基于分支限界的关键路径求解算法

胡美群, 夏银水*, 王伦耀
（宁波大学 信息科学与工程学院, 浙江 宁波 315211）

摘要: 提出一种基于分支限界的关键路径求解算法, 将电路拓扑结构表示成有向带权网(WOEN), 寻找汇点, 使节点到汇点的最大路径时延为该节点分支限界的最小限值, 剪去违反分支限界最小限值的局部非关键路径的连接边以化简 WOEN. 新算法采取节点最大时延链表的存储结构, 使得 WOEN 的存储空间、关键路径计算空间以及计算结果的存储空间共享同一存储空间. 算法用C语言实现, 并在ISCAS标准电路上加以测试. 结果表明: 新算法比现有算法所需的存储空间更小, 求解关键路径的速度更快.

关键词: 关键路径; 分支限界; 算法; 时延; 集成电路
中图分类号: TP302 文献标识码: A 文章编号 1001-5132（2011）02-0037-05

电路的时延一直是集成电路的关键指标之一, 但随着集成电路制造工艺向亚微米及深亚微米发展, 电路规模越来越大, 对时延的估计和优化也越来越困难. 电路时延的估计和优化归根结底为关键路径的寻找[1], 因此如何有效及准确地求解关键路径一直是一个受到关注的问题.

当前, 国内外对于求解关键路径算法的研究有很多, 最常用的是拓扑排列算法[2-3]. 该算法采用拓扑排列和拓扑逆序排列, 分别计算出最早发生时间和最迟发生时间来判断关键路径的连接边. 再根据关键路径的连接边来寻找电路的关键路径以及估计关键路径时延. 基于该思想, 研究者们提出了多种仅需一次拓扑排列的关键路径算法. 根据算法采用存储结构、算法的存储空间以及运算复杂度的不同, 算法采用的存储结构主要可分为十字链表[4]、邻接矩阵[5-6]、稀疏矩阵[7]以及邻接表[8-9]等几种. 比较而言, 文献[9]采用 WOEN 存储空间和计算空间共享的方法, 实现了所需存储空间相对较少的目的. 然而, 现有算法都是在完整保留 WOEN 所有连接边的基础上查找关键路径, 从而导致了运算量大, 而且运算结果需要额外的空间加以存储, 也就造成现有算法存在存储空间大、速度慢的缺点.

笔者提出了一种基于分支限界的算法, 通过剪去局部非关键路径连接边简化 WOEN, 实现关键路径的求解. 在求解过程中, 通过构建节点最大时延链表的存储结构进行存储, 使得 WOEN 的存储空间、关键路径计算空间以及计算结果存储空间共享同个存储空间, 从而达到进一步减少存储空间、加快求解速度的目的.

1 理论基础

1 条路径由若干的电路单元和线网组成. 路径时延指的是信号经过该路径的时间间隔, 它包括两个部分: 电路单元本身的时延和互连线电阻电容引起的时延. 为研究方便, 在求解集成电路关键路径的算法中, 将集成电路拓扑网转换成向带权网(Weight on Edge Network, WOEN). WOEN 中的节点表示电路单元; 有向连接边表示电路的连接情况; 边上的权值表示电路中所对应的连接边以及与该连接边作为输出电路单元的总时延. 因此, WOEN 表示整个电路的拓扑情况. 为方便起见, 给出如下定义.

定义 **1** 设 $N=<V, E, W>$为有向带权网, 其中,

收稿日期: 2010-05-12. 宁波大学学报（理工版）网址: http://3xb.nbu.edu.cn
基金项目: 国家自然科学基金（60871022）; 浙江省自然科学基金（Y1080654）; 浙江省大学生新苗计划.
第一作者: 胡美群（1982－）, 女, 浙江丽水人, 在读硕士研究生, 主要研究方向: 集成电路的速度估计. E-mail: humeiqun88@163.com
*通讯作者: 夏银水（1963－）, 男, 浙江余姚人, 博导/研究员, 主要研究方向: 集成电路的综合和优化. E-mail: xiayinshui@nbu.edu.cn

V 为节点的集合; E 为有向连接边的集合; W 为有向连接边上权值的集合. 如果 N 满足: (1) N 是 1 个简单的有向带权无环网; (2) 1 个节点的输入个数称为入度, 输出个数称为出度, N 中有 1 个入度为 0 的源点和 1 个出度为 0 的汇点; (3)从节点 v_i 到节点 v_j 的有向连接边上带有权值 w_{ij}, 称此 N 为 WOEN.

定义 **2** 如果有 1 条从节点 $v_i(i=0,1,\cdots,n-1)$到节点 v_j ($j\neq i$, $j=0,1,\cdots,n-1$)的有向连接边, 则称节点 v_j 为节点 v_i 的分支节点. 如果节点 v_i 的出度为 m, 则节点 v_i 有 m 个分支节点.

定义 **3** 节点最大时延链表 $CP[v_i]=<len;BV>$ ($i=0,1,\cdots,n-1$), 其中, len 表示节点 v_i 到汇点的最大时延, 即节点 v_i 的分支限界的最小限值; $BV=\{bv_1, bv_2,\cdots,bv_m\}$表示节点 v_i 的 m 个分支节点以及节点 v_i 到各个分支节点的时延. 例如, $bv_k=<v_j, w_{ij}>$($k=0,1,\cdots,m$; $i=0,1,\cdots,n-1$; $j\neq i$, $j=0,1,\cdots,n-1$), 其中, 节点 v_j 是节点 v_i 的分支节点, 权值 w_{ij} 表示从节点 v_i 到分支节点 v_j 的时延.

例如图 1 中节点最大时延链表的表示结果为:

$CP[v_5]=<0;\text{NULL}>$; $CP[v_4]=<1; v_5,1>$; $CP[v_3]=<1; v_5,1>$;

$CP[v_2]=<3;v_4,2;v_3,1>$;$CP[v_1]=<3; v_3,2>$; $CP[v_0]=<5; v_2,2; v_1,2>$.

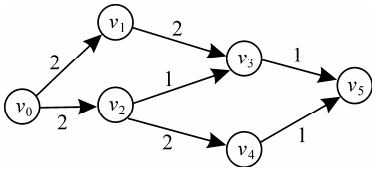

图 1 简单电路的 WOEN

分支限界算法通常被用来快速缩小解的搜索空间[10]. 所谓分支限界算法是将待求解问题分解为子问题, 并对这些子问题确定限界, 以进行逐步求解的方法. 其基本思想是对有约束条件的最优化问题的所有可能解(数目有限)空间进行搜索, 将全部可能解的解空间不断分割为越来越小的子集(称为分支), 并为每个子集内的解值计算 1 个下界或上界(称为限界). 并在每次分支后, 剪去超出限界的分支, 以提高搜索效率. 该过程一直进行到找出优化解为止, 该优化解的值不超出任何子集的限界. 因此这种算法可以求得最优解.

笔者将分支限界算法应用于关键路径的求解, 其基本方法如下:

(1) 由于求解关键路径问题的目标是计算电路时延最大的路径, 可设定目前最优解值为 len=0.

(2) 从尚未被经历的节点中读取 1 个节点 v_i, 计算节点 v_i 到汇点的最大时延. 应用分支限界算法, 若节点 v_i 有 m 个出度, 则从节点 v_i 输出的 m 个节点都是节点 v_i 的分支节点. 根据分支限界法则, 依次计算从节点 v_i 经过这 m 个分支节点到达汇点的时延, 将计算出的最大时延设置为分支限界的最小限值. 如果节点 v_i 通过某个分支节点到达汇点时的时延大于分支限界的最小限值, 那么更新节点 v_i 分支限界的最小限值, 仅保留节点 v_i 与该分支节点的连接边, 剪去节点 v_i 与其他计算过的分支节点的连接边. 但是, 如果节点 v_i 通过某个分支节点到达汇点时的时延小于分支限界的最小限值, 说明该分支节点违反了分支限界的最小限值, 则剪去节点 v_i 与该分支节点的连接边. 因此, 求解后节点 v_i 的分支限界的最小限值就是节点 v_i 到汇点的最大时延.

(3) 源点分支限界的最小限值就是从源点到汇点的最大时延, 即关键路径时延.

按照上述步骤, 即可求解电路的关键路径.

2 建议算法

设定节点到汇点的最大路径时延为分支限界的最小限值, 剪去 WOEN 中违反分支限界最小限值的连接边, 使 WOEN 中的连接边数不断收缩, 从剩下的可连通的连接边中来确定关键路径.

2.1 算法的数据结构

为使 WOEN 的存储、算法计算空间以及结果共享同个存储空间, 算法中采用节点最大时延链表进行存储, 从而在 WOEN 的存储空间中完成算法的计算和计算结果的存储. 存储结构设计如下:

(1) 节点最大时延链表 CP 用来存储节点分支限界的最小限值. WOEN 中的节点 v_i 到汇点的时延 $CP[v_i].len$ 就存储在 CP 中第 i 个节点中, 方便后序计算中随机的访问链表.

(2) 节点最大时延链表 CP 的分支节点(branch node)表示节点与分支节点的连接情况.

2.2 算法的步骤

所建议的算法是一种递推算法, 对电路拓扑网按照拓扑逆序排列, 依次计算出每个节点到汇

点的分支限界的最小限值. 由于采用拓扑逆序排列, 因此在计算过程中, 排在队列后面的节点计算只需要用到排在队列前面节点的计算结果, 所以没有递归和重复操作, 使得计算速度更快.

2.2.1 将电路拓扑网转换成 WOEN

在电路的拓扑网中添加 2 个节点. 在所有的原始输入电路单元前面加 1 个节点, 该节点的入度为 0, 并称为源点(in), 此时将源点与每个原始输入电路单元相连, 并且连接边上的权值(即时延)为 0; 在所有原始输出电路单元之后加 1 个节点, 该节点的出度为 0, 并称为汇点(out), 将每个原始输出电路单元都与汇点连接, 并且连接边的权值(即时延)也为 0. 此时, 就可将 1 个电路的拓扑网转换成 1 个 WOEN.

2.2.2 初始化节点的最大时延链表

节点最大时延链表 CP 按照节点编号的顺序结构进行存储, 以方便后序计算中随机地访问某个节点最大时延链表.

节点最大时延链表 CP 的表头表示各个节点分支限界的最小限值, 如 $CP[v_i].len$ 表示节点 v_i 的分支限界的最小限值. 初始化节点最大时延链表的表头, 使得每个节点分支限界的最小限值为 0, 即 $CP[v_i].len=0(i=0,1,\cdots,n-1)$.

节点最大时延链表 CP 的分支节点表示各个节点与分支节点的连接情况, 初始化为整个电路的 WOEN 连接情况.

2.2.3 求出各个节点分支限界的最小限值

按照拓扑逆序排列, 依次计算节点到汇点的最大时延, 并赋值给该节点分支限界的最小限值, 保留该节点与分支限界最小限值所对应的分支节点连接边, 剪去该节点与其他分支节点的连接边.

不失一般性: 设当前计算的节点为 v_i, 若节点 v_i 的出度为 m, 则节点 v_i 有 m 个分支节点. 若有 1 条从节点 v_i 到节点 v_j 的有向连接边, 则节点 v_j 为节点 v_i 的分支节点, 计算节点 v_i 与分支节点 v_j 的连接边时, 节点 v_i 分支限界的最小限值的取值如下:

$$CP[v_i].len=\begin{cases}CP[v_j].len+w_{ij}, \text{if } CP[v_j].len+w_{ij}>CP[v_i].len,\\ CP[v_i].len, \text{if } CP[v_j].len+w_{ij}\leqslant CP[v_i].len,\end{cases}$$

其中, $CP[v_i].len$ 表示节点 v_i 的分支限界的最小限值, 由于按照拓扑逆序排列顺序计算, 所以 $CP[v_j].len$ 已经被求解出来了.

如果 $CP[v_j].len+w_{ij}>CP[v_i].len$, 其表示分支节点 v_j 对应的分支节点最小限值大于其他计算过的分支节点所对应节点分支限界的最小限值, 所以更新节点 v_i 分支限界的最小限值为 $CP[v_j].len+w_{ij}$, 即 $CP[v_i].len=CP[v_j].len+w_{ij}$, 同时使计算过的分支节点中仅保留节点 v_i 与分支节点 v_j 的连接边, 剪去节点 v_i 与其他计算过的分支节点的连接边.

如果 $CP[v_j].len+w_{ij}=CP[v_i].len$, 则节点 v_i 分支限界最小限值 $CP[v_i].len$ 不变, 并保留节点 v_i 与分支节点 v_j 的连接边.

如果 $CP[v_j].len+w_{ij}<CP[v_i].len$, 表示分支节点 v_j 违反了节点 v_i 的分支限界的最小限值, 则剪去节点 v_i 与分支节点 v_j 的连接边.

2.2.4 计算关键路径时延并确定关键路径

重复 2.2.3 节的步骤, 直到计算出源点 in 的分支限界的最小限值. 计算结束后, 源点 in 的分支限界的最小限值 $CP[in].len$ 就是电路从源点到汇点的最大时延, 即电路的关键路径时延. 计算结束后剩下的从源点到汇点可连通的任意一条路径都是关键路径.

2.3 **算法举例**

如求下列四输入、两输出电路的关键路径, 其电路拓扑网如图 2 所示, 添加源点 in 和汇点 out, 将电路拓扑网转换成 WOEN, 如图 3 所示.

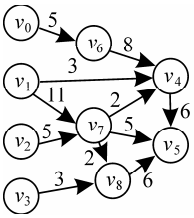

图 2　电路拓扑网

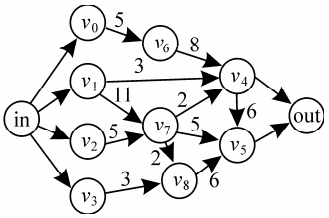

图 3　电路对应的 WOEN

进行拓扑逆序排列, 此 WOEN 的拓扑逆序排列结果是: out, v_5, v_4, v_8, v_7, v_6, v_1, v_2, v_3, v_0, in.

初始化节点最大时延链表 CP, 使每个节点分支限界最小限值为 0, 即 $CP[in].len=0$, $CP[v_i].len=0(i=0,1,\cdots,n-1)$, $CP[out].len=0$. 节点最大时延链表的分支节点表示电路 WOEN 的节点与分支节点的连接情况. 初始化结果如图 4 所示.

按照拓扑逆序排列的结果, 依次计算出各个节点分支限界的最小限值, 保留节点与分支限界

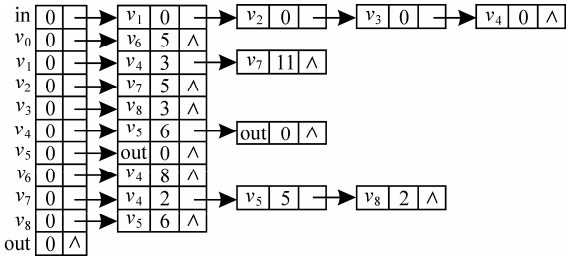

图 4 初始化的节点最大时延链表

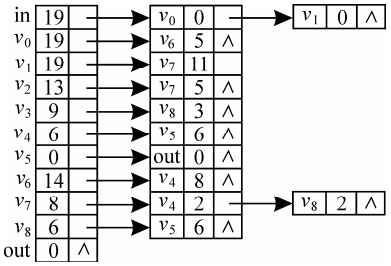

图 5 计算后的节点最大时延链表

最小限值对应的分支节点的连接边，同时剪去节点与违反分支限界最小限值的分支节点的连接边，一直到计算出源点为止.

例如求解 $CP[v_7].len$ 节点 v_7 的分支节点为 v_4, v_5 和 v_8. 按照拓扑逆序排列，节点 v_4, v_5, v_8 均排在节点 v_7 的前面，因此它们分支限界的最小限值已求出，分别为 $CP[v_4].len$=6, $CP[v_5].len$=0, $CP[v_8].len$=6. 在此基础上，求解节点 v_7 的分支限界的最小限值的步骤如下:

(1) 计算第 1 个分支节点 v_4，由于 $CP[v_4].len+w_{74}$=6+2=8>$CP[v_7].len$=0，因此将节点 v_7 的分支限界最小限值更新为 $CP[v_4].len+w_{74}$，即 $CP[v_7].len$=8，由于此时只计算了分支节点 v_4，所以直接保留节点 v_7 与分支节点 v_4 的连接边;

(2) 计算第 2 个分支节点 v_5，由于 $CP[v_5].len+w_{75}$=0+5=5< $CP[v_7].len$=8，表示分支节点 v_5 违反了节点 v_7 的分支限界最小限值，所以剪去节点 v_7 与分支节点 v_5 的连接边;

(3) 计算第 3 个分支节点 v_8，由于 $CP[v_8].len+w_{78}$=6+2=8=$CP[v_7].len$=8，所以保持节点 v_7 的分支限界最小限值不变，同时保留节点 v_7 与分支节点 v_8 的连接边;

计算后的最大时延链表如图 5 所示，源点 in 的分支限界最小限值 CP[in].len=19 就是电路从源点到汇点的最大时延，即电路的关键路径时延.

从源点开始搜索，得到的任一从源点到汇点的可连通的路径都是关键路径，如图 6 所示.

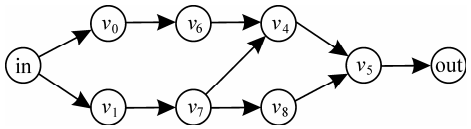

图 6 电路的关键路径网

3 实验结果及分析

所提出的算法已经用 C 语言加以实现，并在 Visual C++ 6.0 下运行. 在 ISCAS 标准电路上加以测试，结果如表 1 所示，由于算法执行的时间很短，而运行该程序的计算机的最小时间分辨率是 16 ms，若只执行 1 次会使结果的精确度降低，所以重复执行，以取得更精确的执行时间. 表 1 中的求解时间是算法执行 100 次的时间. 由表 1 可见，所提出的算法与已发表结果比较，存储空间有比较明显的降低，求解时间有比较明显的减少.

3.1 *存储空间分析*

设拓扑有向带权网 N=<V, E, W>是个具有 n 个节点 e 条边的 WOEN，在现有算法中，文献[9]所提出的算法所需要的存储空间最小，它采用扩展邻接表存储，使 WOEN 的存储空间和关键路径计算空间共享，其存储空间为 $7n$+3e+2h (h 为重复的节点数)，其中，n 个节点需要 4n 个存储空间，e 条边需要 3e 个存储空间，运算结果保存需要 2n+2h 个存储空间，WOEN 拓扑排序编码需要 n 个存储空间. 笔者提出的算法采用节点最大时延链表的存储结构进行存储，存放节点最大时延链表表头需要 2n 个存储单元，存放节点最大时延链表分支节点需

表 1 求解关键路径运行时间

电路名称	S27	S208	S344	S386	S400	S420	S510	S1196	S1238
节点数/个	19	136	179	164	188	299	304	674	721
边数/条	31	219	314	365	373	481	530	1 185	1 285
文献[3]结果/(s^{-1})	0.097	0.203	0.219	0.328	0.515	0.531	0.576	0.599	0.726
文献[9]结果/(s^{-1})	0.031	0.062	0.125	0.125	0.157	0.146	0.183	0.266	0.313
新算法结果/(s^{-1})	0.015	0.031	0.109	0.125	0.139	0.153	0.171	0.250	0.281

要 $3e$ 个存储单元, 拓扑逆序排列需要 n 个存储单元, 因此, 算法共需 $3n+3e$ 个存储单元, 比现有算法节约了 $4n+2h$ 个存储空间. 对于集成电路中连接度比较低(一般连接度为 30%)的情况[3], 节约的存储空间是比较显著的, 而所提出新算法的空间复杂度为 $O(n+e)$.

3.2 时间复杂度分析

新算法在计算节点分支限界最小限值时, 每个节点仅被计算 1 次, 因此外循环的次数为 n. 当某个节点被计算时, 内循环次数等于该节点的出度, 因此每条连接边也仅被计算 1 次, 没有递归操作, 也没有重复操作, 总的时间复杂度降低为 $O(e)$. 执行速度快, 因此可视为是一种高效的算法.

4 结论

新算法基于分支限界的思想, 在求解关键路径过程中, 通过剪去违反分支限界最小限值的连接边的方法求解电路关键路径. 该算法直接利用 WOEN 的存储空间进行关键路径计算和计算结果的存储, 无需额外空间, 所以算法具有 WOEN 的存储空间、关键路径计算空间以及计算结果存储空间共享同一存储空间的特点, 使求解关键路径所需的存储空间进一步减少. 此外, 算法中求解关键路径时延的操作和计算中不包含递归和重复操作, 相比较其他算法则更为简单、快速.

参考文献:

[1] 陈春鸿. 集成电路时延关键路径算法及其实现[J]. 微电子学, 1997, 27(6):375-379.

[2] 严蔚敏, 吴伟民. 数据结构[M]. 北京: 清华大学出版社, 2002.

[3] 佚名. 数据结构中关键路径算法的实现与应用[EB/OL]. [2007-07-01]. http://www.99inf.net/softwaredev/vc/13639.htm.

[4] Chang H, Abraham J A. An efficient critical path tracing algorithm for designing high performance VLSI systems [J]. Journal of Electronic Testing: Theory and Applications, 1997, 11(2):119-129.

[5] Chang D H, Son J H, Kim M H. Critical path identification in the context of a workflow[J]. Information and Softwear Technology. 2002, 44(7):405-417.

[6] Chen Shipin. Analysis of critical paths in a project network with fuzzy activity times[J]. European Journal of Operational Research, 2007, 183(1):442-459.

[7] Liou J J, Wang L C, Krstic A, et al. Critical path selection for deep sub-micro delay test and timing validation[J]. Transaction of Electronics, Communicatios and Computer Sciences. 2003, 86(12):3038-3048.

[8] Venkaramani, Budiu G, Chelcea M, et al. Global critical path: A tool for system-level timing analysis[J]. Journal of Construction Engineering and Management, 2007, 133 (7):483-491.

[9] Li Tianzhi. A novel algorithm for critical paths[J]. 2009 WRI World Congress on Computer Science and Information Engineering, 2009(1):226-229.

[10] Kariwala V, Cao Yi. Branch and bound method for multiobjective control structure design[J]. 4th IEEE Conference on Industrial Electronics and Applications, 2009 (4):2513-2518.

Critical Path Algorithm Based on Branch-and-bound

HU Mei-qun, XIA Yin-shui*, WANG Lun-yao

(Faculty of Information Science and Technology, Ningbo University, Ningbo 315211, China)

Abstract: A novel algorithm for finding critical paths based on branch-and-bound is proposed. By employing Weight On Edge Network (WOEN), those edges of the WOEN in non-critical paths will be removed during the evaluation thus at last one of the critical paths will be detected. Memory saving is implemented by storing the simplified WOEN, computing space and computing results in the shared memory. The algorithm is implemented in C and tested under ISCAS standard benchmarks. The proposed algorithm needs less memory and less time than those previously published algorithms.

Key words: critical path; branch-and-bound; algorithm; delay; VLSI

（责任编辑　章践立）