 之间的最短路径.

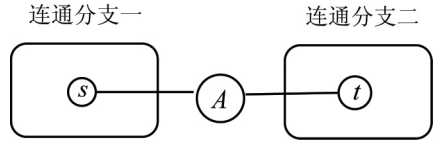

图 1 分割情况一

2)若分割之后,顶点 s,t 处于同一连通分支(如图 2 所示),则顶点 s,t 之间的最短路径必不过连通分支一中的任何顶点(简化理解为三角形三边关系即可理解,故不再详细说明).若此时连通分支二为一个块,问题规模减小到在原图中去掉连通分支一里所有顶点及所关联的边之后,寻找顶点 s,t 之间的最短路径.此时可直接在减小规模后的连通图中利用 Dijkstra 算法寻找顶点 s,t 之间的最短路径,算法时间复杂度较从开始就利用 Dijkstra 算法简化了许多.

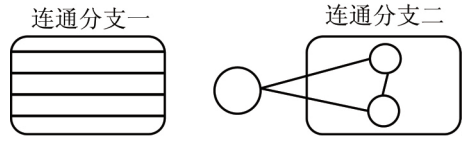

图 2 分割情况二

3)若连通分支二中存在割点(记为 B),割点 B 将连通分支二再次分割为两块(如图 3 所示),显然两割点 A,B 之间的最短距离为固定的,即 $(A,\cdots,B)=(B,\cdots,A)$,利用 Dijkstra 算法分别求出从割点 A 到顶点 s 、顶点 t 的最短路径 $(A,\cdots,s)$,$(A,\cdots,t)$,及在连通分支二中利用 Dijkstra 算法分别求出从割点 B 到顶点 s 、顶点 t 的最短路径 $(B,\cdots,s)$,$(B,\cdots,t)$,则寻找顶点 s,t 之间的最短路径问题化为在路径 $(s,\cdots,A,\cdots,t)$,路径 $(s,\cdots,A,\cdots,B,\cdots,t)$,路径 $(s,\cdots,B,\cdots,t)$ 及路径 $(s,\cdots,B,\cdots,A,\cdots,t)$ 这四种组合中选择权值最小的一条,即为顶点 s,t 之间的最短路径.

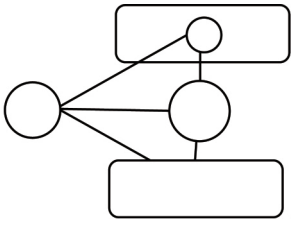

图 3 分割情况三

若连通分支二中存在割点但分割之后顶点 s,t 仍在同一分支内,化为之前所讨论的情况,应用到算法中递归调用即可,只是增加了组合数,原理并未改变.

(2)当 $n=3$ 时,即割点(记作 A)把该图分为三个连通分支.经过上述讨论我们发现,分割之后若顶点 s,t 仍在同一连通分支,则所得问题与原问题为同类型,故以下的讨论,我们仅考虑分割之后,顶点 s,t 处于不同连通分支的情形.

顶点 s,t 分别在不同的连通分支中,不妨记作连通分支一、连通分支二,连通分支三中无目标顶点.由定理 1 易知,顶点 s,t 之间的最短路径必过割点 A ,且不经过连通分支三中的任意一个顶点,即与连通分支三无关.

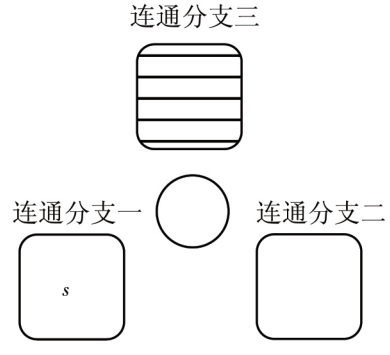

图 4 分割情况四

此时转化为在连通分支一、二中结合割点 A 寻找顶点 s,t 之间的最短路径的问题，即转化为(1)中所讨论的情况，故不再说明.

(3)当 n 为更大的自然数时，即割点把该图分为多个连通分支，与(2)类似，顶点 s,t 之间必经过割点且仅与顶点 s,t 所在的两个连通分支有关，大大减小了搜索范围，降低了问题复杂度.

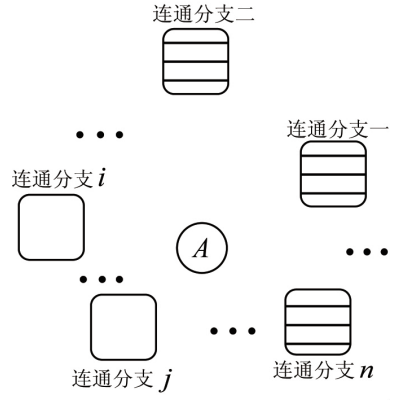

图 5 分割情况五

4.1.2 算法 3：基于割点求最短路径的 Dijkstra 算法

输入：输入无向图的 G 邻接表，确定源点 s 及目标顶点 t .

输出：输出最短路径 $(s,\cdots,t)$ 及其长度 l .

步骤 1：判断无向图 G 是否有割点，若不存在割点，则利用 Dijkstra 算法直接寻找源点 s 到顶点 t 的最短路径 $(s,\cdots,t)$ 及其长度，输出；若存在割点(不妨记为 A)，通过割点将图分为 $n(n\geqslant 2)$ 个连通分支，并转入步骤 3；

步骤 3：判断源点 s 与目标顶点 t 是否在同一连通分支，若在同一连通分支，则在该连通分支中判断是否具有割点，即转入步骤 2；若不在同一连通分支中，转入步骤 4；

步骤 4：利用 Dijkstra 算法分别确定割点 A 到顶点 s,t 的最短路径 $(A,\cdots,s)$，$(A,\cdots,t)$ 及其长度 l_1,l_2，则最短路径为 $(s,\cdots,A,\cdots,t)$ 及其长度 $l=l_1+l_2$，输出.

4.1.3 举例说明

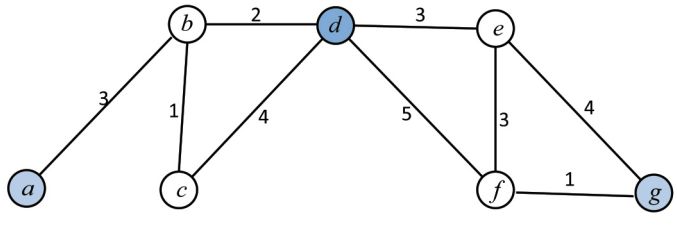

图 6 例图 1

利用割点结合 Dijkstra 算法在如上所示图 6 中，寻找顶点 a 与顶点 g 之间的最短路径.

步骤 1：确定起点为 a ，终点为 g ；

步骤 2：判断图中有无割点，找到 d 为割点；

步骤 3：通过割点 d ，将连通图分为两个连通分支；

步骤 4：判断此时顶点 a ，g 两点是否位于同一连通分支，结果为否；

步骤 5：利用 Dijkstra 分别求 d 到 a 和 d 到 g 的最短路径 $(d,\cdots,a)$ ，$(d,\cdots,g)$ 及其对应长度 l_1，l_2，则 $(a,\cdots,d,\cdots,g)=(d,\cdots,a)+(d,\cdots,g)$ 为 a ，g 之间的最短路径，$l=l_1+l_2$ 为最短路径的长度.

通过 MATLAB 作图求解可得最短路径如下图 7 所示：

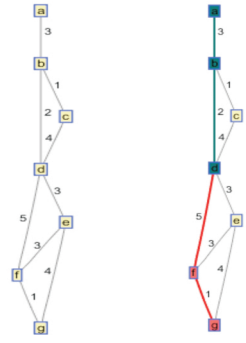

图 7　最短路径示意图

最短路径为 (a,b,d,f,g) ，最短路径长度为 11.

4.1.4　改进前后对比

以上述应用为例，用改进前后算法寻找点 a 到点 g 的最短路径的求解过程如下表 1 所示：

表 1　通过经典 Dijkstra 算法求最短路径的过程

终点 Dist	b	c	d	e	f	g	S(终点集)
K=1	3<a,b>	∞	∞	∞	∞	∞	{a,b}
K=2	∞	4<a,b,c>	5<a,b,d>	∞	∞	∞	{a,b,c}
K=3	∞	∞	5<a,b,d>	∞	∞	∞	{a,b,c,d}
K=4	∞	∞	∞	8<a,b,d,e>	10<a,b,d,f>	∞	{a,b,c,d,e}
K=5	∞	∞	∞	∞	10<a,b,d,f>	12<a,b,d,e,g>	{a,b,c,d,e,f}
K=6	∞	∞	∞	∞	∞	11<a,b,d,f,g>	{a,b,c,d,e,f,g}

表 2　通过并行算法求最短路径的过程

终点 Dist	a	b	c	e	f	g	S(终点集)
K=1	∞	2<b,d>	4<c,d>	3<d,e>	5<d,f>	∞	{b,d,e}
K=2	5<a,b,d>	∞	3<b,c,d>	∞	5<d,f>	7<d,e,g>	{b,c,d,e,f}
K=3	5<a,b,d>	∞	∞	∞	∞	6<d,f,g>	{a,b,c,d,e,f,g}

通过表 1 与表 2 的对比发现，并行算法寻找最短路径所需的计算步骤(3 步)比经典 Dijkstra 算法(6 步)所需步骤明显要少.并且利用并行算法所求最短路径，与由经典 Dijkstra 算法所求结果一致，验证了改进后并行算法的高效性.

在图的顶点更多或割点所划分连通分支数更多时，由于每个连通分支中顶点个数相对更少，与原图直接利用经典 Dijkstra 算法寻找最短路径相比，我们改进后的算法效率更高.

4.2 基于点割集的 Dijkstra 算法研究

研究了割点结合经典 Dijkstra 算法求最短路径的特殊情形之后，我们再考虑无向图中一般点割集(点割集中顶点个数 $n \geqslant 2$)的情形.

4.2.1 改进后并行算法的描述

在无向图 G 中，寻找顶点 s,t 之间的最短路径.所找到的点割集有两种情况.

(1)任意一个点割集都不能将点 s,t 分到两个不同的连通分支，即 s,t 在同一连通分支(不妨记作连通分支一)中，则与割点分析过程类似，此时顶点 s,t 之间的最短路径必不过其他连通分支.

ⅰ)若连通分支一为一个块，问题规模减小到在原图中去掉连通分支一里所有顶点及所关联的边之后，寻找顶点 s,t 之间的最短路径.此时可直接在减小规模后的连通图中利用 Dijkstra 算法寻找顶点 s,t 之间的最短路径，算法时间复杂度较从开始就利用 Dijkstra 算法简化了许多.

ⅱ)若连通分支一中存在割点，则情况化为割点结合 Dijkstra 算法所讨论的内容，此处不再重复.

(2)存在点割集将 s,t 分到不同的连通分支. s,t 两点之间的最短路径可能会经过去掉点割集之后形成的不含 s,t 的连通分支(与仅含有割点情况相似)，此时利用分块思想及组合原理寻找顶点 s,t 的最短路径，由于每块中顶点个数的大量减少，此时在点割集满足以下条件

ⅰ)点割集中点的个数尽可能少，

ⅱ)将点割集去掉之后，使得连通分支中点的个数尽可能均匀分配[6]

时比直接用 Dijkstra 算法对图 G 中的 s,t 两点求最短路径的复杂度要低.

记点割集为 W，含点 s 的连通分支为连通分支一，含点 t 的连通分支为连通分支二.

找到连通分支一中与 W 中顶点之间有边相连的点的集合，记为 C，找到连通分支二中与 W 中顶点之间有边相连的点的集合，记为 D.

利用 Dijkstra 算法分别求出顶点 s 到集合 C 中各个点的最短路径集合，记为 E；用 Dijkstra 算法分别求出顶点 t 到集合 D 中各个点的最短路径集合，记为 F；然后用 Dijkstra 算法求 C 中各点到 D 中各点的最短路径的集合，记为 S.

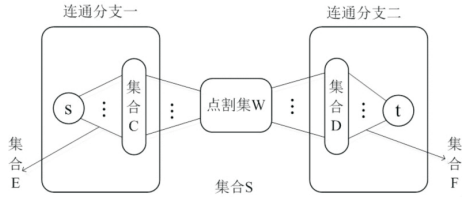

图 8　分割情况图

遍历集合 S 中的每条路径，由当前路径的起点(即集合 C 中的点)，衔接集合 E 中与该起点相连接的路径，由该路径的终点(即集合 D 中的点)，衔接集合 F 中与该终点相连接的路径，即为 s，t 之间经过该路径的最短路径，最后比较所有的这些由集合 S 中局部路径所确定的最短路径，求得点 s 到点 t 的最短路径.

运用组合数和分块的想法将图 G 分成不同的连通分支，然后用 Dijkstra 算法对于每个分块求最短路径，相对于对整个图 G 来求 s，t 之间的最短路径复杂度有所降低.

4.2.2　**算法 4：基于点割集求最短路径的 Dijkstra 算法**

输入：输入图 G，规定源点 s，目标顶点 t.

输出：输出最短路径 $path$ 及其长度 $length$.

步骤 1：求出图 G 的点割集；

步骤 2：判断点割集是否将顶点 s，t 分割到两个不同的连通分支，若在同一个连通分支，则直接利用 Dijkstra 算法求 s，t 两点的最短路径，输出；若不在同一连通分支，则执行步骤 3；

步骤 3：用 Dijkstra 算法分别求出源点 s 到集合 C 中各个点的最短路径的集合，记作 E；用 Dijkstra 算法分别求出顶点 t 到集合 D 中各个顶点的最短路径的集合，记作 F；用 Dijkstra 算法求出集合 C 与集合 D 中各点的最短路径集合，记作 S；

步骤 4：用组合思想，根据集合 E，F，S，求得源点 s 到目标顶点 t 的最短路径及其长度，输出.

4.2.3　**举例说明**

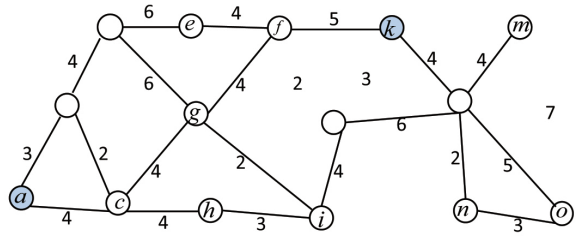

图 9　例图 2

利用点割集结合 Dijkstra 算法在如上所示无向图 G 中，寻找顶点 a 与顶点 k 之间的最短路径.

步骤 1：确定起点为 a ，终点为 k ；

步骤 2：判断图 G 中有无割点，得到割点为 l ；

步骤 3：通过割点 l ，将连通图 G 分为两个连通分支 G_1，G_2；

步骤 4：判断此时 a，k 两点是否位于同一连通分支，结果为是；

步骤 5：判断 a，k 两点所在连通分支 G_1 内有无割点，结果为无；

步骤 6：为使得去掉点割集后各个连通分支中点的个数均匀分配且使点割集中的点可能少，这里选点割集 $W_1=\{e,g,h\}$ 将 G_1 划分为两块，并分别把含点 a ，点 k 的连通分支记为 G_3，G_4；

步骤 7：分别找出 G_3，G_4 中与点割集 W_1 中各点相邻的点集 $S_1=\{c,d\}$，$S_2=\{s,i\}$ ；

步骤 8：用 Dijkstra 算法分别求出点 s 到点集 S_1 中各点的最短路径，记作 A ，点 t 到点集 S_2 中各点的最短路径，记作 B ，点集 S_1 中各点到点集 S_2 中各点的最短路径，记为 C ；

步骤 9：用组合思想，利用集合 A，B，C 求得点 s 到点 k 的最短路径.

首先对无向图进行编号，通过 MATLAB 作图求解可得最短路径如下图所示.可得最短路径为 (a,c,g,f,k) ，其长度为 17.

所得最短路径如下图 10 所示：

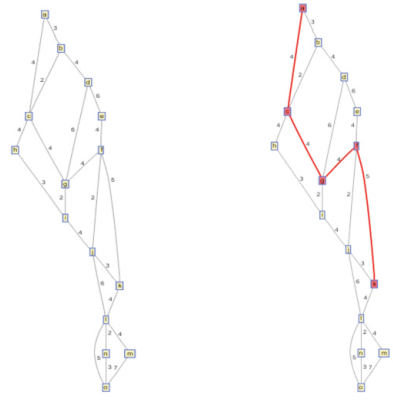

图 10　最短路径示意图二

对上图中路径寻找过程给出更为详细的描述，如下所示：

表 3　两种算法在不含有割点的加权图的运行结果对比图

Dijkstra 算法		并行算法															
0	a			e						g						h	
3	b	6	d	4	f	6	d	4	c	4	f	2	i	4	c	3	i
4	c	10	b	6	j	10	b	8	a	9	k	6	j	6	b	7	j
7	d	13	a	9	k	13	a	8	a			9	k	8	a	10	k
8	h																
8	g																
10	i																
12	f																
13	e																
14	j																
17	k																

注：表中字母顺序为进入终点集的顺序，字母左边的数字表示到源点的最短距离，在经典 *Dijkstra* 算法中以点 a 为源点，在并行算法中分别以点割集 $\{e,g,h\}$ 中各点为源点.

由上表所得所有的组合有：(a,b,d,e,f,j,k)，其长度为 $13+9=22$；(a,b,d,g,f,k)，其长度为 $13+9=22$；(a,b,d,g,i,j,k)，其长度为 $13+9=22$；

(a,c,g,f,k)，其长度为 $8+9=17$；(a,c,g,i,j,k)，其长度为 $8+9=17$；(a,b,c,h,i,j,k)，其长度为 $8+10=18$，经比较可得最短路径为 (a,c,g,f,k) 及 (a,c,g,i,j,k)，且最短路径长度为 17.

综上所述，并行算法寻找各连通分支最短路径的计算步骤最大为 3，比经典 Dijkstra 算法(10 步)步骤少.通过组合得到的最短路径，与经典 Dijkstra 算法所求结果一致，并且给出了所有最短路径可能的结果，比经典 Dijkstra 算法应用性更广.

在图的复杂度更高、点割集所划分连通分支数更多且点割集中含顶点个数相对较少时，由于每个连通分支中顶点个数相对更少，组合数也大大减少，与原图直接利用经典 Dijkstra 算法寻找最短路径相比，改进后算法效率会更高.

5　应用：改进 OSPF 协议中的路由选择算法

OSPF 协议(开放最短路径优先)，是一种运行于路由器上，用来确定最佳到达路径的内部网关路由协议.它用于单一的自治系统，能迅速探测出自治系统内部的拓扑变化，建立出全网的拓扑结构图，著名的 Dijkstra 算法被用来计算最短路径生成树.

如下图 11 所示，OSPF 协议把自治系统划分为多个区域，其中有一种特殊的主干域，其它各区域都与主干域相连.在一个区域内部的路由器只知道本区域的完整网络拓扑，从其他区域

来的信息都由区域边界路由器概括，因此每个区域至少有一个区域边界路由器，这些区域边界路由器同时也是主干域内的主干路由器.协议中有 3 种路由：域内路由、域间路由、域外路由[7].域内路由仅由其所在区域本身的拓扑决定；当在自治系统内的两个普通区域之间进行数据传输时，需要通过主干域.

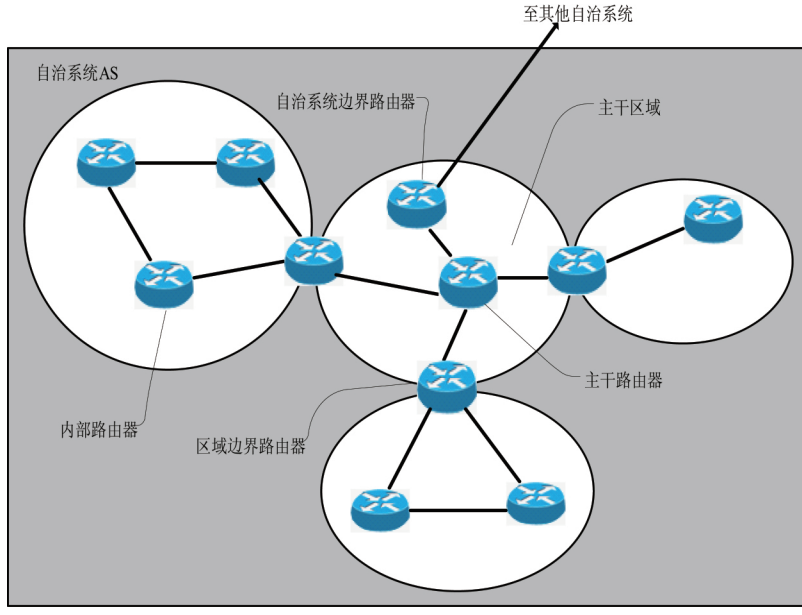

图 11 自治系统区域图

我们发现，本文改进的 Dijkstra 算法可以改进该 OSPF 协议中的路由选择算法，使得协议中的路由选择算法速度更快.这里，我们假设“代价”最小的路由为最佳路由.改进后的算法如下：

步骤 1：我们利用求割点和点割集算法找到可以使自治系统被划分后，各个区域内结点个数大致相同的割点和割点集，将他们作为区域边界路由器(同时也是主干路由器).

步骤 2：根据 OSPF 协议分层路由的特点，若起始路由器在同一个区域中，则两者之间存在一条域内路由，由该区域的网络拓扑结构，直接利用 Dijkstra 算法容易求得“代价”最小的路由.

步骤 3：若起始路由器在同一自治系统的不同区域中，则两者之间的最佳域间路由可以分解为三条最佳域内路由：从起始路由器到源区域边界路由器的最佳路由、从源区域边界路由器到目的区域边界路由器的最佳主干域内路由、从目的区域边界路由器到目的地的最佳路由.

我们可在各个区域中用 Dijkstra 算法求解出最佳路由，而由于区域的边界路由器并不一

定是唯一的，从起始路由器到目的地之间的路由可能有多种组合，因此主干域内需求解各个源区域边界路由器到目的区域边界路由器之间的最佳路由，源区域内需求解源路由器到该区域边界路由器的最佳路由，目的区域内需求解该区域边界路由器到目的路由器之间的最佳路由.将三个区域内所求的最佳路由以边界路由器为衔接点，组合后得到起始路由器之间的不同路由，从中选取“代价”最小的，便是从起始路由器到目的地的最佳路由.

为了更好的说明问题，下面举一个例子来说明：

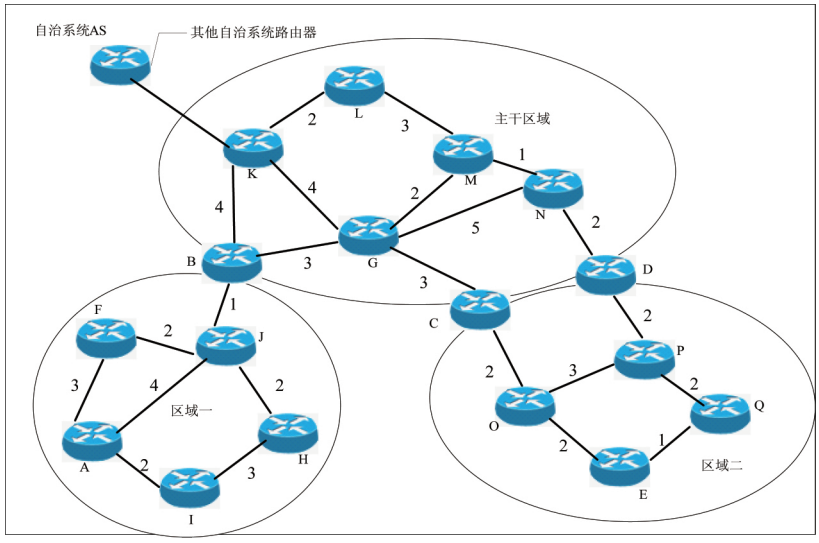

图 12　最佳路由

我们将路由器之间的“代价”量化为权值，如图 12 所示，同一自治系统内共有 17 个路由器，找出源路由器 A 到目的地路由器 E 之间的最佳路由.做计算步骤如下：

步骤 1：找出加权图的割点和点割集，这里为割点 B 、点割集 $\{C,D\}$ ，将自治系统均匀划分为区域一、主干区域、区域二.

步骤 2：由 A，E 在同一自治区的不同区域内，故同时对区域一、主干区域、区域二用 Dijkstra 算法求解最佳路由，得到区域一中最佳路由为：$A->J->B$ 、区域二中的最佳路由为：$B->G->C$、$B->G->M->N->D$ ，区域三中的最佳路由为：$C->O->E$、$D->P->Q->E$.

步骤 3：将各个区域所求的最佳路由，以割点和点割集为衔接点组合后有 $A->B->C->E$ ，$A->B->D->E$ ，选取出最佳的路由为：$A->B->C->E$.

本文在 OSPF 协议路由选择算法中引入了割点和点割集，把较复杂的自治域内路由选择问题分解成为三个小规模的域内路由选择子问题，由于这些子问题之间互不相关，则可并行计

算子问题，从而在原来的迪杰斯特拉算法的基础上，降低了计算的复杂度，进而减少了路由时所带来的开销.

同时，改进的算法比原来的聚合速度更快，聚合是同一自治区内所有路由器对最佳路径达成一致的过程.并且该算法灵活性强，可以快速、准确的适应各种网络环境，在某个网段发生故障时能很快发现故障，并为使用该网段的所有路由选择一条最佳路径.因此，本文的改进算法在计算机网络中具有很重要的现实意义.

6 结束语

我们已经在 C 语言与 Matlab 中对文中提出算法的进行了程序实现，对 Dijkstra 算法进行了初步的改进，使之在满足某些条件下优于经典的 Dijkstra 算法.有关代码可在以下网页中查找：https://bbs.csdn.net/topics/392566958.

由于条件的约束，目前所改进的算法应用范围有所限制，因此减少约束的条件，继续改进算法进行应用是我们继续要研究的内容，也将是未来创新的一个难点.

参考文献

[1] Robert W. Floyd. Shortest path[J], Communications of the ACM, 1962, 5(6):345－345.

[2] Climaco,J.C. and Martins E.V. A bicriterion shortest path algorithm[J], European Journal of Operational Research, 1982, 11(4):399－404.

[3] 李世群，马千里. 离散数学[M].天津：天津大学出版社，2010：157－180.

[4] 殷剑宏，吴开来. 图论及其算法[M].合肥：中国科学技术大学出版社，2003：63－70.

[5] Thomas H. Cormen , Charles E. Leiserson , et al. 算法导论[M].潘金贵译.北京：机械工业出版社，2006.

[6] 张清华，李鸿，沈文.基于点割集的并行最短路径算法[D].重庆：重庆邮电大学，2012.

[7] 王之梁，尹霞，李中杰. OSPF 协议测试中网络拓扑建模及其算法研究[J].计算机工程与应用，2002，(4)：1－35.