## Bucket Elimination

Bucket Elimination method represents the query as a join-expression tree which has set of attributes as nodes, in order to describe an evaluation order to the join operation. More specifically, join operations are evaluated from the lower to the higher one and projection operations are applied for this specific evaluation as soon as possible. The name of this technique is due to the fact that it creates a bucket per variable, which investigates. To clarify how this method works let’s take an order of query attributes, for which we build buckets, one per attribute. Inside we put all the functions which mention attribute/node . The follow algorithm describes the general idea of the technique.

Table 1 General Bucket Elimination Algorithm

|  |
| --- |
| 1. Process buckets backwards according the given order of nodes 2. Eliminate the node in the bucket from subsequent computation 3. Place the computed function in the bucket into the highest variable/node in its scope bucket   Given order: A, C, B, F, D, G  Backwards: G, D, F, B, C, A   1. Eliminate ONE bucket per time (from the nth bucket to the 1st) |

### Preparation for Bucket Elimination Algorithm execution

In order to execute the algorithm we have to define an order of nodes as it is mentioned in 1st step. Thus we find the MCS order of the vertices. MCS is the acronym of Maximum Cardinality Search and MCS algorithm chooses iteratively vertices. The way in which this algorithm produce the output order is explained as “the next (unnumbered) vertex to be chosen: the vertex with the most already numbered neighbors”. The follow algorithm describes how the algorithm works.

Table MCS Algorithm

|  |
| --- |
| Input: Graph G(V, E)  Output: Ordering of Graph G   1. Assign the label to all vertices //each vertex has a label 2. *for* to *do* //the graph has vertices 3. pick the unnumbered vertex with the label   //During the first pick, algorithm chooses a random vertex, since all vertices have label equal to   1. //This action assigns to vertex the position 2. *for each* unnumbered vertex adjacent to vertex 3. increment value by 4. *end for* 5. *end for* |

### Bucket Elimination Algorithm execution

Given the MCS order of the vertices we create a bucket for each vertex and now we consider these buckets iteratively, from the last to the first and eliminate the latest bucket every time (bottom up method). The Bucket of the vertex stores:

1. Every , which contains vertex .
2. Every regarding vertex

During these iterations, inside the considering bucket there are several relations, where there is a specific attribute, iconic for this bucket, in all these relations. For all these relations, their join is computed and attributes are projected out, if they do not belong to the target schema, which we want in the end. For this reason, each bucket has live vertices. These are:

1. Every vertex, which appear inside the s of the bucket.
2. Every vertex, which appear inside section of bucket’s queries

So, as mentioned above, we process these buckets in descending order according to MCS order in the following way.

A bucket is processed by this way:

|  |
| --- |
| 1. Create the following query: all the of the bucket   all and of the bucket  them  approach conditions   1. Find the destination of the above query between the live vertices of the bucket. This means that we have to find the maximal vertex in query’s section. With the term maximal, we indicate the vertex with the position which is closer to the first one according the MCS order. 2. Move the query, which is created during the first step, to chosen vertex’s bucket and then eliminate the processed bucket.   When we reach the final bucket we will not execute the 2nd and the 3rd step, but we will change the section, with only one in case or the free vertices in case. |

The result can be empty, and in that case the result of the whole query is empty. If it is not empty, we place the resulting relation in the highest bucket, which has an attribute in the joined relation that is iconic for the bucket. If this process terminates with there still being a relation in the last bucket, we have found the non-empty result of our query.