Efficient parallel query processing by graph ranking*

Dariusz Dereniowski [†] **and Marek Kubale** [‡]

Department of Algorithms and System Modeling

Faculty of Electronics, Telecommunications and Informatics

Gdańsk University of Technology

Narutowicza 11/12, 80-952 Gdańsk, Poland

{deren,kubale}@eti.pg.gda.pl

Abstract. In this paper we deal with the problem of finding an optimal query execution plan in database systems. We improve the analysis of a polynomial-time approximation algorithm due to Makino et al. for designing query execution plans with almost optimal number of parallel steps. This algorithm is based on the concept of edge ranking of graphs. We use a new upper bound for the edge ranking number of a tree to derive a better worst-case performance guarantee for this algorithm. We also present some experimental results obtained during the tests of the algorithm on random graphs in order to compare the quality of both approximation ratios on average. Both theoretical analysis and experimental results indicate the superiority of our approach.

Keywords: edge ranking, performance guarantee, query processing, spanning tree

1. Introduction

A parallel query execution is a way for improving performance in large database systems. In general, there are three types of parallelism in database systems: *inter-query parallelism*, *inter-operator parallelism* and *intra-operator parallelism*. In all cases the goal is processor allocation but at different levels. In inter-query parallelism we have a set of queries (some of them can be executed concurrently) and we

have to find their parallel scheduling. If more than one processor is assigned to a query then the two other mentioned types of parallelism can be exploited as well. In inter-operator parallelism we schedule operations within a single query. In this paper we consider joins only as they are the most time consuming operations. Thus, we schedule join operations by assigning processors to joins and determining which operations can be computed independently. In the last type of parallel scheduling, intra-operator parallelism, we split one join operation among several processors in order to compute it efficiently. In this paper we present results which can be applied in inter-operator parallelism.

In the inter-operator parallelism we have two main problems to consider: join sequence selection and allocation of processors to joins. In this paper we deal with the first issue. Furthermore, the algorithm under consideration does not deal with the join execution costs. To avoid large intermediate relations we can use a technique described in [9], where the graph ranking approach can still be used.

Finding an optimal parallel *query execution plan* (QEP) is NP-hard. Thus, for complex queries the problem is intractable and most approaches are based on heuristics which generally have no formal analysis (such as worst-case bounds) on their optimality [11]. Makino, Uno and Ibaraki [8] gave an algorithm with a worst-case performance guarantee for solving this problem. In the sequel we will call their approach as the *MUI algorithm*. In the next section we describe the concept of edge ranking and and its application in the field of parallel query processing. Section 3 gives a new upper bound for the edge ranking number of a tree. In Section 4 we improve on the prior analysis of MUI. Finally, Section 5 gives some experimental results.

This paper is an improved version of [3] and has been extended to include the theoretical analysis from Sections 3 and 4.

2.   Preliminaries

The *join graph* of a query is defined as a simple graph G such that the vertices of G correspond to the base relations and two vertices are adjacent in G if the corresponding relations have to be joined during the execution of a query [5, 9, 10]. In this paper we assume that the join operations are executed in steps. Since we are interested in finding as short plans as possible so we assume that in each step several join operations can be performed in parallel. If a base relation is used in two different join operations (which means that in the query graph the edges which correspond to these operations are incident to a common vertex) then these joins cannot be computed in the same step. The scheduling of the join operations can be represented as a rooted tree T', where nodes of this tree represent the join operations and a particular join can be evaluated only if all its descendants have been already computed. Furthermore, if two nodes are unrelated in T' then the corresponding joins can be computed in parallel. The above data structure is known as the *operator tree*. For a more detailed description of operator trees see e.g. [1, 6].

If an operator tree T' has been computed, the join can be scheduled in such a way that the tree is processed in a bottom-up fashion – in the ith step all joins which correspond to the nodes in the ith level of T' are computed, where the last level is the root of T'. Note that the height of T' (the length of the longest path from the root to a leaf) is the number of parallel steps required to process the query. So, it is desirable to find a tree T' of small height. Note that if some relations x and y have been joined in the ith step (let z be the resulting intermediate relation) then in all joins in the later steps we use z instead of x and y.

In the following we assume that a join graph $G = (V(G), E(G))$ is given, where $|V(G)| = n$ and

$|E(G)| = m(G) = m$. We denote by $\Delta(G)$ the maximum vertex degree of G, in short the *degree* of G. An *edge k-ranking* of G is a function $c : E(G) \to \{1, \ldots, k\}$ such that each path between edges x, y satisfying $c(x) = c(y)$ contains an edge with a greater color. The smallest integer k such that there exists an edge k-ranking of G is the *edge ranking number* of G and is denoted by $\chi'_r(G)$. Makino et al. [8] introduced the *minimum edge ranking spanning tree* (MERST) problem. In their problem the goal is to find a spanning tree whose edge ranking number is as small as possible.

Spanning trees and their edge rankings can be used to find good QEPs (which in our case is equivalent to finding low height separator trees) in the following way. In order to complete the execution of the query, we have to compute joins which correspond to edges forming a spanning tree in the query graph G. Indeed, if C is a cycle with l edges in G then computing any $l - 1$ joins from that cycle results in an intermediate relation obtained by joining all base relations corresponding to the vertices of C. Furthermore, if we have selected some spanning tree T of G, then the task is to find an optimal parallel scheduling of joins corresponding to the edges of T. We find an edge ranking c of T and schedule the operations in such a way that for any two edges $e_1, e_2 \in E(T)$ the join corresponding to e_1 is a predecessor of the join corresponding to e_2 in an operator tree T' if and only if $c(e_1) > c(e_2)$ and the path connecting e_2 to e_1 in T does not contain an edge with a color greater than $c(e_1)$. So, the number of colors used by c is the height of the operator tree T' obtained in this way. Furthermore, this approach is optimal in the sense that if the spanning tree of the query graph is given then the height of an operator tree of minimum height equals to the edge ranking number of T [8]. This means that the difficulty lies in finding an appropriate spanning tree of the query graph.

Let us illustrate the above concepts by the following example. Consider a database query of the form

$$\text{Select * from } A, B, C, D, E, F \text{ where } A.x = C.x \text{ and } A.y = D.y \text{ and}$$
$$A.z = E.z \text{ and } A.t = B.t \text{ and } B.u = E.u \text{ and } B.v = F.v \ldots \tag{1}$$

Fig. 1(a) depicts the corresponding query graph G containing edges $\{A, B\}$, $\{A, C\}$, $\{A, D\}$, $\{A, E\}$, $\{B, E\}$, $\{B, F\}$ which correspond to the join operations that have to be computed in order to complete the above query. The edges of a spanning tree T of G are marked with heavy lines in Fig. 1(a). Note
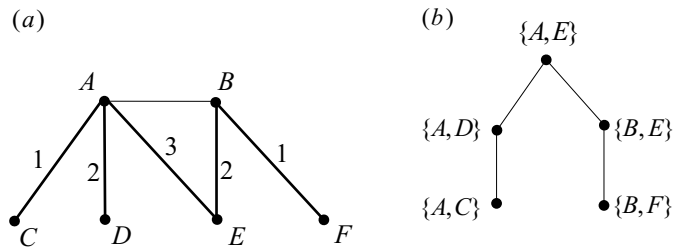

Figure 1.　(a) a query graph G, its spanning tree T and an edge ranking of T; (b) the corresponding operator tree T'

that the degree of T is minimum over degrees of all spanning trees of G. Furthermore, this is an optimal solution to the MERST problem for the graph G. We compute an edge ranking of T in order to create an operator tree. An example of such a coloring is given also in Fig. 1(a). The largest color used is equal to 3. Therefore, this query can be computed in parallel in three steps. The operator tree T' created

on the basis of this edge ranking is given in Fig. 1(b). Nodes of T' are labeled with the corresponding join operations. Finally, the operator tree can be used to create a parallel schedule as described above. In this case the join operations $\{A, C\}$ and $\{B, F\}$ are computed independently in the first step. In the second step we have operations $\{A, D\}$ and $\{B, E\}$. However, instead of using the relation A (B) we use the intermediate relation $\{A, C\}$ $(\{B, F\})$ created previously. In the final step we join the intermediate relations corresponding to the sons of the root. The first relation is the result of joining A, C, D and the other was obtained by joining B, E and F.

Though the MERST problem is polynomially solvable for threshold graphs [9], it is NP-hard for general graphs [8]. The MUI algorithm solves MERST for general graphs with sublinear approximation ratio

$$\frac{\min\{(\Delta^* - 1)\log n/\Delta^*, \Delta^* - 1\}}{\log(\Delta^* + 1) - 1},$$

where Δ^* is the degree of a spanning tree whose Δ is as small as possible, and $\log$ stands for $\log_2$ [8], while our approach leads to the bound

$$\frac{(\Delta^* + 1)\log_{\Delta^*+1}(n - 1)}{\max\{\Delta^*, \lceil \log n \rceil\}}.$$

3. The bound for trees

Star S_n is a tree with $n - 1$ leaves. Given a graph G, graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If $S \subseteq V(G)$ then the *induced subgraph* of G is defined as

$$G[S] = (S, \{e \in E(G) : e \subseteq S\}).$$

If $R \subseteq E(G)$, $S \subseteq V(G)$ then $G - R = (V(G), E(G) \setminus R)$, $G - S = G[V(G) \setminus S]$. We will write $G - e$ instead of $G - \{e\}$, where $e \in E(G)$. Graph is *connected* if there exists a path between every pair of vertices. H is a *connected component* of G if H is connected and H is no proper subgraph of a connected subgraph of G. We say that color i is *visible* for vertex v *under edge ranking* c if there exist $e \in E(G)$ such that $c(e) = i$ and a path connecting v and e such that all edges in this path get smaller colors than i. If the above path is empty then i is said to be *adjacent* to v.

Lemma 3.1. There exists an edge k-ranking c of tree T such that the edges colored with unique labels form a star.

Proof:
Let H_0 be the subgraph of T induced by the edges colored with unique labels under its edge ranking c_0. We prove that each connected component of H_0 is a star. Assume that H_0 contains a connected component H_0' which is not a star. Subgraph H_0' contains two nonadjacent edges e_1, e_2. Let e be any edge of the path connecting e_1 and e_2. We have $e \in E(H_0')$ because H_0' is connected. All three labels $c_0(e), c_0(e_1), c_0(e_2)$ are unique under c_0, thus we may shuffle them so that $c_0(e) > \max\{c_0(e_1), c_0(e_2)\}$. Thus, we can modify c_0 so that $c_0(e_1) = c_0(e_2)$ which leads to an optimal edge ranking of T using $k - 1$ colors – a contradiction. Similarly, we can prove that H_0 does not contain edges e_1, e_2, e_3 (which do not belong to the same connected component of H_0) such that e_2 belongs to the path connecting e_1 and e_2 in T.

Furthermore, at most one connected component of H_0 is not equal to S_2. Suppose, for a contradiction that H_0 contains two stars S_l, S_t such that $t \geq l > 2$. Without loss of generality we may assume that $c_0(E(S_l)) = \{k - l + 1, \ldots, k\}$ and $c(E(S_t)) = \{k - l - t + 1, \ldots, k - l\}$. If e is an edge connecting S_t and S_l then c' defined as

$$
\begin{aligned}
c'(E(S_l)) &= \{k - 2l + 1, \ldots, k - l\}, \\
c'(e) &= k - l + 1, \\
c'(x) &= c_0(x) \text{ otherwise}
\end{aligned}
$$

is an edge $(k - l + 1)$-ranking. By assumption, $l > 2$ which implies $k - l + 1 < k$, a contradiction.

Now we are ready to prove Lemma 3.1. Let us assume that H_0 contains q connected components. If $q = 1$ then we have already proved that this component is a star and we are done. If $q \geq 2$ then we can choose two nonadjacent edges $e_1, e_2 \in E(H_0)$ and we may assume that e_1, e_2 are labeled with two smallest colors under c_0 (as before we can shuffle unique labels). There exists an edge $e \in E(T) \setminus E(H_0)$ which belongs to the path connecting e_1 and e_2. We define edge ranking c_1 of T as follows

$$
\begin{aligned}
c_1|_{T - \{e, e_1, e_2\}} &= c_0|_{T - \{e, e_1, e_2\}}, \\
c_1(e_1) = c_1(e_2) &= \min\{c_0(e_1), c_0(e_2)\}, \\
c_1(e) &= \max\{c_0(e_1), c_0(e_2)\}.
\end{aligned}
$$

The subgraph of T induced by the edges with unique labels under c_1 is denoted by H_1. We repeat the above step for subgraphs $H_1, H_2, \ldots$. The inequality $|E(H_{i+1})| < |E(H_i)|$ implies that there exists an index $j \geq 0$ such that H_j contains one connected component. The corresponding edge ranking c_j is optimal, because it uses the same number of colors as $c_{j-1}, c_{j-2}, \ldots, c_0$ and by the discussion above the subgraph H_j is a star. Thus c_j is the desired edge ranking c. $\square$

Lemma 3.1 implies that each optimal edge ranking of a tree can be modified so that all edges labeled with unique colors are adjacent to some vertex denoted by r, i.e.

$$
\forall_{c(e), e \in E(T), r \notin e} \exists_{e' \neq e} \ c(e) = c(e'). \tag{2}
$$

We will refer to condition (2) as $C_1[c]$ in order to specify ranking c. We assume that all edge rankings considered in this section have the above property and all trees are rooted at vertex r. If $v \in V(T)$ then T_v is the tree rooted at vertex v, i.e.

$$
T_v = T[\{x : x \text{ is a descendant of } v\} \cup \{v\}].
$$

For each $v \in V(T)$ and edge ranking c of T we define set $L_c(v)$ of colors i such that $c(e) = i$ for some $e \in E(T_v)$ and all edges of the path connecting v and e have colors smaller than i, i.e. $L_c(v)$ is the set of visible colors for v assigned to the edges of T_v. We write $L_c(u) \leq_l L(v)$ if $L_c(u)$ is lexicographically smaller or equal to $L_c(v)$, i.e. $L_c(u) = L_c(v)$ or there exists element x such that

$$
x \in L_c(v) \text{ and } x \notin L_c(u) \text{ and } \forall_{y > x} y \in L_c(u) \iff x \in L_c(v).
$$

Relations $\geq_l, >_l, <_l$ are defined analogously.

The sons of r are denoted by $v_1, \ldots, v_p$. The biggest color which is assigned to more than one edge of T under c is denoted by $d(c)$. Let index $s(c)$ be defined so that all unique labels under c are assigned to edges $\{r, v_1\}, \ldots, \{r, v_{s(c)}\}$ (by Lemma 3.1 such a vertex r exists).

Lemma 3.2. ([2]) Given a rooted tree T, there exists an optimal edge ranking c of T such that for each $v \in V(T)$ and any edge ranking c' of T_v we have $L_c(v) \leq_l L_{c'}(v)$.

If

$$\overline{T_r} = T[V(T_{v_{s(c)+1}}) \cup \cdots \cup V(T_{v_p}) \cup \{r\}].$$

then by Lemma 3.2 we may assume that for any edge ranking c' of T

$$L_c(v_i) \leq_l L_{c'}(v_i), \quad i = 1, \ldots, s, \tag{3}$$

$$L_{c|\overline{T_r}}(r) \leq_l L_{c'|\overline{T_r}}(r). \tag{4}$$

Without loss of generality we can assume that the sons of v are numbered in such a way that

$$L_c(v_1) \geq_l L_c(v_2) \geq_l \ldots \geq_l L_c(v_{s(c)}). \tag{5}$$

We will refer to condition (3) as $C_2[c]$ in order to specify coloring c. Note that the values of $d(c)$ and $s(c)$ are unique, i.e. if c_1 and c_2 are two optimal edge rankings of T such that conditions $C_1[c_i], C_2[c_i], i = 1, 2$ are fulfilled then $d(c_1) = d(c_2)$ and $s(c_1) = s(c_2)$. Indeed, by $C_2[c_1]$ and $C_2[c_2]$, $L_{c_1}(v_1) = L_{c_2}(v_1)$ which implies

$$d(c_1) = \max(L_{c_1}(v_1)) = \max(L_{c_2}(v_1)) = d(c_2)$$

and equality $s(c_1) = s(c_2)$ follows from

$$s(c_1) = \chi'_r(T) - d(c_1) = \chi'_r(T) - d(c_2) = s(c_2).$$

Later on we will write d and s instead of $d(c)$ and $s(c)$. Fig. 2 explains the notation given above.
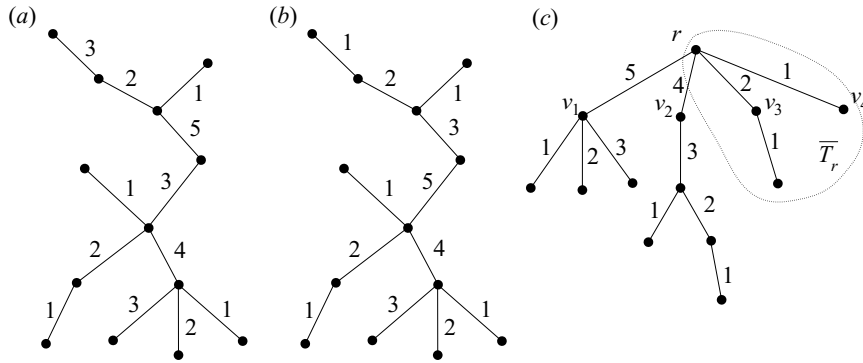

Figure 2. (*a*) an optimal edge ranking of tree T; (*b*) an optimal edge ranking of T such that unique colors form a star; (*c*) optimal edge ranking of rooted tree T satisfying (3), (4) and (5)

Fig. 2(*a*) depicts some optimal edge ranking of a tree T while Fig. 2(*b*) gives an edge ranking with property from Lemma 3.1. Finally, Fig. 2(*c*) shows the corresponding rooted tree and its edge ranking c satisfying properties (3), (4) and (5). For this edge ranking c we have $d = 3, s = 2, p = 4$, $L(v_1) = \{1, 2, 3\}, L(v_2) = \{3\}, L(v_3) = \{1\}, L(v_4) = \emptyset$ and $L(r) = \{1, 2, 4, 5\}$.

Lemma 3.3. If T is a tree such that $T_{v_i} \neq S_1, S_2$ for each $i = 1, \ldots, s$ and $\chi_r'(\overline{T_r}) < d$, then $\chi_r'(T_{v_i}) = d$ for each $i = 1, \ldots, s$.

Proof:
Let us assume that $\chi_r'(T_v) < d$ for some vertex v such that $c(\{r, v\}) > d$, where c is an optimal edge ranking of T. We can create ranking c' of T such that $c'(\{r, v\}) = d$ and $c'(e) = c(e)$ for each $e \neq \{r, v\}$. Edge ranking c' is legal and uses fewer colors than c, which leads to a contradiction. □

Lemma 3.4. Let c be an optimal edge ranking of T. If $d \in c(E(\overline{T_r}))$ then

$$|\chi_r'(T_{v_i}) - \chi_r'(T_{v_j})| \leq 1, \quad i, j \in \{1, \ldots, s\} \tag{6}$$

or

$$\chi_r'(T - e) = \chi_r'(T) \text{ for some } e \in E(T). \tag{7}$$

Proof:
Color d is used to label one edge of $\overline{T_r}$, because it is the biggest color in this subgraph. By (5), $d \in L_c(v_1)$. Let e be some edge adjacent to a leaf in T_{v_1}. Consider the tree $T - e$. Assuming $\chi_r'(T) \neq \chi_r'(T - e)$ (which means that condition (7) is not true) we will prove that (6) holds. We have $\chi_r'(T) = \chi_r'(T - e) + 1$, because e is a leaf and removing an edge from a graph cannot decrease the edge ranking number by more than 1. Let us assume, for a contradiction, that subtree $T_{v_s}, s > 2$ uses at most $d - 2$ colors. We know that $\chi_r'(T_{v_1} - e) \in \{d - 1, d\}$. Let c' be an optimal edge ranking of $T - e$. By Lemma 3.2 we can assume that

$$c|_{T_{v_i}} = c'|_{T_{v_i}} \text{ for each } i = 2, \ldots, p. \tag{8}$$

Observe, that $C_1[c']$ may not hold, but the only possible color which is unique under c' and is not adjacent to r is d. Note that $c'(\{r, v_1\}) \leq d$, because otherwise we can add edge e to subtree T_{v_1} and color T_{v_1} with d labels, which leads to an edge ranking that uses fewer colors than c used, a contradiction. Let us consider two cases.

Case 1: $d \in c'(E(T_{v_1} - e) \cup \{\{r, v_1\}\})$. Then, all edges $\{r, v_i\}, i = 2, \ldots, p$ such that $d \in c(E(T_{v_i}))$ are labeled with a color greater than d under c'. This means that some edge $\{r, v_j\}$, where $j > s$ gets a label greater than d because $d \in c(E(\overline{T_r}))$ and we assumed (4). Now we extend c' to an edge ranking c'' of T as follows

$$\begin{aligned} c''(\{r, v_1\}) &= \chi_r'(T), \\ c''(e) &= c(e), \text{ where } e \in E(T_{v_1}) \\ c''(e) &= c'(e) \text{ otherwise.} \end{aligned} \tag{9}$$

Edge ranking c'' is valid, it uses the same number of colors as c and $L_{c''}(r) <_l L_c(r)$, because in the case of coloring c'' this set does not contain label d. So,

$$L_{c''|_{\overline{T_r}}}(r) <_l L_{c|_{\overline{T_r}}}(r)$$

and by (4) this is a contradiction.

Case 2: $d \notin c'(E(T_{v_1} - e) \cup \{\{r, v_1\}\})$. Clearly, $d - 1 \in c'(T_{v_1})$. Thus, $c'(\{r, v_1\}) < d - 1$. Consider an edge ranking c'' satisfying (9). Note that color $d - 1$ is not visible from r in subtree $T - V(T_{v_1})$ under c''. By assumptions (3) and (4), color $d - 1$ is not visible from r under c and we can modify c by assigning color $d - 1$ to edge $\{r, v_s\}$, because $\max(L_c(v_s)) < d - 1$. The number of colors used by c decreases. Therefore our assumption that c is optimal is contradicted.

$\square$

Theorem 3.1. For each tree T with $n \geq 3$ vertices $\chi'_r(T) \leq B_2(T) = \lfloor \Delta \log_\Delta m(T) \rfloor$.

Proof:

We will use the following inequalities

$$a \log_a x \geq b \log_b x \qquad a \geq b \geq 1, x \geq 1, \tag{10}$$

$$a \log_a x \geq x \qquad a \geq x \geq 2. \tag{11}$$

We prove this theorem by induction on m. So, if we remove the edges colored with unique labels under c then we can apply the induction hypothesis for connected components $T_{v_1}, \ldots, T_{v_s}, \overline{T_r}$.

First, let us consider trees T such that color $c(\{r, v_i\})$ is unique and T_{v_i} is an empty graph or T_{v_i} contains one edge. If subtree T_{v_i} does not contain any edges then we can assume that all labels used to color tree T are incident to r. If this is not the case, i.e. color j is not adjacent to r then we can modify c such that $c(\{r, v_l\}) := c(\{r, v_i\})$, where $j \in c(E(T_{v_l}))$ and $c(\{r, v_i\}) := j$ (we know that j is not visible from r under c). In this way we obtain an edge ranking such that all colors are adjacent to r or all subtrees T_{v_i} are not empty. The second case will be considered later. Thus, we have

$$\begin{aligned}
\Delta(T) \log_{\Delta(T)} m(T) &\geq \Delta(T_\Delta) \log_{\Delta(T_\Delta)} m(T_\Delta) \tag{12} \\
&= \chi'_r(T),
\end{aligned}$$

where T_Δ is a subgraph of T, induced by r and all its neighbors. Inequality (12) follows from (10) and the facts that $m(T) \geq m(T_\Delta)$, $\Delta(T_\Delta) = \Delta(T)$ and $m(T_\Delta) = \Delta(T_\Delta)$.

Now, let us consider the case when T_v is nonempty, where v is a son of r. One label is not visible from vertex r, because each subtree T_v (v is a son of r) contains one edge and we consider edge rankings satisfying (3) and (4). Thus, we have that colors $2, \ldots, \chi'_r(T)$ are adjacent to r. Tree T contains $2(\chi'_r(T) - 1)$ edges. We have

$$\begin{aligned}
\Delta(T) \log_{\Delta(T)} m(T) &\geq (\chi'_r(T) - 1) \log_{\chi'_r(T) - 1} m(T) \\
&\geq (\chi'_r(T) - 1) \log_{\chi'_r(T) - 1} (2(\chi'_r(T) - 1)) \\
&= (\chi'_r(T) - 1) \log_{\chi'_r(T) - 1} 2 + \chi'_r(T) - 1 \\
&\geq \chi'_r(T).
\end{aligned}$$

The last inequality follows from (11).

Let T be any tree not considered in the above two cases. The induction hypothesis says that the theorem is true for all trees T' such that $m(T') < m(T)$.

Case 1*: Subtree $\overline{T_r}$ does not contain an edge labeled with color d.* Let

$$T' = T[V(T_{v_1}) \cup \cdots \cup V(T_{v_s}) \cup \{r\}].$$

By Lemma 3.3

$$d \in c(T_{v_i}) \text{ for each } i = 1, \ldots, s.$$

Let j be such an index that $m(T_{v_j}) = \min\{m(T_{v_i}) : i = 1, \ldots, s\}$. Then

$$
\begin{aligned}
\Delta(T) \log_{\Delta(T)}(m(T)) &\geq \Delta(T) \log_{\Delta(T)}(m(T')) \\
&= \Delta(T) \log_{\Delta(T)}(s + \sum_{i=1}^{s} m(T_{v_i})) \\
&\geq \Delta(T) \log_{\Delta(T)}(s\, m(T_{v_j})) \\
&\geq \Delta(T) \log_{\Delta(T)} s + \Delta(T) \log_{\Delta(T)}(m(T_{v_j})) \\
&\geq \Delta(T) \log_{\Delta(T)} s + \Delta(T_{v_j}) \log_{\Delta(T_{v_j})}(m(T_{v_j})) &(13) \\
&\geq s + \Delta(T_{v_j}) \log_{\Delta(T_{v_j})}(m(T_{v_j})) &(14) \\
&\geq s + \chi'_r(T_j), &(15) \\
&= \chi'_r(T).
\end{aligned}
$$

Inequality (13) follows from $\Delta(T) \geq \Delta(T_{v_j})$ and inequality (10). Conditions (11) as well as inequalities $\Delta(T) \geq s$ and $s \geq 2$ (d is not unique under edge ranking $c|_{E(T')}$) imply (14). In order to obtain (15) we applied the induction hypothesis for subgraph T_{v_j}, namely $\chi'_r(T_{v_j}) \leq \Delta(T_{v_j}) \log_{\Delta(T_{v_j})} m(T_{v_j})$.

Case 2*: Subtree $\overline{T_r}$ contains an edge labeled with d.* By Lemma 3.4, there exists an edge e such that $\chi'_r(T - e) = \chi'_r(T)$ or $|\chi'_r(T_{v_i}) - \chi'_r(T_{v_j})| \leq 1$ for $i, j = 1, \ldots, s$. In the first case we can apply the induction hypothesis for the subgraph $T - e$

$$\chi'_r(T) = \chi'_r(T - e) \leq \Delta(T - e) \log_{\Delta(T-e)}(m(T - e)).$$

The following equality holds

$$\Delta(T - e) \log_{\Delta(T-e)}(m(T - e)) \leq \Delta(T) \log_{\Delta(T)}(m(T)),$$

because $\Delta(T - e) \leq \Delta(T)$ and due to (10). If $\chi'_r(T - e) < \chi'_r(T)$ for all edges $e \in E(T)$ then by Lemma 3.4 $|\chi'_r(T_{v_i}) - \chi'_r(T_{v_j})| \leq 1$ for $i, j = 1, \ldots, s$, which implies

$$
\begin{aligned}
\Delta(T) \log_{\Delta(T)}(m(T)) &= \Delta(T) \log_{\Delta(T)}(s + m(\overline{T_r}) + \sum_{i=1}^{s} m(T_{v_i})) \\
&\geq \Delta(T) \log_{\Delta(T)}((s + 1)m(T_j)) \\
&= \Delta(T) \log_{\Delta(T)}(s + 1) + \Delta(T_{v_j}) \log_{\Delta(T_{v_j})}(m(T_{v_j})) \\
&\geq s + 1 + d - 1 &(16) \\
&= \chi'_r(T).
\end{aligned}
$$

The subgraph T_{v_j} above is defined as

$$m(T_{v_j}) = \min\{m(\overline{T_r}), m(T_{v_1}), \ldots, m(T_{v_s})\}.$$

Inequality (16) follows from (11) and the induction hypothesis applied to subgraph T_j:

$$\Delta(T_j) \log_{\Delta(T_{v_j})} m(T_{v_j}) \geq \chi_r'(T_{v_j}),$$

where $\chi_r'(T_{v_j}) \geq d - 1$ by (6).   □

4.   Worst-case analysis of MERST

The algorithm MUI can be described as follows:

1. Find a spanning tree T of G with $\Delta(T) \leq \Delta^* + 1$.

2. Find an optimal edge ranking of T.

In the first step we use a 1-absolute approximation polynomial-time algorithm given in [4]. The problem of finding an optimal edge ranking of a tree is polynomially solvable [7].

   In the following Δ^* is used to denote the maximum degree of a spanning tree of G whose maximum degree is as small as possible among all spanning trees of graph G.

Theorem 4.1. ([4]) There exists an $O(mn\alpha(m,n) \log n)$-time algorithm which for a given graph G finds its spanning tree T such that $\Delta(T) \leq \Delta^* + 1$, where α is the inverse Ackermann function.

Theorem 4.2. ([7]) There exists an $O(m)$-time algorithm which for a given tree finds its optimal edge ranking.

   Since the MUI algorithm is a combination of algorithms from Theorems 4.1 and 4.2, its running time is $O(mn\alpha(m,n) \log n)$.

Lemma 4.1. ([8]) For any tree T, $\chi_r'(T) \geq \max\{\Delta(T), \lceil \log n \rceil\}$.

Theorem 4.3. ([8]) If T is a tree then

$$\chi_r'(T) = \lceil \log n \rceil, \quad \text{if } \Delta(T) = 0, 1, 2$$

$$\chi_r'(T) \leq \frac{(\Delta(T) - 2) \log n}{\log \Delta(T) - 1} = B_1(T), \quad \text{if } \Delta(T) > 2.$$

   To compare bounds given in Theorems 3.1 and 4.3, let us consider the inequality $2 \log \Delta \leq \Delta$ which is true for $\Delta(T) = \Delta \geq 4$. Thus $\Delta \log \Delta - \Delta \leq \Delta \log \Delta - 2 \log \Delta$. Equivalently

$$\Delta \frac{\log n}{\log \Delta} \leq (\Delta - 2) \frac{\log n}{\log \Delta - 1}.$$

The above formula implies

$$\Delta \log_\Delta(n-1) \leq \frac{(\Delta - 2) \log n}{\log \Delta - 1}, \quad \Delta \geq 4. \tag{17}$$

Furthermore

$$B_1 - B_2 \geq (\frac{\Delta - 2}{\log \Delta - 1} - \frac{\Delta}{\log \Delta}) \log(n - 1)$$

and for $\Delta \geq 4$ we have

$$\frac{\Delta - 2}{\log \Delta - 1} - \frac{\Delta}{\log \Delta} > 0.$$

Thus, for a fixed value of $\Delta \geq 4$ we have $B_1 - B_2 = \Theta(\log n)$.

If T_o is an optimal solution to MERST for G and T is a tree produced by the algorithm from Theorem 4.1, then on the basis of Lemma 4.1 and Theorem 4.3 we have [8]

$$
\begin{aligned}
\frac{\chi'_r(T)}{\chi'_r(T_o)} &\leq \frac{(\Delta^* - 1) \log n / (\log(\Delta^* + 1) - 1)}{\max\{\Delta^*, \lceil \log n \rceil\}} \\
&\leq \frac{\min\{(\Delta^* - 1) \log n / \Delta^*, \Delta^* - 1\}}{\log(\Delta^* + 1) - 1} = R_1(G).
\end{aligned}
\tag{18}
$$

On the basis of Lemma 4.1 and Theorem 3.1

$$\frac{\chi'_r(T)}{\chi'_r(T_o)} \leq \frac{(\Delta^* + 1) \log_{\Delta^* + 1}(n - 1)}{\max\{\Delta^*, \lceil \log n \rceil\}} = R_2(G). \tag{19}$$

By (17), if $\Delta^* + 1 \geq 4$ then the approximation ratio R_2 is better than R_1, i.e. $R_2(G) \leq R_1(G)$, when $\Delta^* \geq 3$. Note, that for a fixed value of Δ^* or for $\Delta^* = \Theta(n)$ we have $R_2 = \Theta(1)$.

5. Experimental results

Below we present some experimental results gained while testing MUI on random graphs. Each chart shows the values of parameters as functions of n, the size of graph. We generated graphs for each $n \in \{50, 60, \ldots, 150\}$. Each point denoted in the chart is the average of 100 values. All graphs created during the tests were connected. Note that if G is not connected then solving the MERST problem for G reduces to solving the problem for all connected components of G separately. Then a solution for G is a spanning forest and its edge ranking number is the maximum over the edge ranking numbers of the trees forming the forest. In order to create a random graph we constructed its random spanning tree and then, according to the graph density g, more edges were added. This means that for small g (i.e. $g = 0.005$) the random graphs were trees. Fig. 3 shows the results of the computer experiments.

Fig. 3(a) depicts the relation between the edge ranking number of a tree and the bounds B_1 and B_2. The degree of all trees generated was equal to 10. Two types of inaccuracies are included in the approximation ratios considered in this paper. The first follows from the fact that the MUI algorithm uses a heuristic for finding a minimum degree spanning tree. However, it is possible to create examples of graphs for which a spanning tree with minimum vertex degree is not the one with minimum edge ranking number. Then, there is some inaccuracy in bounding the edge ranking number when only n and Δ are given. Thus, the purpose of comparing the edge ranking number with its bounds is that the quality of the approximation ratio directly depends on the bounds used.

The second test is presented in Fig. 3(b), where we tested only the first part of MUI, i.e. we generated random graphs G with $\Delta(G) = 10$ and densities $g = 0.01, 0.05, 0.1$, respectively. We used only small values of g, because as the experiments show, already for these values of g, the degree of the spanning
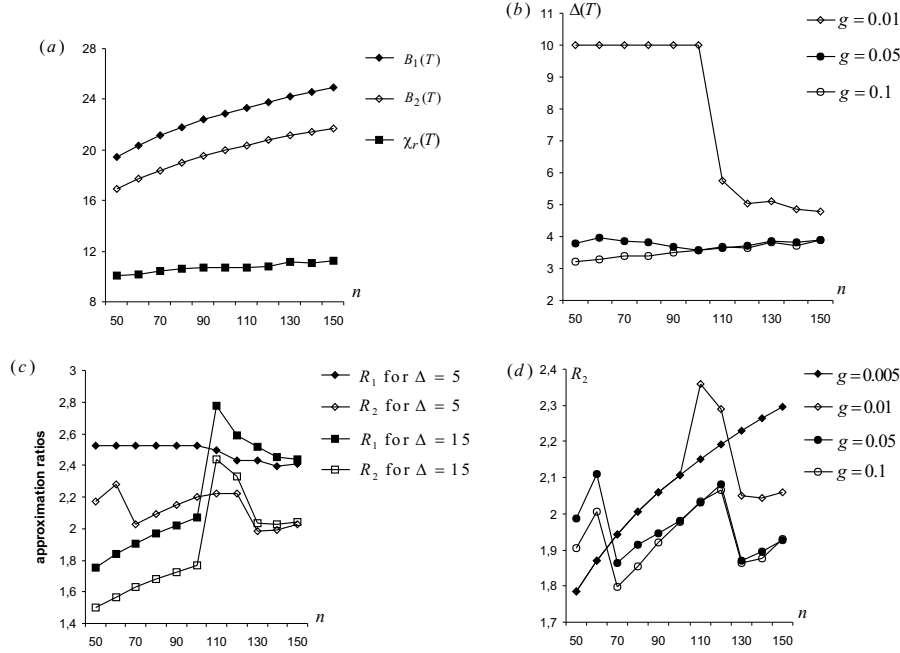
Figure 3.   Experimental results

trees is very small. From the theoretical analysis it follows that once we obtain a spanning tree which is a path, the problem has been solved optimally, so we want to avoid such cases. Some values of the first function for $g = 0.05$ are equal to 10, because random graphs obtained in these cases were trees.

Fig. 3(c) compares approximation ratios R_1 and R_2. In this test the graph density $g = 0.01$. We used a small value of g, because as the previous experiment shows, for larger g the average over degrees of T was smaller than 4 and in that case we would rather use the theoretical analysis from Section 4 to compare R_1 and R_2.

Fig. 3(d) presents the approximation ratio R_2. In this test, for each value of n and g we generated random graphs G with $\Delta(G) = 10$. Then, we computed R_2 for the spanning tree obtained in the first part of MUI. As before, each point is the average of 100 values (i.e. 100 different random graphs G were created). The first function, because of the small value of g, presents R_2 for trees with $\Delta = 10$. For smaller values of n and $g = 0.01$ graphs G are also trees. Larger graphs G are not acyclic. The approximation ratio R_2 is clearly not monotonic in Δ^*, which explains why the second function presented in Fig. 3(d) can get smaller and bigger values than the first one.

The experimental results presented in this section compare both bounds in the average case. The mean value of R_1 (taken over all test cases) was 15% bigger than the mean value of R_2.

References

[1] Chaudhuri, S.: An overview of query optimization in relational systems, *Proc. PODS,* Seattle (WA), USA, 1998.

[2] de la Torre, P., Greenlaw, R., Schäffer, A. A.: Optimal edge ranking of trees in polynomial time, *Algorithmica*, **13**, 1995, 529–618.

[3] Dereniowski, D., Kubale, M.: Parallel query processing and edge ranking of graphs, *Proceedings of the Sixth International Conference on Parallel Processing and Applied Mathematics,* Poznan, Poland, September 2005, (to appear).

[4] Furer, M., Raghavachari, B.: Approximating the minimum-degree Steiner tree to within one of optimal, *J. Algorithms*, **17**, 1994, 409–423.

[5] Ibaraki, T., Kameda, T.: On the optimal nesting order for computing N-relational joins, *ACM Transactions on Database Systems*, **9**, 1984, 482–502.

[6] Kremer, M., Gryz, J.: A survey of query optimization in parallel databases, *Technical Report* CS-1999-04 York University, 1999.

[7] Lam, T. W., Yue, F. L.: Optimal edge ranking of trees in linear time, *Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998, 436–445.

[8] Makino, K., Uno, Y., Ibaraki, T.: On minimum edge ranking spanning trees, *J. Algorithms*, **38**, 2001, 411–437.

[9] Makino, K., Uno, Y., Ibaraki, T.: Minimum edge ranking spanning trees of threshold graphs, *LNCS*, **2518**, 2002, 428–440.

[10] Ullman, J. D.: *Principles of Database and Knowledge-Base Systems*, Vol. 1. Computer Science Press, Maryland, 1990.

[11] Yu, P. S., Chen, M.-S., Wolf, J. L., Turek, J.: Parallel query processing, in: N.R. Adam, B.K. Bhargava, editors, *Advanced Database Systems LNCS,* Springer-Verlag, **759**, 1993.