Algorithms for IVDP Matching on Short Trees

Dr.G. SUGANTHI,
dr_suganthi_wcc@yahoo.co.in
Department of Computer Science,
Women's Christian College,
Nagercoil ,INDIA-629001

Summary
A matching in a graph is a set of edges, no two of which have a vertex in common. If edge (u,v) belongs to a matching, we say that u and v are matched to each other. One is usually interested in finding maximum matching that is, matching having a maximum number of edges. Sometimes the edges have associated weights and one is interested in finding maximum weight matchings. Problems involving matching occur in many situations. Workers may be matched to jobs, machines to parts, players to teams etc. A path matching in a graph is a set of simple paths with distinct end vertices. Two paths are said to be vertex disjoint if they don't have any vertex in common. They are internally vertex disjoint if no vertex is an internal vertex of both the paths. A set of paths P in a graph G is said to be an internally vertex disjoint path matching(IVDP) if it is a path matching and every pair of paths in P are internally vertex disjoint. A perfect matching of G is a matching M which matches all the vertices except possibly one. This paper deals with the necessary and sufficient conditions for the existence of perfect IVDP matching for the trees of height 1 and 2. We have developed sequential and parallel algorithms and time complexity is determined. The odd and even trees are treated separately.

Key words:
Parallel Algorithms, Tree, Rooted Tree, Matching Problems, IVDP Matching

1. Introduction

Given an undirected graph G = (V,E), a matching is a set of edges such that no two edges in M incident on the same vertex WM[88]. Xavier[XA95] has defined perfect IVDP matchings and analysed its structural properties. In this paper we establish the necessary and sufficient conditions for the existence of Perfect IVDP matching for the trees of height 1 and 2. First we discuss the existence of Perfect IVDP matching for trees of height 1 and 2. We develop sequential and parallel algorithms for determining the existence. Here odd and even trees are treated separately.
A set of paths P in a graph G, is said to be an internally vertex disjoint path matching (IVDP) if it is a path matching and every pair of paths in P are Internally Vertex Disjoint. A perfect matching is a matching in which atmost one vertex is left unmatched. A tree with even number of nodes is an even tree. A tree with odd number of nodes is an odd tree.

Manuscript received October 5, 2009
Manuscript revised October 20, 2009

2. IVDP Matching in Trees

There are trees having no perfect IVDP matchings. Theorem 1 and 2 establish the equivalent conditions for the existence of perfect IVDP matching for even and odd trees respectively.

Theorem 1 : Let T be an even tree (tree with even number of nodes). If a node u of T has more than three leaf children, then T doesn't have a Perfect IVDP matching.
Theorem 2 : Let T be an odd tree. If a node u of T has more than four leaf children, then T doesn't have a perfect IVDP matching.

3. Trees of height 1

In this section we will construct a perfect IVDP matching for trees of height 1. The following are the results proved for trees of height 1.

Theorem 3: An even tree T of height 1 has a perfect IVDP matching if and only if T has at the most 4 nodes.
Theorem 4 : Let T be an odd tree of height 1. If the tree T has seven nodes then T has no perfect IVDP matching.
Theorem 5 : Let T be an odd tree in which there exists two nodes u and v each having four leaf children. Then T has no IVDP matching.
Theorem 6 : A tree of height 1 has IVDP matching if and only if it has at the most five nodes.

4. Algorithm for tree of height 1

The following is a very simple algorithm to find if a tree of height 1 is IVDP.

Algorithm IsIVDPHeight1(T)
Input : Tree T of height 1 in the form of parent array. n is the number of nodes. The nodes are numbered from 1 to n. p[i] is the parent of node i. The parent of root is itself. It is given that the height of the tree is 1.
Output : A Boolean value result to say if the tree is IVDP.

1. If n□ □ 5 the tree is IVDP
   So result = true
   Else The tree is not IVDP,
   So result = false

Complexity Analysis
Since n itself is given as an input, this can be done in O(1) time. This leads to the following theorem:
Theorem 7 : In a tree T of height 1 verification of the existence of a perfect IVDP can be done in O(1) time.

5. Even Trees of Height 2

In this section we develop algorithm to verify the existence of perfect IVDP matching in trees of height 2. Let T be an even tree of height 2. Let r be the root of T. Let nodd denotes the number of odd children of r, neven denotes the number of even children of r and l denotes the number of leaf nodes which are children of r.

Example : Consider the tree shown in Figure 6. r is the root. a is a child of r. The maximal subtree with a as the root has the nodes a, h, and k. So, a is an odd child of r. b is also an odd child of r. c, d and e are even children of r. Hence in this case

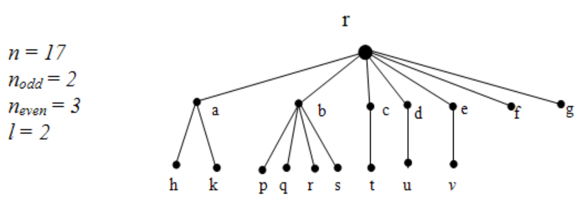

Figure 6. A tree t

The following are the results proved for the trees of height 2.

Theorem 8 : *Let T be an even tree with root r of height 2. T has a perfect IVDP matching if and only if the following two conditions are satisfied.*

1. *The sum of the number of leaf children of r and the number of odd children of r is at the most 3.*
2. *Each odd and even subtree of r has a perfect IVDP matching.*

Theorem 9 : *Let T be an even tree of height 2. T has a perfect IVDP matching if it satisfies the following three conditions*

1) $(n_{odd}+l)$ □ □ □ 3
2) The root doesnot have a subtree isomorphic to T_5, where T_5 is a tree of height 1 with 5 nodes.
3) Each *subtree of r has a perfect IVDP matching*.

6 Odd trees of height 2

The following is the result proved for odd trees of height 2.
Theorem 10 : *Let T be an odd tree of height 2. T has a perfect IVDP matching if it satisfies the following three conditions.*

1) *The root has at the most one subtree isomorphic to T_5.*
   Where T_5 is the tree of height 1 with 5 nodes.
2) *If T_5 is present then $n_{odd}+l$ □ □ 3*
   else $n_{odd}+l$ □ □ 4
3) *Each subtree of r has a perfect IVDP matching.*

7. Algorithm for Trees of height 2

Theorem 9 and 10 gives the necessary and sufficient conditions for the existence of perfect IVDP matching for trees of height 2. In this section we develop the sequential and parallel algorithms for determining the existence of perfect IVDP matching for trees of height 2.

7.1 To find the root

When the tree is represented in the form of the parent array *p[i]*, we can identify the root as follows:
Algorithm FindRoot (p, n)

```
{
For i = 1 to n
If p[i] = i then root = i
}
```

In sequential algorithm, this can be implemented in *O(n)* time.

7.2 To count the number of leaf children for each node

Let *T* be the tree with root *r*. The tree is represented in the form of parent array *p[i]*. The algorithm to find the number of leaf children is given below.

Algorithm ***FindNoOfLeafChildren (p, n)***
Input : *p[i]* *Parent array*
Output : *1) C[i] number of children for i*
2) LC[i] number of leaf child for I

```
Step 0 : Initialize C[i] = 0 and LC[i] = 0 for i = 1 to n.
Step 1 : For i = 1 to n
         {    C[p[i]] = C[p[i]] + 1
         }
Step 2 : For i = 1 to n
         {  If C[i] = 0
         LC[p[i]] = LC [p[i]] + 1
         }
```

Example : Consider the tree shown in the Figure 7.

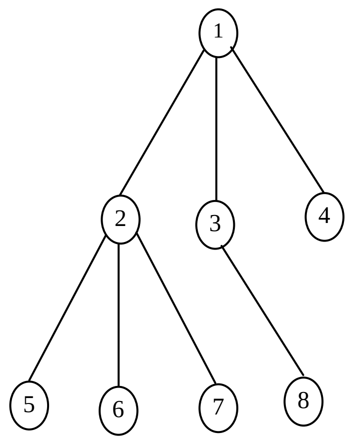

Figure 7. A Tree T'

Table 1 The node, parent, number of children and number of leaf children arrays of Figure 7.

i	1	2	3	4	5	6	7	8
p[i]	1	1	1	1	2	2	2	3
C[i]	3	3	1	0	0	0	0	0
LC[i]	1	3	1	0	0	0	0	0

The node *i*, for a parent *p[i]*, and the corresponding child nodes and number of leaf children are given in the Table 1.

Complexity Analysis

From the above algorithm, the time complexity is *O(n)*. This leads to the following theorem.

Theorem 11 : *In a tree T the sequential algorithm to count the number of children C[i] and number of leaf children LC[i] can be determined in linear time.*

7.3 To count the number of odd and even children of the root

Let *T* be the tree with root *r*. Let *i* be a child of *r*. If *i* is the root of the subtree, with odd number of children, *i* is called an odd child of *r*. Similarly we can define the even child of *r*. Let n_{odd} and n_{even} be the number of odd and even children of *r*. Assume that *T* is represented in the form of its parent array.

Algorithm FindOddAndEvenChildrenOfRoot (p, n, root)

1) Find C[i]
2) $n_{odd} = 0$
3) $n_{even} = 0$
4) for i = 1 to n
4a) If ((p[i] = root) and (i $\neq$ root))
if C[i] is odd
n_{even} ++
else
n_{odd} ++
end if
endif

From the above algorithm, the number of odd and even children can be determined in *O(n)* time.

7.4 Sequential algorithm for even trees of height 2

The sequential algorithm to check whether the tree is perfect IVDP is given below.

Algorithm EvenTwo( )

Input : i) A tree *T* of height 2
ii) Parent array *p[i]*
iii) Number of nodes *n*

Output : A Boolean value *result* which indicates whether *T* is perfect IVDP

1. Find n_{odd} = number of odd subtrees of r.
l = number of leaf children of r
2. If n_{odd} + l > 3 then result = false; exit
else proceed to step 3
3. For every subtree of r verify if it has a perfect IVDP matching.
If any of them doesn't have a perfect IVDP matching then
result = false; exit
If all the even subtrees have perfect IVDP matching
proceed to step 4.
4. result = True

Complexity Analysis

In a tree *T* of height 2, the sequential algorithm EvenTwo( ) checks whether the tree is IVDP which is determined as follows.
Step 1 can be found out in *O(n)* time. Steps 2, 3 check for the existence of IVDP matching. Step 4 gives the Boolean value result which is true or false. So the algorithm is determined in *O(n)* time.

7.5 Algorithm for odd tree of height 2

The sequential algorithm to determine the perfect IVDP matching is given below.

Algorithm OddTwo( )

Input *: i) A tree T of height 2*
ii) Parent array p(i)

iii) Number of nodes n

Output : *A Boolean value result which indicates whether T is perfect IVDP*

1. *Find n_{odd} = number of odd subtrees of r*
   l = number of leaf children of r
   t_5 = number of subtrees with 5 nodes
2. *If $t_5 > 1$ then result = false ; exit.*
3. *if ($t_5 = 1$) and ($n_{odd} + l$) > 3 then*
   result = false; exit.
4. *If ($t_5 = 0$) and ($n_{odd} + l$) > 4 then*
   result = false; exit
5. *For every subtree T_i verify if the subtree is perfect IVDP.*
   If any one is not perfect IVDP,
   then result = false; exit
   If all the subtrees are perfect IVDP,
   proceed to step 6.
6. *result = True*

Complexity Analysis

The complexity of the algorithm OddTwo( ) is same as the complexity of the algorithm EvenTwo( ).

8. Parallel Algorithms for Trees of height 2

In this section we develop a parallel implementation of the algorithm given in the previous section. Consider a tree of height 2 represented in the form of its parent array. To implement the algorithm in parallel machines, consider the two dimensional array, *child(i, j)* which consists of *n* rows and *n* columns. Where *n* is the number of nodes, defined as follows:

child (i, j) = 1 if i is the child of j and i $\neq$ j
= 0 otherwise

For Example consider the tree in Figure 8.

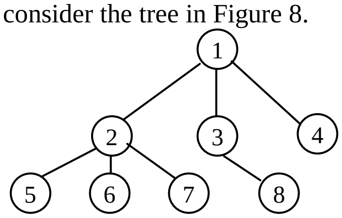

Figure 8. A Tree T

The parent relation of the above tree is given in Table 2

Table 2. Parent relation of tree T

i	1	2	3	4	5	6	7	8
p[i]	1	1	1	1	2	2	2	3

The two dimensional array *child(i, j)* is given in Table 3. The column sum gives the number of child nodes

Table 3. Number of child nodes for each i of Tree T in Figure 8

Child	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1								
2	1							
3	1							
4	1							
5		1						
6		1						
7		1						
8			1					
Column Sum	3	3	1					

The parallel algorithm to find the number of child nodes is given below.

8.1 Parallel Algorithm to count the number of children

Algorithm CountNumberOfChildren

Input : *Tree T*

Output : *C[i] Number of children for i*

1) *For i = 1 to n do in parallel*
   If i $\neq$ p[i]

1.1.1.1.1.1 Child [i, p[i]] = 1

2) *For each column j do in parallel*
   Find column sum in child matrix
   C[j] = Column sum of j^{th} column

Complexity Analysis

The above algorithm can be implemented in *O(logn)* time using $O(n^2)$ processors in EREW PRAM

Theorem 12 ***:*** *In a tree T with root r, the number of children for each node i can be determined in O(logn) time using $O(n^2)$ processors in EREW PRAM.*

8.2 Parallel Algorithm to find the number of leaf children

To find the number of leaf children in parallel, consider another two dimensional array

LChild [i, j] with *n* rows and *n* Columns, where
LChild (i, j) = 1 if i is a leaf child of j
= 0 otherwise

Example

Consider the tree given in the Figure 8. Also consider the following arrays *p[i]* and *C[i]* as given in Table 4.

Table 4. The child, Parent, number of children of Tree T in Figure 8.

i	1	2	3	4	5	6	7	8
p[i]	1	1	1	1	2	2	2	3
C[i]	3	3	1	0	0	0	0	0

The two dimensional array *LChild[i, j]* for this tree is given in Table 5. The column sum of the table gives the number of leaf children for each node *i*.

Table 5. Number of leaf children for each node i of Tree T in Figure 8.

LChild	1	2	3	4	5	6	7	8
1								
2								
3								
4	1							
5		1						
6		1						
7		1						
8			1					
Column sum	1	3	1					

The parallel algorithm to find out the number of leaf children is given below:

Algorithm CountLeafChildren
Input : *1) Tree T*
2) parent array p[i]
3) Array C[i] which gives the number of children of each node i
Output : *number of leaf children LC[i]*
1) For i = 1 to n do in parallel
if C[i] = 0 then
LChild [i, p [i]] = 1
2) For each column j
Find column sum in child matrix
LC[j] = Column sum of j^{th} column of LChild matrix

Complexity Analysis

The above algorithm can be implemented in *O(logn)* time using $O(n^2)$ processors in EREW PRAM which leads to the following theorem.

Theorem 13 : *In a tree T with root r the algorithm CountLeafChildren for each node i can be determined in O(logn) time using $O(n^2)$ processors in EREW PRAM.*

8.3 Algorithm to find the number of odd and Even subtrees of root.

Consider two arrays EVEN [i] and ODD [i]
EVEN [i] = 1 if *i* is a child of root and *i* is the root of an even subtree
= 0 otherwise
ODD [i] = 1 if *i* is a child of root and *i* is the root of an odd subtree
= 0 otherwise

Algorithm CountEvenOddSubtreesOfRoot *(p,n,root, C[i], LC[i])*
Input : *1) Tree T of height 2*
2) Parent array p[i]
3) Root of T
4) Array C[i] which gives the number of children for each node i
5) Array LC[i] which gives the number of leaf children for each node i
Output : *Number of odd subtrees and number of even subtrees*
1. For i = 1 to n do in parallel
1.1 If (p[i] = root and C[i] = odd) then
EVEN [i] = 1;
1.2 If (p[i] = root and C[i] = even) then
ODD [i] = 1
2. Find the sum of the arrays
n_{odd} = sum of the array ODD[i]
n_{even} = sum of the array EVEN [i]

Theorem 14 : *The algorithm CountEvenOddSubtreesOfRoot correctly determines the value of n_{odd} and n_{even}*

Complexity Analysis

Step1 can be implemented in *O(1)* time using *O(n)* processors. As sum of *n* numbers can be computed in *O(log n)* time using *O(n)* processors in EREW PRAM. Step 2 can be implemented in *O(log n)* time using *O(n)* processors. So, the above algorithm can be implemented in *O(log n)* time using *O(n)* processors in EREW PRAM. This leads to the following result.

Theorem 15 : *In a tree T of height 2 with root r, the number of odd subtrees n_{odd} and the number of even subtrees n_{even} can be determined in O(log n) time using O(n) processors in EREW PRAM.*

Par8.4 l Algorithm for even trees of height 2.

Algorithm EvenIVDPheight2
Input: *i) A tree T of height 2*
ii) Parent array p[i]
iii) Number of nodes n
Output *: A Boolean value result which indicates whether T is perfect IVDP*
1. Find n_{odd} = number of odd subtrees of r
l = number of leaf children of r

2. *If* $n_{odd} + l > 3$ *then result = false; exit*
   else proceed to step 3
3. *For each subtree* T_i *of r do in parallel*
   3.1 *Check if* T_i *is perfect IVDP. If* T_i *is not perfect IVDP,*
   result = false: exit.
4. *result = true*

Complexity Analysis

The above algorithm can be implemented in *O(logn)* time using $O(n^2)$ processors. This leads to the following result.

Theorem 16 : *If T is an even tree of height 2, we can verify if T has a perfect IVDP matching in O(log n) time using* $O(n^2)$ *processors in EREW PRAM.*

8.5 Parallel Algorithm for odd trees of height 2

Algorithm OddIVDPheight2

***Input**: i) A tree T of height 2*
ii) Parent array p[i]
iii) Number of nodes n

Output: *A Boolean value result which indicates whether T is perfect IVDP*

1. *Find*
   n_{odd} *= number of odd subtrees of r*
   l *= number of leaf children of r*
   t_5 *= number of subtrees with 5 nodes*
2. *If* $t_5 > 1$ *then result = false; exit*
3. *if* $(t_5=1)$ *and* $(n_{odd} + l) > 3$ *then*
   result = false; exit
4. *if* $(t_5 = 0)$ *and* $(n_{odd} + l) > 4$ *then*
   result = false; exit
5. *For every subtree* T_i *do in parallel*
   5.1 *Check if* T_i *is perfect IVDP. If* T_i *is not perfect IVDP,*
   result = false; exit
6. *result = true*

Complexity Analysis

The above algorithm can be implemented using *O(log n)* using $O(n^2)$ processors. This leads to the following result.

Theorem 17 : *If T is an odd tree of height 2, we can verify if T has perfect IVDP matching in O(log n) time using* $O(n^2)$ *processors in EREW PRAM.*

9. Conclusion and Open Problems

In the problem that we have discussed the matching paths are vertex disjoint. Consider the following applications of this problem. Suppose the vertices denote the computer terminals and the edges a connecting network. By a path matching we mean pairing computers in order to do a work in parallel. Since the works are done in parallel, we desire to have edge disjoint path matching. Since an edge (a wire connecting two nodes) can be used for limited data flow. In the edge disjoint path matching, if an edge is used by more then one matching path, the parallel processing operation may not be efficient.

The existence of IVDP matching for trees of arbitrary height may be studied. In each case sequential and parallel algorithms may be developed and the execution time may be determined.

References

[1] Aho.A.V. Hopcroft J.E. and Ullman J.D. The design and Analysis of computer Algorithms. Addison-wesley, 1974

[2] Brassard G. and Brately P. Algorithmics. Prentice Hall, 1988

[3] C. Berge and C. Chvatal, editors. Topics on perfect graphs, North-Holland, 1984. Annals of Discrete Mathematics (21)

[4] J.A. Bondy and U.S.R Murthy. Graph Theory with applications. North-Holland, NewYork 1976.

[5] H. J. Bandelt and H.M. Mulder. Distance-hereditary graphs. J. of combin. Theory series B. 41 : 182-208. 1986

[6] Date structures and Algorithms www. mpi-sb.mpg.de/~sanders/courses/algdat2/

[7] M. Habib and M.C Maurer. On the X – join decomposition for undirected graphs Disc. Math, 3 : 198 – 207, 1979.

[8] D.Helmbold and E.Mayr. Perfect graphs and parallel algorithms. In 1986 International conference on Parallel processing. pages 853 – 860. IEEE 1986.

[9] Joseph Ja Ja, Introduction to parallel algorithms, Addison Wesley (1992).

[10] A library of parallel Algorithms. www. cs.cmu.edu/`scandal/nesi/algorithms. 31 Aug 2002.

[11] Sun Wu and Udi Manber, Algorithms for generalized matching. Technical Report TR 88-89, Department of computer Science, University of Arizona (1998).

[12] [Sun Wu and Udi Manber, Path Matching Problems, Algorithmica(1992) 8:89 – 101

[13] C. Xavier and G. Arumugam Algorithms for parity path problems in some classes of graphs, computer science and Informatics Vol 24, No. 4, December 1994. pp50–54

[14] C. Xavier Sequential and parallel algorithms for some graph theoretic problems Ph.D Thesis

[15] C. Xavier and S.S. Iyengar, Introduction to Parallel algorithms, John Wiley & Sons Inc 1998.

Dr.G.Suganthi is a Reader in Computer Science in women's Christian College, Nagercoil, Kanyakumari District ,South India. She received her Ph.D in ComputerScience from Manonmanium Sundaranar University, Tirunelveli, South India in June 2004. Her research work is on Parallel Algorithms . She is guiding students to M.phil and Ph,D in various universities. Her research guidance areas are networking, Image processing, Secured data transfer etc. She has published five papers in international and three papers in national level.