RAIRO-Oper. Res. 50 (2016) 315–325
DOI: 10.1051/ro/2015024

RAIRO Operations Research
www.rairo-ro.org

LINEAR TIME ALGORITHMS TO SOLVE THE LINEAR ORDERING PROBLEM FOR ORIENTED TREE BASED GRAPHS

Alain Quilliot[1] and Djamal Rebaine[2]

Abstract. We present in this paper two simple linear algorithms that solve to optimality the linear ordering problem for unweighted tree based graphs *viz.* the oriented trees and the oriented divide-and-conquer graphs.

Mathematics Subject Classification. 9008.

Received May 18, 2015. Accepted June 1, 2015.

1. Introduction

Let $G = (V, E)$ be a graph, where V and E denote respectively the set of vertices and the set of arcs. The linear ordering problem for G consists of finding a one-to-one mapping σ from V to $\{1, \ldots, |V|\}$ that minimizes the following cost function

$$f(G, \sigma) = \sum_{(x,y)\in E} |\sigma(y) - \sigma(x)|. \tag{1.1}$$

The corresponding decision problem was first shown to be $\mathcal{NP}$-complete for arbitrary graphs [9]. This problem was also shown to remain $\mathcal{NP}$-hard even for some restricted classes of graphs such as interval graphs [5], and bipartite graphs [9]. However, polynomial time algorithms were also developed for other restricted graph classes such as trees [4], unit interval graphs [6], paths, cycles, complete graphs, complete bipartite graphs, and grid graphs [10]. A general survey on linear ordering problems is given in [3, 8, 10].

When the graphs are oriented, we first observe that the linear ordering σ to seek is such that if $(x, y) \in E$ then $\sigma(x) < \sigma(y)$. Moreover, cost function (1.1) simplifies into

$$f(G, \sigma) = \sum_{(x,y)\in E} \sigma(y) - \sigma(x). \tag{1.2}$$

The complexity status of the corresponding decision problem may be resumed as follows. It remains $\mathcal{NP}$-hard for the directed acyclic graphs as shown in [9]. The time complexity status gets however better for oriented trees

Keywords. Linear ordering, linear time algorithms, divide-and-conquer graphs, directed tree.

[1] Université Blaise Pascal, LIMOS, UMR CNRS 6158, BP 10125 Campus des Cézeaux, 63173 Aubière, France. alain.quilliot@isima.fr

[2] Université du Québec à Chicoutimi, Département d'informatique et mathématique, 555, Bld. de l'Université, Saguenay, Québec, Canada. djamal_rebaine@uqac.ca

Article published by EDP Sciences

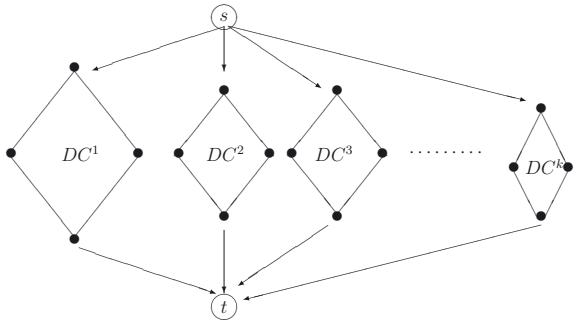

FIGURE 1. The structure of a divide-and-conquer graph.

as the corresponding problem was shown in [2] to be solvable in $O(n \log n)$, or for a subclass of serie-parallel graphs as the corresponding problem was shown in [1] to be solvable in $O(n^2)$.

In the present study we focus on two oriented special graphs connected to trees. The first one is simply the oriented tree for which we develop another optimal solution. By doing so, we achieve two goals. First, this solution is simple to understand and straightforward to implement. Second, the time complexity of the derived algorithm is linear thus optimal and improving by the same way the existing $O(n \log n)$-time algorithm. Let us, however, point out that the $O(n \log n)$ algorithm presented in [2] solves the weighted version of the problem we are considering here, that is to say the cost function to minimize is now $\sum_{(x,y)\in E} \omega(x,y)\,(\sigma(y) - \sigma(x))$, where $\omega(x,y)$ is the weight associated with arc (x,y). In our case, we assume that the weights associated with each arc are of unit length.

The second special graph we consider is the so called divide-and-conquer graph for which we develop a linear algorithm that solves the linear ordering problem. Let us mention that these graphs were introduced by Rayward-Smith and Clark [12] within the framework of scheduling theory. As pointed out in the literature, divide-and-conquer graphs naturally model the execution of the recursive control abstraction of divide-and-conquer algorithms. A great number of problems lend themselves to the divide-and-conquer strategy. These include binary search, minimum and maximum finding, quicksort and mergesort, the fast Fourier transform, and more (see for *e.g.* [7]. A divide-and-conquer graph can be defined recursively as follows[3]. A single node is a divide-and-conquer graph. Otherwise, let s and t be two distinct single nodes and $DC^1, \ldots, DC^k$, for some $k \geq 1$, disjoint divide-and-conquer graphs. Then, the graph denoted by $DC = (s, DC^1, \ldots, DC^k, t)$ in which node s precedes all $DC^1, \ldots, DC^k$ and node t is preceded by all $DC^1, \ldots, DC^k$, is also a divide-and-conquer graph. This type of graph is pictured by Figure 1.

The study of the complexity status of the linear ordering problem for divide-and-conquer graphs is motivated as follows. As we can see from Figure 1, a divide-and-conquer graph is nothing else than two oriented trees (an in-tree and an out-tree) that are somehow *"glued"* together correspondingly from their terminal nodes. Indeed, if we bend a DC-graph along the nodes which are located in the middle of the paths joining the two roots, s and t, then there is a one to one matching between the vertices of the two trees that form this DC-graph (this is the reason why these graphs are sometimes called mirror trees). Since the complexity status is known for oriented trees, then it seems natural to look at the complexity status of the combination of two trees as it is the case for DC-graphs.

This paper is organized as follows. In Section 2, after presenting notations and definitions for the forthcoming sections, we reformulate the cost function of (1.2) to suit our proof needs. In Section 3, we discuss the case of oriented trees. Section 4 is devoted to the case of divide-and-conquer graphs. Section 5 is our conclusion.

[3]This recursive definition is borrowed to Kubiak Potts [11].

2. Break reformulation of the linear ordering problem

In this section, we present basic notations and definitions that will be used in the remaining sections. Then, in order to meet some needs in the proof of Theorems 3.1 and 4.1, we reformulate the definition of the cost function (1.2) that is traditionally associated with the linear ordering problem.

Definition 2.1. If (x, y) is an arc of a graph, then y is the successor of x, denoted by $\mathrm{succ}(x)$, and x is a predecessor of y, denoted by $\mathrm{pred}(y)$.

Definition 2.2. Two vertices of a graph are considered as siblings if, and only if, they have the same predecessor.

Definition 2.3. If x is a vertex of V in a given graph G, then the set of successors of x is $\Gamma^+_{DC}(x) = \{y \in V \mid (x, y) \in E\}$, and the set of predecessors of x is $\Gamma^-_{DC}(x) = \{y \in V \mid (y, x) \in E\}$.

Definition 2.4. A linear ordering on a set V is a binary order relation σ (non reflexive and transitive) such that, for any couple (x, y) in V, $x \neq y$, we either have $x \ \sigma \ y$ or $y \ \sigma \ x$. If $A \subset V$, and σ is a linear ordering on V, then $\min(A, \sigma)$ and $\max(A, \sigma)$ denote the unique element x_0 of A such that for $x \neq x_0$ in A, we have $x_0 \ \sigma \ x$ and $x \ \sigma \ x_0$, respectively.

Clearly, such a linear ordering σ may be identified with the list obtained by sorting the vertices in increasing order according to σ. This makes possible to apply to linear orderings the standard algorithmic operators usually related to lists such as HEAD, TAIL, CONCAT (for concatenation), and the CONS operator which builds a list from its head and tail. When referring to a linear ordering, we indistinctly see it in the following sections as either a binary relation or a list.

We now introduce another way of formulating the cost function (1.2) associated traditionally with the linear ordering problem. Doing so makes it possible to use some counting arguments in the derivation of the optimality of the solution we are presenting in the next section.

Let σ be a linear ordering on vertex set V of $G = (V, E)$. We say that σ is compatible with G if for any arc $e = (x, y)$ in E we have $x \ \sigma \ y$. If σ is such a compatible linear ordering on V, then, for any arc $e = (x, y)$ and any vertex $z \in V$, we set $BE(e, z, \sigma) = 1$ if $x \ \sigma \ z \ \sigma \ y$, and $BE(e, z, \sigma) = 0$ otherwise. We say that $BE(e, z, \sigma)$ is the elementary break value of arc e by vertex z according to σ.

Let us note that it is straightforward to see that solving the Linear Ordering Problem on V is equivalent to seek a linear ordering σ, compatible with $G = (V, E)$, which minimizes

$$BG(G, \sigma) = \sum_{(x,y)\in E} |\{z \in V \mid x \ \sigma \ z \ \sigma \ y\}| . \tag{2.1}$$

Observe that the cost functions defined in (1.2) and (2.1) are equivalent since they only differ by $|E|$.

Definition 2.5. The global break of G according to σ is $BG(G, \sigma)$, and the break of arc e according to σ is $B(e, \sigma) = \sum_{z\in V} BE(e, z, \sigma)$.

The arguments we are going to develop, in the next sections to get the minimum value of (2.1), are related to the way the elementary break and break quantities are counted according to some compatible linear ordering for a given oriented graph. In addition, when it comes to algorithms, we will be using the fact that a linear ordering may also be viewed as a list. Therefore, in the following, we are going to deal with the Linear Ordering Problem on an oriented graph $G = (V, E)$, by considering that solving this problem corresponds to the problem of seeking a linear ordering σ on V, compatible with G, for which the value of $BG(G, \sigma)$ is as small as possible. The minimum value is denoted by $OPB(G)$. In our case, G stands for either an oriented tree or an oriented divide-and-conquer graph.

3. Linear ordering for oriented trees

In this section we discuss the oriented trees case. We first present a structural result, then an optimal algorithm along with its proof, and finally we discuss the implementation issues of this algorithm.

3.1. A structural result

Let us consider an oriented tree $T = (V, r, F)$ with arc set E, $n = N(r)$ the cardinality of V, r the root vertex, and $\{x_1, \ldots, x_s\}$ the sons of r. For any vertex x_i we denote by $n_i = N(x_i)$ the cardinality of $A(x_i)$, the subtree of T rooted at node x_i. We suppose that $\{x_1, \ldots, x_s\}$ are indexed in such a way that the sequence $(n_1, \ldots, n_s)$ is in non-decreasing order. The following is a recursive lower bound on the optimal global break value $OPB(T)$.

Theorem 3.1. *If T is an oriented tree T, then*

$$OPB(T) = \begin{cases} 0 & \text{if } T \text{ is reduced to its root,} \\ \sum_{i=1}^{s} OPB(A(x_i)) + \sum_{i=1}^{s} (s-i)n_i & \text{otherwise.} \end{cases}$$

Proof. Before proceeding any further, let us explain the idea behind the proof. Given a tree T, we are first going to make appear the existence of a linear ordering σ, compatible with T, such that

$$BG(T, \sigma) = \sum_{i=1}^{s} OPB(A(x_i)) + \sum_{i=1}^{s} (s-i)n_i.$$

This obviously implies

$$OPB(T) \leq \sum_{i=1}^{s} OPB(A(x_i)) + \sum_{i=1}^{s} (s-i)n_i. \tag{3.1}$$

Next, we consider an arbitrary linear ordering τ, compatible with T, and then prove

$$BG(T, \tau) \geq \sum_{i=1}^{s} OPB(A(x_i)) + \sum_{i=1}^{s} (s-i)n_i. \tag{3.2}$$

The result we are seeking then follows once inequalities (3.1) and (3.2) are both established.

In order to prove (3.2), we shall count the values $BE(e, z, \tau)$, $e \in E$, $z \in V$, of T, while distinguishing between two kinds of couples (e, z) as follows:

(L1): the two vertices of e and vertex z belong to the same set $r \cup A(x_i)$.
(L2): the two vertices of e belong to some set $r \cup A(x_i)$, and z belong to $A(x_j)$ with $i \neq j$.

Clearly, for a given index value i, and since we must have $r \; \tau \; x_i \; \tau \; x$ for any linear ordering τ and any vertex x in $A(x_i) - \{x_i\}$, the number of couples (e, z) such that $BE(e, z, \tau) = 1$, related to (L1), cannot be larger than the optimal value $OPB(A(x_i))$ of the Linear Ordering Problem restricted to $A(x_i)$. So, in order to prove

$$BG(T, \tau) \geq \sum_{i=1}^{s} OPB(A(x_i)) + \sum_{i=1}^{s} (s-i)n_i,$$

we shall show that $\sum_{i=1}^{s} (s-i)n_i$ is indeed a lower bound on the number of couples (e, z) related to (L1) such that $BE(e, z, \tau) = 1$.

We now go into the details of the proof. In order to do so, we first introduce some additional notations, related to a given linear ordering τ on the vertices of T, and we assume it is compatible with T. For i and j in $\{1,\ldots,s\}$, we set

$$BG_1(i,j,\tau) = \sum_{(x,y)\in A(x_i)} \sum_{z\in A(x_j)} BE((x,y),z,\tau). \tag{3.3}$$

$$BG_2(i,j,\tau) = \sum_{z\in A(x_j)} BE((r,x_i),z,\tau). \tag{3.4}$$

$$BG_3(i,j,\tau) = BG_1(i,j,\tau) + BG_2(i,j,\tau).$$

Let us observe that if $i=j$, then $BG_2(i,j,\tau)=0$, and $BG_1(i,j,\tau) \geq OPB(A(x_i))$. In addition,

$$\begin{aligned} BG(T,\tau) &= \sum_{i,j} BG_3(i,j,\tau) \\ &= \sum_{i\neq j} BG_3(i,j,\tau) + \sum_i BG_3(i,i,\tau) \\ &\geq \sum_{i\neq j} BG_3(i,j,\tau) + \sum_i OPB(A(x_i)). \end{aligned} \tag{3.5}$$

Clearly, from (3.3), (3.4) and (3.5), we may distinguish between the elementary break values $BE(e,z,\tau)$, $e\in E$, $z\in V$, according to the idea outlined in (L1) and (L2). Let us first prove

$$OPB(T) \leq \sum_{i=1}^{s} OPB(A(x_i)) + \sum_{i=1}^{s} (s-i)n_i.$$

In order to do so, we may assume that tree T is not reduced to its root, otherwise the result is trivial. Let us consider an optimal linear ordering σ_i for $A(x_i)$ for $i=1,\ldots,s$. Note that this part of the proof is not constructive and that we do not know here anything about the structure of those linear orderings σ_i, $i=1,\ldots,s$. However, the existence of i leads to the existence of a linear ordering σ for T derived from the concatenation of the linear orderings σ_i, $i=1,\ldots,s$, that is $\sigma = \mathrm{CONCAT}(r,\sigma_1,\ldots,\sigma_s)$. Linear ordering σ is clearly compatible with T. First, note that

$$BG_1(i,i,\sigma) = OPB(A(x_i)), \text{ for } i=1,\ldots,s.$$

We also have, for $i,j=1,\ldots,s$, $i<j$, $BG_1(i,j,\sigma)=0$, $BG_2(j,i,\sigma)=0$, and $BG_2(i,j,\sigma)=n_i$. Then, from (3.5), we may derive

$$\begin{aligned} BG(T,\sigma) &= \sum_{i,j} BG_3(i,j,\sigma) \\ &= \sum_{i\neq j} BG_3(i,j,\sigma) + \sum_i BG_3(i,i,\sigma) \\ &= \sum_{j<i} BG_3(i,j,\sigma) + \sum_i BG_3(i,i,\sigma) \\ &= \sum_{i=1}^{s} OPB(A(x_i)) + \sum_{i=1}^{s} (s-i)n_i. \end{aligned}$$

Thus, the result follows.

Let us now prove that

$$OPB(T) \geq \sum_{i=1}^{s} OPB(A(x_i)) + \sum_{i=1}^{s} (s-i)n_i.$$

To do so, we consider an arbitrary linear ordering τ for T, compatible with T, and i, j, $i \neq j$ in $\{1, \ldots, s\}$. The goal here is to find a lower bound for $BG_3(i,j,\tau) + BG_3(j,i,\tau)$. We may assume that $i < j$, which also means $n_i \leq n_j$. Let $z_i = \max(A(x_i), \tau)$ be the largest vertex according to τ for $A(x_i)$, and distinguish between the following two cases.

Case 1. $z_j \ \tau \ z_i$: Then, for any vertex z of $A(x_j)$, there exists at least one arc $e = (x, y)$ such that e is either (r, x_i) or an arc of $A(x_i)$, and $x \ \tau \ z \ \tau \ y$. Therefore,

$$BG_3(i,j,\tau) + BG_3(j,i,\tau) \geq BG_3(i,j,\tau) \geq |A(x_j)|.$$

Case 2. $z_i \ \tau \ z_j$: Proceeding as above we get

$$BG_3(i,j,\tau) + BG_3(j,i,\tau) \geq BG_3(i,j,\tau) \geq |A(x_i)|.$$

In any case, we have

$$\begin{aligned} BG_3(i,j,\tau) + BG_3(j,i,\tau) &\geq \inf(|A(x_i)|, |A(x_j)|) \\ &\geq \min(n_i, n_j) = n_i. \end{aligned} \tag{3.6}$$

Taking into account the fact that the sequence $(n_1, \ldots, n_i, \ldots, n_s)$ is in non-decreasing order, we derive from (3.6) that

$$\begin{aligned} \sum_{i \neq j} BG_3(i,j,\tau) &= \sum_{i<j} (BG_3(i,j,\tau) + BG_3(j,i,\tau)) \\ &\geq \sum_{i=1}^{s} (s-i)n_i. \end{aligned}$$

If we combine (3.6) with (3.5) we get

$$\begin{aligned} BG(T,\tau) &= \sum_{i,j} BG_3(i,j,\tau) \\ &= \sum_{i \neq j} BG_3(i,j,\tau) + \sum_{i} BG_3(i,i,\tau) \\ &\geq \sum_{i=1}^{s} (s-i)n_i + \sum_{i=1}^{s} OPB(A(x_i)). \end{aligned}$$

Therefore, the result of the converse inequality follows. This completes the proof of the theorem. □

3.2. A linear time algorithm

In this section, we transform the above structural result into an efficient algorithm that solves the Linear Ordering Problem for an oriented tree T. Clearly, the first part of the proof of Theorem 3.1 gives rise to the recursive algorithmic scheme Tree-Linear-Ordering function that generates an optimal linear ordering σ for T. We assume $\{x_1, \ldots, x_s\}$ the sons of root r are such that $|A(x_1)| \leq \cdots \leq |A(x_s)|$.

```
function Tree-Linear-Ordering (T, r)
{
  if (r is a leaf)
     Tree-Linear-Ordering ={r};
  else {
          Let (x_1, ..., x_s) be the sons of root r such that |A(x_1)| ≤ ··· ≤ |A(x_s)|;
          for i = 1 to s do
              σ_i = Tree-Linear-Ordering(A(x_i), x_i);
       } //end of if
  Tree-Linear-Ordering = CONCAT(r, σ_1, σ_2, ..., σ_s);
} // end of function
```

Corollary 3.2. *The Tree-Linear-Ordering function generates an optimal solution for the oriented tree T.*

Proof. The proof is derived through induction in an immediate way. Indeed, if we assume that the linear ordering σ_i = Tree-Linear-Ordering $(A(x_i))$ is optimal for $A(x_i)$, $i = 1, \ldots, s$, then the value associated with the linear ordering produced by Tree-Linear Ordering (T) is nothing else than the lower bound of Theorem 3.1. Thus, the result follows. □

Example 1. Let T be an oriented tree as shown in Figure 2. Function Tree-Linear-Ordering (T) generates the ordering $\sigma = \{1, 13, 12, 3, 9, 10, 11, 2, 8, 4, 5, 6, 7\}$ with $BG(T, \sigma) = 16$, while we have $f(T, \sigma) = 28$ as defined in (1.2).

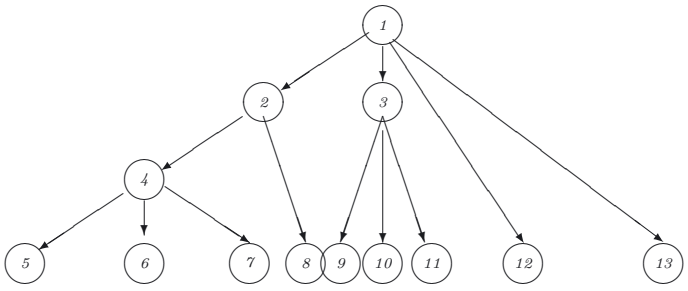

FIGURE 2. An oriented tree.

The remainder of this section is devoted to the implementation issues of the above recursive function, and we shall indeed show that it can be implemented to run in linear time.

3.3. Encoding the tree-linear-ordering function

In this section we discuss implementation issues of the above algorithm. As we will see it shortly, we show that this algorithm can be implemented to run in linear time. Tree T is encoded with two vectors FATHER and SONS indexed on the vertices of T in such a way that

- For vertex x of T, FATHER(x) denotes the father vertex of x.
- For vertex x of T, SONS(x) denotes the list of sons of x.

Let us first recall that a Depth First Search (DFS) process works by performing two classes of moves: PUSH and POP.

- PUSH moves: one arrives at vertex y while coming from its father x = FATHER$[y]$. We say that y is then visited. Such a move is performed exactly once for a given vertex y distinct from root r. When vertex y is

visited this way, all vertices z of SONS[x] located before y in the list SONS[x] have already been visited, as well as their heirs.

– POP moves: one starts from some vertex y, different from root r, and comes back to the father x = FATHER[y]. At this time, all heirs of y have already been visited.

The initialization of such a DFS process consists in a PUSH move which makes that root r is visited. The process ends up when root r is popped out. In the case when the number of operations, performed every time one arrives at some vertex (through either a PUSH or a POP move) is bounded by a constant, then the corresponding DFS process is clearly of linear time.

The following is the description in three steps of the implementation of Tree-Linear-Ordering function.

Three-Step Implementation

1. The first step builds, while exploring tree T through a DFS process, a vector VECT such that for any number $p = 1, \ldots, n$, VECT[p] is the list of vertices x of T such that $N(x) = p$. An auxiliary vector COUNTER is involved, with indexation on the vertex set V with values in $\{1, \ldots, n\}$. At each step of the DFS process, we need to consider two cases:
   (a) one arrives at y from x = FATHER[y] through a PUSH move. Then, COUNTER[y] = 1.
   (b) one arrives from y at x = FATHER[y] through a POP move. Then,
      (i) COUNTER[x] is incremented by COUNTER[y]: COUNTER[x] = COUNTER[x] + COUNTER[y];
      (ii) y is inserted at the head of VECT[COUNTER[y]]: VECT[COUNTER[x]] = CONS(y, VECT[COUNTER[y]]).
2. In the second step we derive from vector VECT another vector ORDER-SONS, which provides, for any vertex x, the list, ORDER-SONS[x], of the sons y of x, ordered in non-decreasing values of $N(y)$. ORDER-SONS[x] is initialized to NIL for every vertex x. We use a vector LAST, which provides, for any vertex x, a direct access to the last element of ORDER-SONS[x]. Note that LAST[x] is initialized, for any x, with a NIL value. This process is resumed as follows:

```
for p = 1 to n do
   for y in VECT[p]
      {
         x = FATHER[y];
         insert y at the end of list ORDER-SONS[x];
         update LAST[x];
      }
```

3. What we get at the end of the second step is another representation of T, through vector ORDER-SONS, which provides, for any vertex x, the list ORDER-SONS[x] of the sons y of x, ordered in non-decreasing values of $N(y)$. Then, as a third step, we apply the DFS process to T by using the ORDER-SONS vector instead of the SONS vector.

Theorem 3.3. *The Three-Step Implementation generates an optimal linear ordering for the oriented tree T in linear time.*

Proof. Step 1 and step 2 of the above Three-Step Implementation provides a representation of T through vector ORDER-SONS, containing, for any vertex x, the list ORDER-SONS[x] of the sons y of x. Clearly, the order in which the vertices of T are visited if we perform a DFS process, while using the ORDER-SONS vector instead of the SONS vector, corresponds to the linear ordering produced by the Tree-Linear-Ordering function.

The first and third steps of the above implementation are DFS processes. The number of elementary operations performed every time one arrives at some vertex is therefore bounded by a constant. It then follows that the two steps work in linear time. The second step also works in linear time since it scans once vertex set V, and, every time it deals with some vertex x, it performs a number of operations bounded by a constant. Observe that since we are using vector LAST, inserting y at the end of the current list ORDER-SONS[x] does not require scanning the list ORDER-SONS[x]. Therefore, the linearity of the time complexity of the implementation follows. □

4. The linear ordering for the divide-and-Conquer graph

In this section, the precedence graph we are considering is an oriented divide-and-conquer graph $DC = (s, DC^1, \ldots, DC^k, t)$ with arc set E, $n = |V|$ the number of vertices of graph DC. Subgraphs $\{DC^1, \ldots, DC^k\}$ denote the set of k components of graph DC, and s_i and t_i denotes the top node and the bottom node of component DC^i. In this section we prove that the following recursive algorithm generates an optimal linear ordering for graph DC. The idea of this algorithm is to put the top node and bottom node of the corresponding DC-graph in the first and last position, respectively. Then, in any order, the algorithm repeats recursively the same process for each of the different components composing the DC-graph.

function DC-Linear-Ordering (DC, s, t)
{
  if $(s == t)$
    return (s);
  else for $(i = 1$ to $k)$
    {
      Let $\Gamma^+(s) = \{s_1, \ldots, s_k\}$;
      Let $\Gamma^-(t) = \{t_1, \ldots, t_k\}$;
      σ_i = DC-Linear-Ordering(DC^i, s_i, t_i);
    } // end of for
  σ = CONCAT$(s, \sigma_1, \sigma_2, \ldots, \sigma_k, t)$;
  return (σ);
} // end of function

Example 2. Let DC be an oriented divide-and-conquer graph as shown in Figure 3. As long as we do not jump to another component if the previous one is not completed, while building the ordering, Function DC-Linear-Ordering (DC, 1,18) generates an optimal layout regardless of how the order of the components is handled. So, $\sigma = \{1, 3, 9, 10, 1, 17, 12, 13, 14, 2, 8, 4, 5, 6, 7, 15, 16, 18\}$ is one of the optimal ordering that can be generated by the above function. Its global breaking is $BG(T, \sigma) = 73$, while $f(DC, \sigma) = 99$ as defined in (1.2).

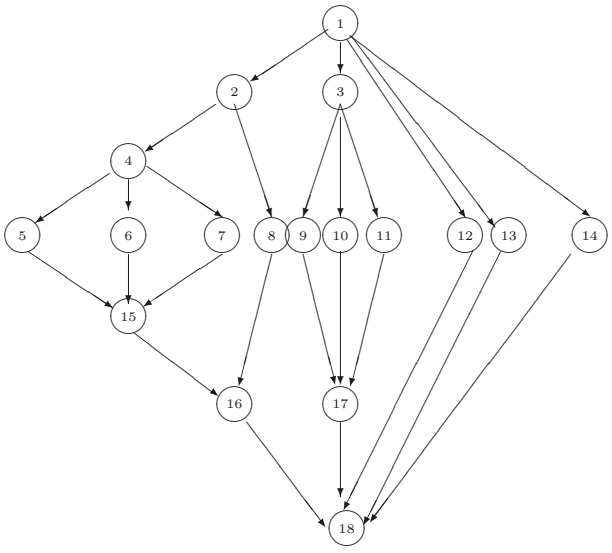

Figure 3. An example of a divide-and-conquer graph.

Theorem 4.1. *DC-Linear-Ordering function generates an optimal linear ordering for the divide-and conquer graph* $DC = (s, DC^1, \ldots, DC^k, t)$.

Proof. The idea behind the proof is to first establish a formula of the value of the layout generated by the above function. Then, we prove that this formula is nothing else than a lower bound on the value of the generated layout. The optimality of this layout thus follows immediately.

Let σ and σ_i, $i = 1, \ldots, k$ be respectively the global linear ordering and the partial linear orderings generated by the above function at each of its recursive calls. It is clear that they are all compatible with the DC-graph and within the components DC^i. Now, if we proceed by induction, then it is easy to get $BG(DC^i, \sigma_i) = OPB(DC^i)$. Then the linear orderings we get from the recursive calls lead to the following

$$BG(DC, \sigma) = \sum_{i=1}^{k} OPB(DC^i)) + (k-1) \sum_{i=1}^{k} |DC^i|. \tag{4.1}$$

It is interesting to note that (4.1) does not depend on the ordering of the DC^i components as the expression $(k-1)\sum_{i=1}^{k} |DC^i|$ is a constant.

We now prove that (4.1) is indeed a lower bound on any linear ordering for DC-graphs. To do so, we introduce additional notations, related to a given compatible linear ordering, say τ, on the vertices of the DC-graph. For i and j in $\{1, \ldots, k\}$, we set

$$BG_1(i, j, \tau) = \sum_{(x,y) \in DC^i} \sum_{z \in DC^j} BE((x, y), z, \tau). \tag{4.2}$$

$$BG_2(i, j, \tau) = \sum_{z \in DC^j} BE((s, s_i), z, \tau). \tag{4.3}$$

$$BG_3(i, j), \tau) = \sum_{z \in DC^j} BE((t_i, t), z, \tau). \tag{4.4}$$

$$BG_4(i, j, \tau) = BG_1(i, j, \tau) + BG_2(i, j, \tau).$$

Let us observe that if $i = j$, then $BG_2(i, j, \tau) = 0$, $BG_3(i, j, \tau) = 0$, $BG_1(i, j, \tau) \geq OPB(A(x_i))$. In addition,

$$\begin{aligned} BG(DC, \tau) &= \sum_{i,j} BG_3(i, j, \tau) \\ &= \sum_{i \neq j} BG_3(i, j, \tau) + \sum_{i} BG_3(i, i, \tau) \\ &\geq \sum_{i \neq j} BG_3(i, j, \tau) + \sum_{i} OPB(DC^i). \end{aligned} \tag{4.5}$$

Let us now consider two indices i and j, $i \neq j$, and set $\alpha = |DC^j| - BG_2(i, j, \tau) - BG_3(i, j, \tau)$. Every node $z \in DC^j$, with $t_i \ \tau \ z \ \tau \ s_i$, is such that there exists at least one arc $(x, y) \in DC^j$ with $x \ \tau \ z \ \tau \ y$. It then follows that each node $z \in DC^j$, with $t_i \ \tau \ z \ \tau \ s_i$, intervenes at least once in $\sum_{z \in DC^j} BE((x, y), z, \tau)$. Therefore we get $BG_1(i, j, \tau) \geq \alpha$. It follows

$$\begin{aligned} BG_4(DC, \sigma) &= BG_1(i, j, \tau) + BG_2(i, j, \tau) + BG_3(i, j) \\ &= \alpha. \end{aligned}$$

We then derive

$$\sum_{i \neq j} BG_4(i, j, \tau) \geq \sum_{i=1}^{k} \sum_{i \neq j} |DC^j| \geq (k-1) \sum_{i=1}^{k} |DC^i|.$$

Then we can conclude from (4.1) that

$$BG(DC, \tau) \geq BG(DC, \sigma).$$

This completes the proof. □

Regarding the time complexity of the above function, a simple Depth First Search strategy might be used to visit the *DC*-graph to generate a compatible linear ordering. Therefore, the linear time complexity of the above DC-Linear-Ordering function follows.

5. Conclusion

In this paper we presented two new optimal algorithms to solve the linear ordering problem for two special oriented graphs. Indeed, we first developed a simple optimal solution with an improved time complexity running in linear time for the oriented tree. Then, we developed a linear-time algorithm for divide-and-conquer graphs. We achieved these two results by introducing a new way of formulating the cost associated with such graphs. Doing so made it possible to use some counting arguments in the derivation of the optimality of the solutions we proposed. We believe that this approach may lead to new results on other restricted graphs such as the non-oriented divide-and-conquer graphs, and the generalization to the oriented divide-and-graph with weighted arcs.

Acknowledgements. This research was partially financed, for Djamal Rebaine, by the Natural Sciences and Engineering Research Council of Canada (NSERC), the *Ministère des Relations Internationales du Québec* (MRI), the *Consulat de France à Montréal*, and the *Université Blaise Pascal at Clermont-Ferrand* (France).

References

[1] S. Achouri, T. Bossart and A. Munier-Kordon, A polynomial algorithm for MINDSC on a subclass of serie parallel graphs. *RAIRO: OR* (2009) 145–156.

[2] D. Adolphson and T.C. Hu, Optimal Linear ordering. *SIAM J. Appl. Math.* **25** (1973) 403-423.

[3] I. Charon and O. Hudry, An updated survey on the linear ordering problem for weighted or unweighted tournaments. *Ann. Oper. Res.* **175** (2010) 107–158.

[4] F.R.K. Chung, On optimal linear arrangements of trees. *Comput. Math. Appl.* **10** (1984) 43–60.

[5] J. Cohen, F. Fomin, P. Heggernes, D. Kratsch and G. Kucherov, Optimal Linear Arrangement of Interval Graphs, *Proc. of MFCS'06 Proceedings of the 31st International Conference on Mathematical Foundations of Computer Science*. Springer-Verlag, Berlin, Heidelberg (2006) 267–279.

[6] D.G. Corneil, H. Kim, S. Natarajan, S. Olariu and A.P. Sprague, A simple linear time algorithm of unit interval graphs. *Inf. Process. Lett.* **55** (1995) 99–104.

[7] S. Dasgupta, Ch. Papadimitriou and U.V. Vazirani, Algorithms. McGrawHill (2006).

[8] J. Diaz, J. Petit and M. Serna, A survey of graph layout problems. *J. ACM Comput. Surveys* **349** (2002) 313–356.

[9] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-Completeness. Computer Press (1979).

[10] S.B. Horton, *The optimal linear arrangement problem: algorithms ans approximation*. Ph.D. thesis, Georgia Institute of Technology (1997).

[11] W. Kubiak, D. Rebaine and C. Potts, HLF is optimal for scheduling a divide and conquer graph on m identical parallel machines. *Discrete Optim.* **6** (2009) 79–91.

[12] V.J. Rayward-Smith and A.J. Clark, Scheduling theory applied to divide and conquer task systems on identical parallel machines, in *Conpar'88*, BCS Work shop Series. Edited by C.R. Jessehope, K.D. Reinartz. Cambridge University Press, Cambridge (1989).

[13] J. Valdes, R.E. Tarjan and E.L. Lawler, The recognition of series parallel digraphs. *SIAM J. Comput.* **11** (1982) 298–317.