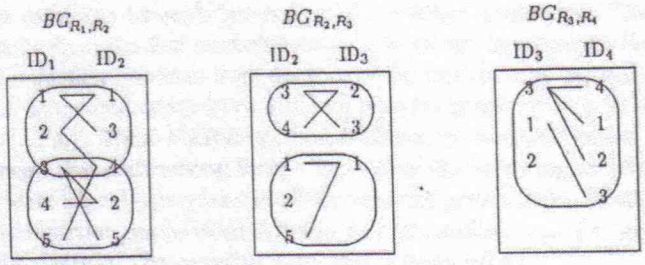
identifiers from ID_{i-1}. Therefore, during the forward transmission it maybe necessary to transmit, conceptually messages of the form $(id_i, a_i, \{page(id_i)\})$. If a sort-merge is the method of choice for the join to be performed at S_i, then indeed the messages will have the above format. On the other hand, if pipelining is employed, then this information will be transmitted with some slight overhead, i.e., a message of the form $(id_i, a_i, page(id_i))$ needs to be sent for every distinct page at S_i containing tuple identifiers from ID_{i-1} connected to id_i. For example, assume that id_i is connected with two identifiers id_{i-1} and id'_{i-1} and that the corresponding tuples in $BG_{R_{i-1}.R_i}$ are stored on pages p_1 and p_2 respectively. If pipelining is employed, S_i will send the messages (id_i, a_i, p_1) and (id_i, a_i, p_2). Irrespective of the join method used at S_{i+1}, the graph at this site will contain both tuples (id_i, id_{i+1}, p_1) and (id_i, id_{i+1}, p_2) if id_{i+1} is connected to id_i.

The forward reduction phase starts at each site with the construction of the relation $BG_{R_{i-1}.R_i}$ $(ID_{i-1}, ID_i, PAGE(ID_{i-1}))$ and its storage on secondary storage. Next, each page of the bipartite graph is read in order to construct the forward messages, and the page is written back to storage after it has been sorted on attribute ID_i. The sort step is done in order to facilitate the backward reduction and the graph traversal at the query site. We observe here that this sorting step does not incur any additional overhead in terms of I/Os, since the graph had to be stored first on secondary storage in order to obtain the corresponding page numbers necessary for the transmission.

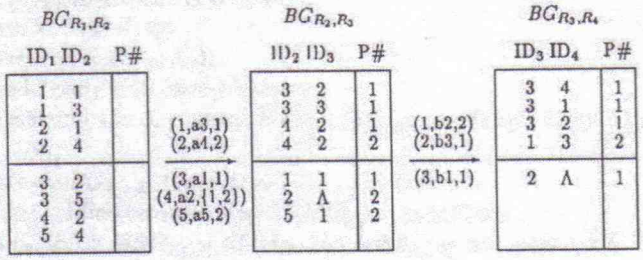
The backward reduction phase at S_i identifies as before all the identifiers id_{i-1} that are not connected to any id_i's and constructs messages of the form $(id_{i-1}, page(id_{i-1}))$. An additional step needs to be performed now, before the actual backward transmission can start, namely all the messages to be sent need to be sorted first by $page(id_{i-1})$. This step is necessary in order to guarantee that at the receiving site, S_{i-1}, each page will be read and written back to storage only once. However, the total amount of memory required at a given site for its outgoing messages is quite small, and this sort can be performed in main memory. In addition, since at the receiving site S_{i-1} each page is sorted on id_{i-1}, a binary search can be performed in order to identify the tuples in the relation $BG_{R_{i-2}.R_{i-1}}$ that need to be eliminated. In our current implementation, these tuples are marked as deleted.

After the backward reduction is completed the size of the bipartite graphs, if compaction were to be executed, could be small enough so that all bipartite graphs fit into main memory at the query site. If this is the case, then Step 4 of the PIPE_CHQ algorithm can be applied the same way, by just ignoring the page numbers. Otherwise, this step is modified and proceeds by interleaving the transmission of bipartite graphs with the construction of temporary relations holding the implicit join tuples. Let us denote by $R'_i \bowtie R'_{i+1} \cdots \bowtie R'_{i+j}$ the implicit join of $R_i, R_{i+1}, \ldots, R_{i+j}$, i.e., the projection of the join on $(ID_i, ID_{i+1}, \ldots, ID_{i+j})$. First, site S_n sends its graph to the query site where the graph is sorted according to $page(id_{n-1})$. Then, we proceed in increasing page number order by joining the tuples in this graph with those in the graph at S_{n-1}. Note that this implicit join can be performed by sending to the query site the pages in the graph of S_{n-1} one at a time and then performing a binary search in the corresponding subgraph of $BG_{R_{n-2}.R_{n-1}}$. After finding the implicit join $R'_{n-1} \bowtie R'_n$ we sort this temporary relation according to $page(id_{n-2})$ and continue in a similar fashion to find the implicit join $R'_{n-2} \bowtie R'_{n-1} \bowtie R'_n$. We repeat this procedure until we obtain $R'_1 \bowtie R'_2 \cdots \bowtie R'_n$.

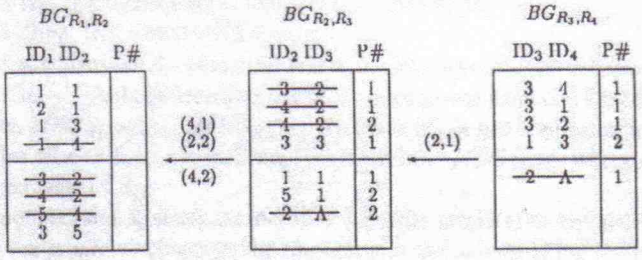
Example 2. Let us consider again the chain query $R_1 \bowtie_B R_2 \cdots \bowtie_D R_4$. We will assume that at each site sort-merge has been selected as the optimal join method in our join sequence. The bipartite graphs to be constructed at each site together with their corresponding partitioning into subgraphs (pages) are shown in figure 3(a). Note that the underlying database used for this example is slightly modified from the one given in Table 1. We have deliberately changed the database in order to illustrate some details about the partitioning of the bipartite graphs. Figure 3(b) shows the results of the modified PIPE_CHQ algorithm after the execution of the forward reduction. Note that at site S_2, the page numbers in the relation BG_{R_1,R_2} are left null. During forward reduction, S_2 sends the augmented messages (1,a3,1), (2,a4,2), etc. to S_3. Since the graph BG_{R_1,R_2} has crossing edges (2,4) and (5,4), the



3a: Graphs to be constructed



3b: Graphs with page numbers in forward reduction



3c: Graphs with page numbers in backward reduction

Figure 3. Graphs for disk-based systems.