Chapter 4

Algorithm OOB: Obtaining Optimal Backbones

In order to compare to the backbone set obtained by algorithm SB with the optimal one, by utilizing the concept of A* search[11], we design algorithm OOB (standing for Obtaining Optimal Backbones) which is able to determine the optimal backbones. Determining optimal backbones can be represented by a search problem based on state transition. In fact, algorithm OOB is a best-first state transition search. The search made by algorithm OOB can be represented by a solution tree where each node is associated with a state of backbones. Figure 4.1 shows part of an example state transition search which corresponds to the case of selecting backbones among four example query trees shown in Figure 3.1. Starting from the root in the level one, algorithm OOB generates nodes in level i to explore the possible backbone selections with the number of backbones being $i-1$. As shown in Figure 4.1, a node in the solution tree contains one possible backbone and non-backbone sets (i.e., B and NB). The goal node contains the solution of optimal backbone and non-backbone set with the purpose of minimizing the number of messages involved in all query trees. The search for goal node starts from the root node of solution tree. The root node represents that we do not select any backbones yet. The child nodes of a node is generated as we select one of the query tree

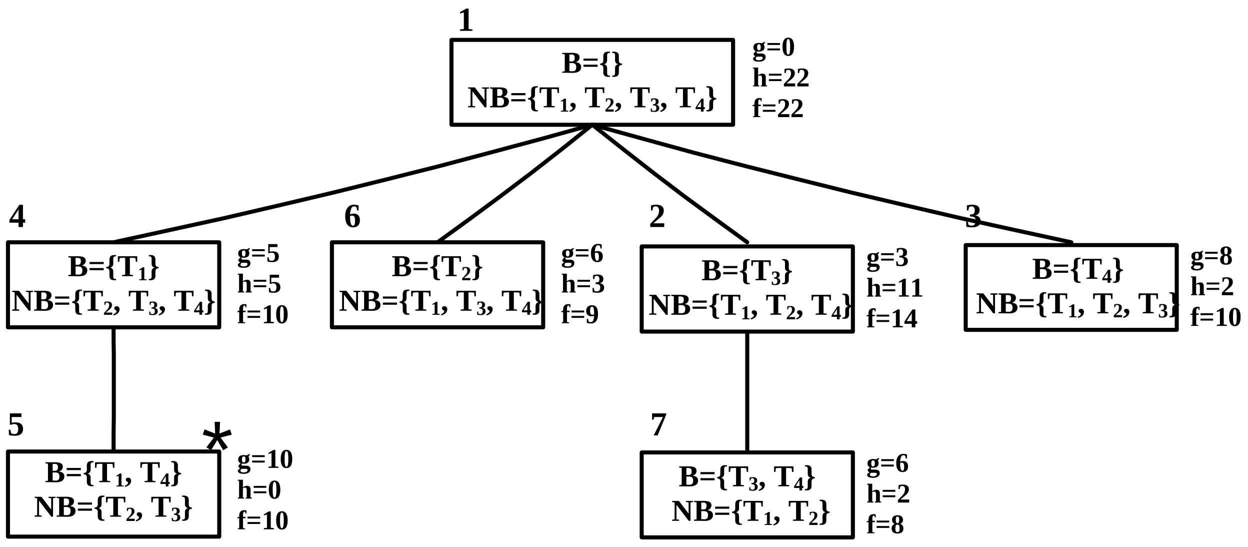

Figure 4.1: The solution tree for seleting backbones from four example queries

$T_j \in NB$ to join B, where $j > i$ for each T_i in B and $BR(T_j) > 0$. For example, assume that the current node has $B = \{T_3\}$ and $NB = \{T_1, T_2, T_4\}$. One child node is formed as we select T_4 into B instead of selecting T_1 or T_2. This design is to avoid generating the same child nodes from different tree nodes. Furthermore, when selecting backbone tree from non-backbone set, only those trees that have $BR(T_j) > 0$ should be considered. With a given a solution tree, the problem of selecting backbones becomes a search problem based on state transition in which a node has the optimal backbone set and non-backbone set with the minimal cost is to be found (i.e., the goal node).

Similarly to an A* search, the search in algorithm OOB is controlled by an evaluation function $f(\cdot)$. Denote P as the current reached node, $P.B$ as its backbone set and $P.NB$ as its non-backbone set. Node P chosen for further searching its child nodes is the one which has the largest value of $f(\cdot)$ among all generated nodes which have not been visited so far. Algorithm OOB stops the search until the goal node is found. Therefore the function $f(P)$ that guides the search in algorithm OOB consists of two components: the cost of reaching node P from the root, i.e., $g(P)$, and the expected cost of arriving at the goal node from node P, i.e., $h(P)$. Accordingly, $f(P) = g(P) + h(P)$. The function $g(P)$ is defined as the number of messages reduced by existing backbones $P.B$, and formulated as follows:

$$g(P) = \sum_{T_j \in P.NB} R(T_j, P.B)$$

For example, consider a node with its B=$\{T_4\}$ and $NB = \{T_1, T_2, T_3\}$,which represents the node marked by 3, in Figure 4.1. By exploring the procedure developed in Section 3.2, we could derive $g(P) = R(T_1, P.B) + R(T_2, P.B) + R(T_3, P.B) = 1 + 5 + 2 = 8$.

As pointed out earlier, the function $h(P)$ is the estimated cost from node P to the goal node. In order to obtain the maximal number of messages reduced from selecting some query trees in $P.NB$, $h(P)$ is formulated as follows:

$$h(P) = \sum_{T_j \in P.NB} max(\delta(T_j), 0).$$

Intuitively, we intend to select those query trees with their corresponding $\delta(T_j)$ larger than zero. Thus, h(P) is used to estimate as the sum of all those query trees with their values of backbone gains larger than zero to make sure the success of reaching the goal node. Consider the node (i.e., node marked by 3 in Figure 4.1) in the above example, $h(\cdot)$ of this node can be calculated by $max(BR(T_1), 0) + max(BR(T_2), 0) + max(BR(T_3), 0) = max(2, 0) + max(-2, 0) + max(-2, 0) = 2 + 0 + 0 = 2$.

In light of the evaluation function, algorithm OOB can be outlined in Algorithm 2.

In the beginning of algorithm OOB (from line 1 to line 2), the root node is initially set up with the backbone set being empty and the non-backbone set as the set of a given query trees. Then, algorithm OOB constructs the heap data structure and inserts the root node into the heap (from line 3 to line 5). The heap data structure is a maximal heap that removes a node with the maximal value of the evaluation function $f(\cdot)$. $Best\_P$ is used to represent the node with the optimal solution so far. Algorithm OOB removes the node i from the heap (line 6

Algorithm 2 OOB: Obtaining Optimal Backbones

```
Input: A set of query trees $AT = \{T_1, ..., T_n\}$
Output: A set of backbones $B$
1: $P.NB \Leftarrow AT$
2: $P.B \Leftarrow \phi$
3: Construct a heap by the value of the evaluation function $f(\cdot)$
4: $Best\_P \Leftarrow P$
5: Insert $P$ to the heap
6: while heap is not empty do
7:    Remove node P from the heap
8:    if $g(P) > g(Best\_P)$ then
9:       $Best\_P \Leftarrow P$
10:   end if
11:   if $f(P) > g(Best\_P)$ then
12:      Expand the child nodes of $P$ and insert them into the heap
13:   end if
14: end while
15: $B \Leftarrow Best\_P.B$
16: Arrange for each non-backbones in $NB$ to take the partial results of backbones in $B$
```

to line 7) and check whether $Best\_P$ should be adjusted by node i or not by comparing the value of function $g(\cdot)$ of node i and that of $Best\_P$ (line 8 to line 10). If the value of $f(\cdot)$ of node i is larger that of $g(Best\_P)$, node i will expands the descendants of node i. At the same time, these descendants of node i are inserted into the heap (line 12). Algorithm OOB expands the nodes iteratively until the heap is empty and $Best\_P$ will contain the optimal solution for the backbone set and the non-backbone set.

Consider the four query trees in Figure as an example, where the solution tree in Figure 4.1 and the number next to each node represents the sequence of node expansion. Algorithm OOB starts with the root node and then expands the descendants. The values of functions $g(\cdot)$, $h(\cdot)$, and $f(\cdot)$ are calculated accordingly, as shown in Table 4.1. Consider the root node, where no backbones have been selected (i.e., $B = \phi$ and $NB = \{T_1, T_2, T_3, T_4\}$). Since $B = \phi$, there is no messages reduced and thus $g(root) = 0$. The corresponding backbone gain values are calculated, which is shown in Table 4.1. Consequently, we could have $h(root) = 5 + 6 + 3 + 8 = 22$. Hence, $f(root) = g(root) + h(root) = 0 + 22 = 22$. Then, the root node is inserted into the heap. Since the heap is not empty, the root node will be removed

Search Order	Backbone and Non-backbone set	Backbone Gain				g	h	f
		T_1	T_2	T_3	T_4			
1	$B = \{\}, NB = \{T_1, T_2, T_3, T_4\}$	5	6	3	8	0	22	22
2	$B = \{T_3\}, NB = \{T_1, T_2, T_4\}$	2	6	-	3	3	11	14
3	$B = \{T_4\}, NB = \{T_1, T_2, T_3\}$	2	-2	-1	-	8	2	10
4	$B = \{T_1\}, NB = \{T_2, T_3, T_4\}$	-	0	0	5	5	5	10
5	$B = \{T_1, T_4\}, NB = \{T_2, T_3\}$	-	-7	-2	-	10	0	10
6	$B = \{T_2\}, NB = \{T_1, T_3, T_4\}$	-1	-	3	0	6	3	9
7	$B = \{T_3, T_4\}, NB = \{T_1, T_2\}$	2	-2	-	-	6	2	8

Table 4.1: The execution scenarios of algorithm OPT.

from the heap and expand the child nodes of the root node. The child nodes are the nodes on the level 2 of the solution trees. The corresponding values of the evaluation functions of these child nodes will be calculated and these child nodes are then inserted into the heap. Then, the node with the maximal value of $f(\cdot)$ will be removed (i.e., node 2 with $B = \{T_3\}$ and $NB = \{T_1, T_2, T_4\}$. Since g(node 2) is larger than the $g(Best\_P)$, node 2 will be marked as $Best\_P$. Since f(node 2) is larger than the $g(Best\_P)$, node 2 expands the child node and inserts its child nodes into the heap. Algorithm OOB stops when the heap is empty. It can be seen that the goal node of this solution tree in node 5 which has the B=$\{T_1, T_4\}$ and NB=$\{T_2$, $T_3\}$, which is indeed the same as the one obtained by algorithm SB. As the number of query trees increases, the number of nodes in the solution trees increases drastically. It is worth mentioning that with the proper design of the guide function, algorithm OOB is able to determine the optimal solution efficiently.

Theorem 1: $f(P)$ provides an upper bound of the number of messages reduced if we select backbones by a solution of one child node of P.

Proof: Denote $h^*(P)$ as the maximal extra number of messages reduced if some query trees in $P.NB$ are selected as backbones. In light of $h^*(P)$, the maximal number of messages reduced after we select backbones by a solution of one child node of P is $g(P)+h^*(P)$. On the other hand, $h(P)$ is the sum of all backbone gains for query trees in $P.NB$. In other words, $h(P)$ estimates $h^*(P)$ by assuming that all query trees in $P.NB$ can be backbones and can further reduce the total number of messages spent for all queries. In fact, only some query trees in $P.NB$ have been selected as backbones. Thus, we will update the backbone gains for other query trees in $P.NB$ and the backbone gains of other query trees will decrease after some backbones are selected. Consequently, we have $h(P) > h^*(P)$ and this theorem follows. **Q.E.D.**