**Developing B&B-Based Algorithms to Solve the Resource-Sharing and Scheduling Problem (RSSP)**

**Introduction**

In 1999, the first B&B-based algorithm was developed to find an optimal solution for the resource-sharing and scheduling problem (RSSP) (Samaddar et al., 1999). In the present work, a new B&B algorithm was developed to find an optimal solution for the RSSP. This chapter first reviews the general principles for B&B algorithms and the criteria for evaluating their performance according to Ibaraki (1987). Next, several options for implementing the principles to solve the RSSP are discussed. Finally, a theoretical and empirical comparison is made between the algorithms discussed in the RSSP literature and the proposed algorithms.

**1. Principles and Performance Evaluation of a B&B Algorithm**

One way of finding an optimal solution for a problem is to lay out all possible solutions, and select a solution with an optimal value (a minimal value for a minimization problem, or a maximal value for a maximization problem). This method is called the Enumeration Method, and it is usually considered inefficient, since in most situations, problems have tremendously large solution spaces. For this reason, there is great interest in developing more efficient methods that allow reduction of the solution spaces in which the optional solution to the problem lies. A B&B algorithm represents a collection of methods intended to achieve this objective. Without limiting generality, we'll assume that from now on, we are solving a minimization problem.

**1.1.** **Principles of B&B Algorithms**

In developing a B&B algorithm to solve the problem, it is important to focus on the following three principles:

**Methods for Branching / Decomposition**

Optimization problem  is in most cases a major problem that can be represented as a finite set of smaller partial sub-problems , where each sub-problem in the set can be represented as another set of sub-problems, . In B&B, each partial sub-problem is considered a *branch*, and the process by which the problem is broken down into sub-problems and the hierarchical relation between them determined, is called *decomposition*.

Using decomposition, a tree of all  solution options can be obtained; or in other words, a directed tree , where  is a set of nodes and  is a set of directed arcs with . Each node  represents a sub-problem, and arc  indicates that sub-problem  is created as a result of branching from sub-problem . In order for a search algorithm to find an optimal solution by examining a small portion of the branches, each sub-problem must have the following two properties:

* If an optimal solution is attained for  , then there is no need to continue to decompose  into additional sub-problems.
* If the conclusion is reached that  does not contain an optimal solution for , then can be bounded (truncated).

The more sub-problems that have both these properties, the more efficient the search is. The *bounding operations* performed during the search use the principles of these properties.

**Methods for Pruning and Bounding Options**

As previously stated, bounding operations are performed to prune or reduce search options in a tree. There are two basic methods for implementing the bounding operations: the lower bound test and the dominance test.

The lower bound test is designed to check whether the optional solution value of a sub-problem  is greater than or equal to the best value of  obtained in the search process up to the test point, i.e. . A sub-problem that meets this condition is bounded, meaning there is no need to continue decomposing it into additional sub-problems. Since during the search the value ofis unknown and the number of sub-problems to be examined could be considerable, instead of calculating , the lower bound value  of is calculated (a calculation of much lower complexity). Function  must have a value that is lower than or equal to the value of the optimal solution of a sub-problem, that is, , , and exactly equal to the value of the optimal solution on leaves, , . When  is a set of sub-problems that could not be decomposed, and were fully solved, it means they are leaves on the search tree. We also define that if there are two bounds such that ,  then bound  is considered *tighter* to the solution than .

The dominance test checks whether there is a (dominated) sub-problem that necessarily does not contain a better solution than another (dominating) sub-problem. Dominated problems can be bounded during the search process.

Domination ratio  is a binary ratio defined on a set of sub-problems. Ratio  means that sub-problem  dominates sub-problem . Domination ratio  means that  and for , it means that for every offspring of  which is  there is an offspring of ,  such that  exists.

It is important to note that in order for a tighter lower bound to result in more effective search results, it must be *consistent* with domination ratios. That is, if exists, then . Otherwise, it cannot be said regarding algorithm performance with respect to tightening the lower bound.

For the purposes of a general description of B&B algorithms, let us define:

- A set of sub-problems formulated to the current stage of the search.

- A set of active sub-problems, where sub-problem  which was not formulated, or not tested, is called an active sub-problem.

 - Lowest upper bound.

 - Solution with upper bound .

 - Upper bound function for which , .  , . is the value of a possible solution for sub-problem .

Examination of the next active sub-problem is performed according to search strategy such that . Figure 1 illustrates the general structure of a B&B algorithm.



**Figure 1 - General Structure of a B&B Algorithm**

|  |
| --- |
| התחל – Start  כאשר – Where / for  A9:  כן (right branch)- Yes – There is no solution possible for Po  (לא (left branch) – No – The optimal solution is at Z. The value of the optimal solution is z. |

**Search Strategy**

B&B algorithms always find an optimal solution in a finite number of steps regardless of the search strategy . But the effectiveness of the algorithm and the memory space required to execute it, are very much dependent on . Following is an overview of several of the most common search strategies.

**Heuristic Search**

We'll define  as a heuristic value that can be calculated for . The value of  represents an estimate of the true value, and it can be calculated quickly. The heuristic search function  is  , i.e. each time, a node  is selected with the lowest value of .

When it is possible to calculate a lower bound and an upper bound of a sub-problem, then it is possible to use a heuristic function of the following form , .

Heuristic function  is called non-misleading when for every pair of nodes  and  the following is true:

If  then .

If then .

**Best Bound Search**

With this search, it is possible to see a heuristic search, where the heuristic function in this case is. The search function by lower bound  is , that is, a node  is selected each time with the value of the lowest .

**Depth First Search**

Given a set of nodes, it is possible to define a subset of these nodes with maximum depth,  .

Depth first search function  is , i.e. each time a node  is selected with the value of the lowest .

**Breadth First Search**

Given a set of nodes , it is possible to define a subset of nodes with the lowest depth,  .

The depth first search function  is , that is, a node is selected each time with the value of the lowest .

**1.2. Evaluating B&B Algorithm Performance**

B&B-based algorithm performance can be evaluated according to the amount of work invested until the first possible solution to the problem is found, the amount of work invested until an optimal solution is found, and the total amount of work invested until the end of the algorithm run. Since the search process uses a strategy based on node classification, another index that can be used to evaluate algorithm performance is the maximum value of . In view of this, it is possible to define the following performance indices:

- Number of  that were decomposed before finding the first possible solution.

- Number of  that were decomposed before finding an optimal solution.

- Maximal value of  throughout the algorithm run.

- Total number of  that were decomposed during the algorithm run. The total time required to run a search strategy is equal to .

Table 1 shows a comparison of performance indices for each search strategy. It indicates that the heuristic and best lower bound strategies have the advantage in two performance indices: amount of work invested until an optimal solution is found, and total work invested until the end of the algorithm run. The depth first search strategy has the advantage in two other indices: amount of work invested to find the first solution to the problem, and the maximum number of active nodes during the algorithm run. The breadth first search strategy has no advantages in performance indices in comparison with other strategies, but Ibaraki (1987) stresses that this strategy could have advantages like those of other strategies if there were good domination ratios.

**Table 1 - Comparison of Search Strategies with Respect to Performance Indices**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Search Strategy | Performance Indices | | | |
|  |  |  |  |
| Heuristic | + | - | + | - |
| Best lower bound | + | - | + | - |
| Depth first | - | + | - | + |
| Breadth first | - | - | - | - |

The value of  depends on the value of the lower bound. It has been proven that  when . It is important to choose a suitable representation of the tree to cause the lower boundto be as precise as possible. In addition, based on simulation runs conducted by Ibaraki (1987), it is possible to see the effect of  and  on values  and  in the heuristic search. If there is a desire to reach good results quickly, effort must be invested in development of . On the other hand, if we are interested in finding an optimal solution to a problem, most efforts should be spent in developing . In a depth first search,  and  have a similar effect on values  and .

**2. B&B Principle Implementation Options for Solving an RSSP**

The objective of this section is to show the development of an algorithm for finding the first optimal solution for an RSSP (which we assumed to be a minimization problem). This algorithm would take full advantage of what is known about B&B algorithm performance from the literature and knowledge on RSSP; and thus reach several application possibilities to be examined in an empirical experiment. Therefore, fundamental decisions for the purpose of implementing B&B algorithms are based on the following decision criteria:

* Required memory space – Although increasing computer memory is not a significant problem nowadays, we would still like the search algorithm to use as little computer memory as possible. A large problem that requires much running time consumes a lot of computer memory, and this can be a limitation to finding an optimal solution to the problem.
* The amount of work required to obtain a first solution – Obtaining a first solution is a very important point in the search process since starting from that point, other options can be bounded.
* First solution quality – The closer the first solution is to an optimal solution, the greater the chances of bounding more options.
* Maximal use of a lower bound – Lower bound properties must be considered in order to achieve a tighter lower bound for a solution that enables bounding off more options.
* Implementation complexity – The level of implementation complexity affects not only the amount of work required for computation, but also the complexity of programming and testing required for it; as well as the reliability of the system that executes the algorithm.
* The amount of work required through completion of the algorithm run – Only at the end of the algorithm run is it possible to determine whether the solution found is an optimal solution.

**2.1. Branch Decomposition**

The problem in question involves three main levels of decision-making:

* Resource allocation – This level deals with making a decision on how to execute the operations, i.e., which resources are to carry out the operations.
* Order of operations – This decision relates to the position [in the schedule of the resource] that the operation is executed on, or in other words, the order in which the operations are executed on the resources.
* Operation scheduling – Determining the starting time for executing an operation on the executing resources.

A solution for an RSSP must include a decision on the three levels listed above. The process of decomposing the problem to sub-problems on the B&B tree structure defines in practice in which order and by which method these decisions are made. Following is a discussion of the various options for decomposing the problem.

**Types of Tree Nodes and the Order of Decision-Making**

It is not possible to determine the order of operations on a resource without simultaneously (or previously) determining the assignment of operations for a resource. Any attempt to determine the timing of operations, and only afterwards search for suitable assignment and ordering solutions for the determined scheduling, would pose an extremely complex challenge, as scheduling without making advance allocation decisions would be largely arbitrary. After making an allocation decision, it would not be possible to attempt to determine scheduling and only then the ordering, because with the very determination of the scheduling, the ordering is also determined. Therefore, three types of nodes were decided upon (in addition to the problem root node), and at each type of node, a decision is made on only one level. The following are the node types and their serial location on the tree.

* Allocation node – At nodes of this type, a decision is made regarding the allocation of resources for operations (what resources are assigned to execute the operations in the problem). This node appears as the first one under the problem root node.
* Ordering node – At nodes of this type, a decision is made on the ordering of operations on the resources (in what order the operations are to be executed on the resources). This node appears under the allocation nodes.
* Scheduling node – At nodes of this type, a decision is made on the scheduling of operations on the resources (at what times operation execution begins on the assigned resources). This node appears under an ordering node.

In other words, while searching for a solution, a decision on allocation of resources for operations is made, and only after that in accordance with the selected allocation is the order of executing operations on the resources determined; and at the bottom of the tree, the exact times of the operations on the allocated resources are determined by the predetermined order of operation execution. It should be noted that this order matches the order in Samaddar et al. (1999), though it was not explained as was done here.

**Decision Options at Every Node**

At every type of node, a decision is made related to a resource, as is a decision related to an operation which is performed on that resource. Since there are usually several resources and several operations, there are in principle four decision options at each node:

* A decision made on one resource and one operation, **1:1** ratio.
* A decision made on several resources and one operation, **Many:1** ratio.
* A decision made on one resource and several operations, **1:Many** ratio.
* A decision made on several resources and several operations, **Many:Many** ratio.

Later in this section, discussions and comparisons are introduced relating to the type of ratio that is best to implement at allocation and ordering nodes.

**Allocation Nodes**

Table 2 shows the calculation of the total number of nodes that can be obtained as the result of each type of decision regarding allocation of resources. The maximum number of nodes on a tree depends on the number of levels, and the number of branches at each level.

**Table 2 - Decision Options at an Allocation Node Based on the Criterion of Required Memory Space**

|  |  |  |  |
| --- | --- | --- | --- |
| Number  Ratio | Levels | Branching per level | Nodes |
| 1:1 |  | – At each node a decision is made on the method of performing an operation |  |
| 1:Many |  | – Number of possible sequences of operation methods that can be performed on resource . | where  and therefore several nodes are bounded by |
| Many:1 |  | – At each node, a decision is made on the way an operation is performed. |  |
| Many:Many | 1 | – Number of possible sequences of methods of operations. Number of sequences bounded by |  |

Table 3 shows the considerations regarding the rest of the decision criteria for each of the decision options.

**Table 3 – Decision Options at an Allocation Node Based on the Decision Criteria (excluding the required memory space criterion)**

|  |  |
| --- | --- |
| Decision Criterion | Considerations |
| Amount of work required before obtaining a first solution | Amount of work required until the first solution is obtained depends on the search strategy, size of the tree, and quality of the first solution. For the present discussion, we’ll refer only to the element of tree size as a property that affects the criterion being tested.  As stated in Table 2, the size of the shortest tree (total number of nodes) is of ratio Many:Many. |
| Quality of the first solution | The quality of the first solution depends on the search strategy. Regarding the current discussion, it is a possible solution for the problem that includes information on timing of each of the operations on each of the resources allocated to perform them, and therefore there is no difference between the decision options. |
| Maximum exploitation of the lower bound | 1:1 Ratio – At this ratio, a resource is selected to perform one operation. This choice does not add a great deal of information to the value of the calculated lower bound based on the critical path length of the operations, and/or the lower bound based on the calculation of work load on the resource, as from the choice itself, it is not certain that it would be clear how long these operations would take.  1:Many Ratio – At this ratio, for one resource, all operations that can be performed on it are selected. With this choice, there would be a greater addition of information for the value of the bound based on the length of the critical path and the work load on the resource in comparison with Ratio 1:1, because several operations would be selected.  Many:1 Ratio – With this ratio, for one operation, all resources that would execute it are selected. With this choice, there would be more additional information for the lower bound based on the calculation of critical path length in comparison with the 1:1 Ratio, and there would be a smaller addition of information for the lower bound based on the work load on the resource, in comparison to the 1:Many ratio. From the choice itself, we would know the length of time it takes for one operation.  Many:Many ratio – At this ratio, for each resource, the operations it will execute are known. With this choice, we would know the duration of executing an operation, and what operations are to be executed on each of the resources. In terms of the lower bound based on the length of the critical path and/or the work load on the resource, at this ratio there is the greatest addition of information in comparison with the rest of the ratios. |
| Implementation complexity | There is no difference between the options. |
| Amount of work required to the end of the algorithm run | It can be assessed that better exploitation of the lower bound and a smaller tree would lead on the whole to less work required to complete the algorithm run. |

To summarize what has been raised thus far, for the decision options at **allocation nodes**, it can be seen that there is preference for the **Many:Many** ratio for the following four decision criteria: required memory space, amount of work required until the first solution is obtained, maximum exploitation of the lower bound, and the amount of work required until the algorithm run is completed. For the rest of the decision criteria, there is no difference between these four options in implementing the decision.

**Ordering Nodes**

Table 4 shows the calculation of the total number of nodes that can be obtained as a result of each type of decision regarding the ordering of operations on the resources. The maximum number of nodes on the tree depends on the number of levels and the number of branches on each level.

**Table 4 – Decision Options at an Ordering Node Based on the Criterion of Required Computer Memory**

|  |  |  |  |
| --- | --- | --- | --- |
| Number  Ratio | Levels | Branches per level | Nodes |
| 1:1 | - for each resource , the number of levels is the number of tasks on the resource  () | The number of branches on a resource level becomes smaller than the maximum number equal to for a minimal number equal to 1. |  |
| 1:Many |  | – Number of possible orderings of the operations that can be carried out on resource . The number of orderings is bounded by . |  |
| Many:1 |  | The number of branches for a level is equal to the number of possible sequences of the operation as a task on the resource . This number is bounded by . |  |
| Many:Many | 1 | All solutions are possible. |  |

Table 5 shows the considerations for the rest of the decision criteria that were previously defined.

**Table 5 – Decision Options at an Allocation Node Based on the Decision Criteria (except required memory space)**

|  |  |
| --- | --- |
| Decision Criterion | Considerations |
| Amount of work required until the first solution is obtained | Amount of work required until the first solution is obtained depends on the search strategy, size of the tree, and quality of the first solution. For the present discussion, we’ll refer only to the element of tree size as a property that affects the criterion being tested.  As previously stated, the shortest tree size (total number of nodes) is of a Many:Many ratio. |
| Quality of the first solution | The quality of the first solution depends on the search strategy. With respect to the current discussion, this is a possible solution to the problem that includes information on the scheduling of all the operations on all the resources assigned to execute them, and so there is no difference between the decision options. |
| Maximum exploitation of the lower bound | Ratio 1:1 – At this ratio, one operation is selected to be executed as a task on the resource. With this selection, a large addition of information is obtained for the value of the lower bound calculated based on the critical path of the operations and/or the lower bound based on the calculation of work load on the resource, as we advance to the lower level of the resource.  1:Many Ratio – At this ratio, for one resource all the operations are selected that can be executed on the resource in a possible execution order. With this selection, there is a maximum addition of information for the value of the bound based on the critical path length and the work load on the resource.  Many:1 Ratio – At this ratio, one operation obtains a spot on each of the resources executing it. With this selection, there is a small addition of information both for the lower bound based on the length of the critical path and for the lower bound based on the work load on the resource. The bound will have a larger addition of information as we advance to the last level of the ordering nodes.  Many:Many Ratio – At this ratio, it is known for each resource which operations it is to execute. With this selection, we'll know the duration of the operation and what operations are to be executed by each of the resources. In terms of the lower bound based on the length of the critical path and/or the work load on the resource, at this ratio there is the greatest addition of information in comparison with the rest of the ratios. |
| Implementation complexity | The most complex implementation is the Many:Many ratio because of the complexity of formulating a possible ordering for each resource.. |
| Amount of work required to the end of the algorithm run | It can be assessed that better exploitation of the lower bound and a smaller tree would lead on the whole to less work required to complete the algorithm run. |

To summarize thus far, regarding the decision options at an **ordering node**, it can be seen that the Many:Many ratio is preferable, based on the following four decision criteria: required memory space, amount of work required until obtaining a first solution, maximum exploitation of a lower bound, and the amount of work required until the end of the algorithm run. For the quality of the first solution criterion, there is no difference in the implementation options; and for the implementation complexity criterion, the Many:Many ratio is the most problematic. In order to implement this ratio, a complex data structure must be managed, and searching conducted using an additional tree. For this reason, it can be said that it is preferable to use the next most preferable ratio based on the decision criteria, which is 1:Many.

As previously stated, the order of executing decisions is allocation of resources, ordering operations in accordance with the allocation, and finally, scheduling operations on the resources in accordance with the determined order. **At the scheduling nodes** are decision options of ratios 1:1, 1:Many, and Many:1, which lead to greater computational complexity than with the **Many:Many** ratio, which leads to one level of nodes of this type and allows finding the length of the critical path of operations which is actually a solution to the problem.

To summarize what has been said regarding the decision options at each type of node, it can be seen that there is an advantage in the following implementation option: **allocation node** – **Many:Many ratio**. As a result of this, one level of nodes of this type is obtained. **Ordering Node** – **1:Many ratio.** As a result of this, several levels of nodes of this type are obtained. The number of levels is the same as the number of resources in the problem (). **Scheduling node** – **Many:Many ratio.** As a result of this, one level of nodes of this type is received.

Figure 2 illustrates the general structure of the tree.



|  |
| --- |
| Root of the problem  Allocation nodes allocation allocation allocation  Ordering ordering  Ordering nodes  Ordering  Ordering ordering  Scheduling nodes  scheduling scheduling |

**Figure 2 - Tree Structure of the B&B Algorithm for Solving an RSSP**

**Arrangement of the Nodes on the Operation Order on the Resource Decision Level:**

Since there are several levels at the ordering nodes (the same as the number of resources), the question is asked, how should they be ordered during the search for the solution? Among the options for ordering the resources under a given resource allocation, we can consider the following:

* Arbitrary – Resources ordered arbitrarily.
* By resource workload – Resources ordered by workload, from high to low.
* By the number of critical operations – Resources arranged by the number of critical operations from high to low, i.e. a resource that executes more operations that are located on the critical path, is positioned closer to the root of the tree than another resource that executes fewer critical operations.
* By frequency of operations – Resources ordered by frequency of operations from high to low, i.e. a resource executing more operations that are also executed on more additional resources is positioned closer to the tree root.

Table 6 shows the considerations in selecting different options for ordering resources under a given allocation of resources, for all operations, with respect to decision criteria. In light of this discussion, we shall order the resources taking into account: work load on the resource, critical operations carried out on the resource, and frequency of operations executed on the resources. In addition, to ease implementation complexity, resource ordering could be performed once per given resource allocation. For this purpose, an index was defined to classify the resources into tree levels as follows: where  is the weight vector and  the workload vector for all the resources that are common to the resource (resources that execute operations that resource  executes). This arrangement aims to have those decisions that are related to operations located on the critical path of the solution and resources that are busy and/or involved in executing many operations come earlier in the order of ordering decisions.

**Table 6 - Methods for Arranging Resources Under a Given Resource Allocation**

|  |  |
| --- | --- |
| Decision Criterion | Considerations |
| Required memory space | All options require the same memory space. |
| Amount of work required before obtaining a first solution | Quantity of work required to arrange resources from the lowest to the highest is in the following order: Arbitrarily🡪 By workload; by frequency of operations 🡪 By number of critical operations. |
| Quality of the first solution | The earlier we consider the workload on the resources and the critical operations, the earlier we would obtain a solution that approaches an optimal solution. |
| Maximum exploitation of a lower bound | Ordering resources by load and by number of critical operations are two orderings that influence the value of the lower bound based on the length of the critical path, and on the lower bound based on the work load of the resource. |
| Implementation complexity | Putting resources in an arbitrary order with the least complex implementation. Ordering resources by load or by frequency of operations are orderings that can be executed the moment the allocation of resources for all operations is known. On the other hand, ordering by number of critical operations is an arrangement that must be updated as advancement is made to the bottom of the tree (the critical path of operations changes in accordance with the scheduling of operations on the resources) and therefore it is an ordering with a higher complexity of implementation. |
| Amount of work required to the end of the algorithm run | In accordance with what is stated in the maximal exploitation of lower bound criterion, for the present criterion, there is preference for ordering the resources by load and critical operations. |

**Ways to Implement the Decision:**

Every decision type can be implemented in two ways: by generating all the possible solutions simultaneously and selecting the preferred, or generating one or some of the solutions in the hope that we do not need all the solutions later in the process.

Formulation of all the options for allocating resources for all operations with exponential complexity bounded by. Formulation of all possible sequences of operations on the resource having exponential complexity bounded by . For this reason, it could be worthwhile each time to generate one solution of resource allocation and one solution of operation sequencing operations on the resource (depth first approach).

**2.2. Methods of Pruning and Bounding Options**

As mentioned earlier, there are two methods of bounding options on a tree: use of the lower bound, and maintaining a dominance ratio.

In the present study, no good dominance ratios were found for the purpose of bounding options. The following example demonstrates a possible dominance ratio between two allocations of resources.

Let's assume a problem with the following properties: , , , , , , , , , , , , , , , 

This problem has two possible resource allocations: , . It can be seen that the two allocation alternatives are not fundamentally different, that there is a dominance ratio between them and therefore if necessary, it is sufficient to study just one of them. Dominance ratios like these can exist in RSSP, but they are rare cases and much computational work is required to test for their existence during the running of an algorithm.

In light of the findings of Ainbinder et al. (2018), options can be bounded using any of seven methods for calculating a lower bound for an RSSP, but it is more useful to use bound  since it has been proven to be tighter than the rest of the bounds.

**2.3. Search Strategy**

In Section 1, four main search strategies were mentioned: Heuristic, Best lower bound, Depth first and Breadth first. Likewise, the advantages of each of the strategies were presented in relation to the performance indices according to theoretical and/or empirical research of Ibaraki (1987). For the present study, we'll propose several search strategies:

* Depth first – as described in the previous sub-section.
* A combination of depth first and best lower bound search strategies – finding a first solution by depth first, and then the search process continues to examine the nodes with the lowest lower bound until the algorithm is stopped.
* Combination of depth first and heuristic search strategies – finding a first solution by depth first and then the search process continues to examine the nodes with the lowest heuristic value until the algorithm is stopped.

In the study, an effort was made to develop good (not misleading) heuristics to navigate the tree, but this attempt failed and therefore a search strategy combining depth first and heuristic strategies was not considered in the present study. In view of the findings of Ainbinder et al. (2018) on the bounds, it can be seen that bound  has a high correlation with the target and therefore, in our opinion, it can be used not only as a bound but also as a heuristic.

Due to considerations of memory size, implementation complexity, and amount of work invested until a first solution is found, each time a search is made for an optimal solution under a given resource allocation, and at the end of its examination, a switch is made to study a new resource allocation that has not yet been bounded. Table 7 presents the considerations for selecting search strategies that were presented in relation to decision criteria. In view of the considerations raised in Table 7 regarding tree search strategies, it can be seen that it is useful to combine depth first and best lower bound search strategies.

**Table 7 – Search Strategy on a B&B Tree**

|  |  |
| --- | --- |
| Decision Criterion | Considerations |
| Required Memory Space | From small to large memory space in the following order: depth first, combination of depth first and best lower bound search strategies/ combination of depth first and heuristic search strategies. |
| Amount of work required before obtaining a first solution. | A depth first search strategy is faster than two other strategies until a first solution is found. |
| Quality of the first solution | The first solution is closer to the optimal solution with the best lower bound search strategy. |
| Maximum exploitation of a lower bound | The value of the lower bound does not depend on the search strategy. |
| Implementation complexity | There is no difference. |
| Amount of work required to the end of the algorithm run | It cannot be determined in advance as this criterion is also affected by other properties such as the lower bound. |

**3. Properties of B&B Algorithm for Solving an RSSP**

The objective of the present section is to examine the new search strategies vis-a-vis a search strategy from the literature (Samaddar et al., 1999). For this reason, we’ll provide a summary of what is the same in all the algorithms, and state the properties in which the algorithms differ.

Identical Properties in the Algorithms:

* How to decompose the problem – In all the algorithms, there are three types of nodes as described previously (i.e. allocation node, ordering node, and scheduling node). At an allocation node, a decision is made on the allocation of all the resources for all the operations. At an ordering node, a decision is made on the possible ordering of all the operations performed on the resource presented by the node. At a scheduling node, a decision is made on the execution times of all the operations on all the resources. In terms of the order of making the decisions, first of all a decision is made on the allocation of resources, after that on the order of operations under the allocation that was decided on, and finally, a decision is made on the execution times of the operations, according to the pre-determined order.
* How to bound options – All the algorithms use the lower bound  .

Differing Properties in the Algorithms:

* How allocation decisions are implemented – Two ways are examined: the first is advance formulation of all the allocation options and the second is formulating one resource allocation option (and going to depth).
* Order of the resources at ordering nodes – Two ways are examined: one is arbitrary ordering and the second is ordering by descending value of index.
* How to implement ordering decisions for a given resource – Two ways are examined: one is formulating possible orderings of operations on the resource and the second is selecting one possible ordering of operations (and going to depth).
* Search strategy – Two search strategies are examined: one is by depth and the second is combined search by depth and best lower bound (explained below).

Table 8 summarizes the combinations of strategies in the algorithms whose performance is compared empirically.

**Formulation of Resource Allocations for All the Operations:**

As was stated, resource allocation is determined by selecting a way to execute each operation. Formulation of all the allocation options means determining all the possible combinations of ways to perform all the operations. On the other hand, formulation of one allocation option (as needed) means determining one possible combination for how to perform all the operations. Every such option defines a sub-problem *SP* of the RSSP. Appendix A provides a description of the algorithm according to which one resource allocation is formulated each time as needed.

The analysis below is based on the allocation decision process in the structure of the "methods tree" (Definition 1) in which at each level a method is determined for execution of one of the operations. The conclusion of the analysis indicates a combination of execution methods leading to sub-problem *SP* with the lowest bound. The proofs for Propositions 1 and 2 are provided in Appendix B.

***Definition 1:***

The methods tree is a layout of resource allocation options (selecting ways to perform operations) according to the option of implementing the allocation decision with the 1:Many ratio (meaning, for one operation an allocation is selected on several resources.)

***Definition 2***

****is a set of operations for which the method of execution was determined to be ****.

***Definition 3***

**** is a node on the methods tree at which were determined methods of execution for a set of operations ****.

***Proposition 1***

In order to move in an RSSP problem **** from node **** to a sequence of methods with a minimal critical path, from among all the possible method sequences branching from ****, for every operation **** the method of execution with the minimum execution time should be selected.

From Proposition 1, it can be seen that the bounds**** that were proposed by Ainbinder et al. (2018) are based on durations of operations in accordance with the selection of execution methods and ordering constraints between the operations are a lower bound for the value of the solution in the root of the methods tree or at any of its nodes. It is important to mention that the rest of the bounds proposed by Ainbinder et al. (2018) are not a lower bound for the methods tree since they are based on selection of an execution method for all the operations in the problem.

***Proposition 2***

An algorithm for formulating resource allocations for all the operations (Appendix A) lays out the sequences of methods of the operations in a non-descending order of their bound value (****) (i.e. of lengths of critical paths in accordance with the formulation of the method sequences.)

**General algorithm for finding an optimal solution for an RSSP (  ):**

1. Formulation of a first resource allocation by selecting the execution method with the shortest execution time.

2. Examine the given resource allocation using B&B.

3. If the stop condition is true, go to 7.

4. Obtain a new resource allocation using an algorithm for formulating resource allocations for all operations.

5. If the stop condition is true, go to 7.

6. Go to 2.

7. End. The best solution found is the optimal solution.

Stop condition: The best solution value found thus far that is smaller than or equal to the value of the lower bound of the solution under the given resource allocation (which defines a specific sub-problem *SP*.)

**4. Comparison of B&B Algorithms**

Based on the considerations raised regarding the way to decompose the problem and search strategies, it was decided to focus on the six algorithms described in Table 8. The bounding strategy selected in all the algorithms is use of bound  (which was proven to have the best performance of all of those developed and examined by Ainbinder et al. (2018).

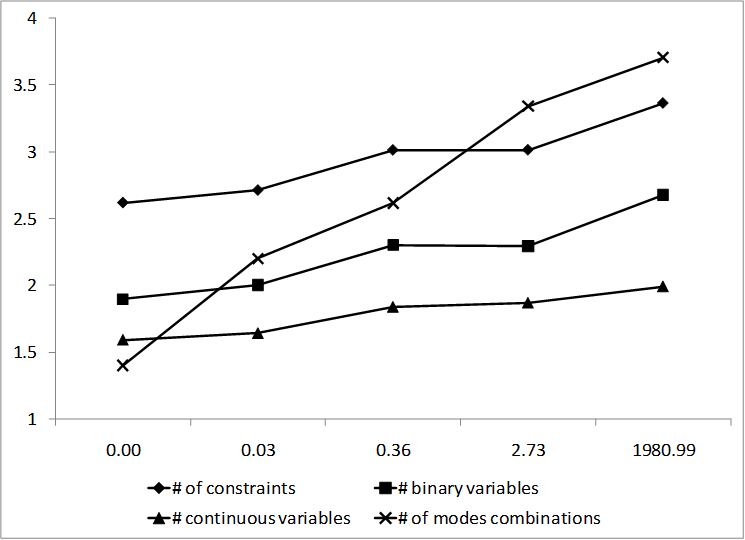
**Table 8 - B&B Algorithm Properties**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | | | 1 | 2 | 3 | 4 | 5 | 6 |
| Method to implement the decision at ordering nodes | Method to implement the decision at allocation nodes | All options | X | X | X | X |  |  |
| One option |  |  |  |  | X | X |
| Order of ordering nodes | Arbitrarily | X |  |  |  |  |  |
| By load |  | X | X | X | X | X |
| Method to implement the decision at ordering nodes | All options |  |  | X | X | X | X |
| One option | X | X |  |  |  |  |
| Search strategy | | By depth | X | X | X |  | X |  |
| Depth and bound |  |  |  | X |  | X |

For empirical comparison of the six algorithms, 105 problems were run, of which 16 were from Samaddar et al. (1999) and the rest were random problems.

**Figure 3 - Division of Problems into Complexity Quintiles, By Median Run Time of Algorithms 1 through 6**

Before comparing the performance of the algorithms, all the problems were divided into quintiles in increasing order of complexity, where the complexity of the problem was equal to the median value of the running durations of the different algorithms. In Figure 3, on the horizontal axis of the graph are quintiles by the median value of total run time, and on the vertical axis of the graph is the logarithm of the average total run time of each of the quintiles. In Algorithms 4 and 6 there were four problems included in the last quintile, and no optimal solution was achieved for them due to the memory limitation of the calculation means. For this reason, the two curves representing the total run time of these two algorithms do not reach the last quintile. In other words, it can be said that in the last quintile, Algorithms 4 and 6 reach a much higher run time than the rest of the algorithms. Figure 4 shows the complexity of the problems included in each of the quintiles by the average of several binary variables, several continuous variables, several constraint equations in representing the problem as MILP, and a total number of possible sequences of methods for the problem. It can be seen that the problem sets as divided indeed present problems in rising complexity (the first set is a set of problems with low complexity and the fifth set is the most complex.



**Figure 4 - Problem Properties by Complexity Sets**

The algorithms were compared on three main levels:

1. The amount of work invested until finding the first solution, and quality of the first solution relative to the optimal solution

2. The amount of work invested until finding an optimal solution to the problem

3. The total amount of work invested

Following is a description and findings of the indices taken into account in comparing performance of the algorithms.

* Average relative number of nodes decomposed until a first solution was found.

Method for calculating the index was , where  is the number of nodes decomposed until a first solution was found using the algorithm .. Table 4 shows the average and standard deviation of the index calculated for the problems in each quintile by each of the algorithms. It can be seen that two algorithms that on average require the least investment until finding a first solution are Algorithms 5 and 6, which combine the implementation of one option at the allocation stage, ordering resources by work load on the resource and the shared resources (Index *Hr*) and implementation of all the options at the ordering nodes. The two algorithms that invest the most work until a first solution is found are Algorithms 3 and 4, which implement all the options at an allocation node, ordering resources by *Hr* and implementation of all the options at ordering nodes. The two remaining Algorithms 1 and 2, which invest work at an intermediate level until finding a first solution to the problem, differ from each other in the order of resources.

**Table 9 - Average Number of Relative Nodes Decomposed until Finding a First Solution Using a B&B Algorithm**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem Set No. | Algorithms | | | | | | | | | | | |
| 1 | | 2 | | 3 | | 4 | | 5 | | 6 | |
| **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation |
| 1 | **0.78** | 0.25 | **0.80** | 0.25 | **0.93** | 0.18 | **0.93** | 0.18 | **0.52** | 0.40 | **0.52** | 0.40 |
| 2 | **0.77** | 0.22 | **0.78** | 0.22 | **0.82** | 0.24 | **0.82** | 0.24 | **0.51** | 0.42 | **0.51** | 0.42 |
| 3 | **0.72** | 0.33 | **0.72** | 0.33 | **0.94** | 0.16 | **0.94** | 0.16 | **0.45** | 0.41 | **0.45** | 0.41 |
| 4 | **0.90** | 0.18 | **0.90** | 0.18 | **0.93** | 0.18 | **0.93** | 0.18 | **0.20** | 0.35 | **0.20** | 0.35 |
| 5 | **0.85** | 0.35 | **0.85** | 0.35 | **1.00** | 0.00 | **1.00** | 0.00 | **0.16** | 0.35 | **0.16** | 0.35 |

* Average relative running time until a first solution is found.

Method to calculate the index is , where  is the running time until a first solution is found using Algorithm . Table 10 shows the average and standard deviation of the index calculated for the problems in each quintile according to each of the algorithms. It can be seen that the fastest algorithms are 5 and 6, and accordingly the execution order of the other algorithms is maintained, in keeping with the number of nodes that were decomposed.

**Table 10 - Relative Average Run Time Until Finding a First Solution with a B&B Algorithm**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem Set No. | Algorithms | | | | | | | | | | | |
| 1 | | 2 | | 3 | | 4 | | 5 | | 6 | |
| **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation |
| 1 | **0.85** | 0.33 | **0.89** | 0.28 | **0.98** | 0.11 | **0.96** | 0.20 | **0.78** | 0.40 | **0.78** | 0.40 |
| 2 | **0.79** | 0.27 | **0.81** | 0.29 | **0.85** | 0.26 | **0.98** | 0.07 | **0.50** | 0.49 | **0.50** | 0.49 |
| 3 | **0.61** | 0.30 | **0.56** | 0.30 | **0.69** | 0.19 | **0.98** | 0.08 | **0.29** | 0.39 | **0.27** | 0.38 |
| 4 | **0.77** | 0.18 | **0.71** | 0.17 | **0.79** | 0.12 | **0.96** | 0.09 | **0.04** | 0.07 | **0.04** | 0.08 |
| 5 | **0.59** | 0.33 | **0.74** | 0.35 | **0.92** | 0.11 | **0.78** | 0.14 | **0.12** | 0.27 | **0.16** | 0.35 |

* Average relative value of first solution from the optimal solution value.

The method for calculating the index is , where  is the first solution value found for problem , and  is the optimal solution value for the problem. Table 11 shows the average and standard deviation of the index calculated for the problems in each quintile according to each of the algorithms. It can be seen that also here the two algorithms with the best performances are Algorithms 5 and 6. Accordingly the order of the execution of the other algorithms was maintained as with the two previous indices.

**Table 11 - Average Relative Value of the First Solution Found with a B&B Algorithm from an Optimal Solution**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem Set No. | Algorithms | | | | | | | | | | | |
| 1 | | 2 | | 3 | | 4 | | 5 | | 6 | |
| **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation |
| 1 | **1.09** | 0.16 | **1.09** | 0.16 | **1.04** | 0.06 | **1.04** | 0.06 | **1.04** | 0.06 | **1.04** | 0.06 |
| 2 | **1.27** | 0.23 | **1.27** | 0.23 | **1.22** | 0.24 | **1.22** | 0.24 | **1.05** | 0.09 | **1.05** | 0.09 |
| 3 | **1.35** | 0.28 | **1.35** | 0.28 | **1.13** | 0.12 | **1.13** | 0.12 | **1.06** | 0.10 | **1.06** | 0.10 |
| 4 | **1.42** | 0.33 | **1.42** | 0.33 | **1.19** | 0.23 | **1.19** | 0.23 | **1.08** | 0.14 | **1.08** | 0.14 |
| 5 | **1.39** | 0.37 | **1.39** | 0.37 | **1.16** | 0.14 | **1.16** | 0.14 | **1.25** | 0.46 | **1.25** | 0.46 |

* Average relative number of nodes decomposed until an optimal solution was found.

The method of calculating the index is , where  is the number of nodes decomposed until an optimal solution was found using Algorithm . Table 12 presents the average and standard deviation of the index calculated for the problems in every quintile by each of the algorithms. According to the results, the most efficient algorithm for this index is Algorithm 5 which implements one option at the allocation nodes, resource ordering by work load on the resource and the shared resources, implementation of all the options at ordering nodes, and search by depth. Algorithm 6 produced a somewhat poorer result, with a different search strategy than Algorithm 5. The algorithm that invested the most work among all the algorithms is Algorithm 4, which implements all the options at allocation nodes, all the options at ordering nodes and the depth first search strategy with a lower bound. Among the remaining algorithms, there did not appear to be a significant difference in the mediocre results obtained.

**Table 12 - Average Relative Number of Nodes Decomposed before Finding an Optimal Solution with a B&B Algorithm**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem Set No. | Algorithms | | | | | | | | | | | |
| 1 | | 2 | | 3 | | 4 | | 5 | | 6 | |
| **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation |
| 1 | **0.64** | 0.36 | **0.65** | 0.36 | **0.69** | 0.35 | **0.78** | 0.33 | **0.36** | 0.34 | **0.59** | 0.44 |
| 2 | **0.89** | 0.24 | **0.88** | 0.24 | **0.85** | 0.27 | **0.95** | 0.11 | **0.13** | 0.14 | **0.23** | 0.34 |
| 3 | **0.69** | 0.36 | **0.69** | 0.36 | **0.63** | 0.39 | **0.97** | 0.12 | **0.15** | 0.23 | **0.37** | 0.43 |
| 4 | **0.87** | 0.25 | **0.87** | 0.25 | **0.85** | 0.25 | **0.93** | 0.21 | **0.10** | 0.16 | **0.28** | 0.37 |
| 5 | **0.60** | 0.40 | **0.60** | 0.39 | **0.60** | 0.40 | **-** | - | **0.23** | 0.32 | **-** | - |

* Average relative run time until an optimal solution was found.

Method to calculate the index is , where  is the running time until an optimal solution was found using Algorithm . Table 13 shows the average and standard deviation of the index calculated for the problems in each quintile according to each of the algorithms. According to the results, the most efficient algorithm for this index is Algorithm 5, and accordingly the performance of the rest of the algorithms is also in line with the results of the previous index (Table 6).

**Table 13 - Average Relative Run Time before Finding an Optimal Solution with a B&B algorithm**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem Set No. | Algorithms | | | | | | | | | | | | |
| 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | |
| **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation |
| 1 | **0.76** | 0.41 | **0.71** | 0.43 | **0.77** | 0.39 | **0.81** | 0.37 | **0.61** | 0.47 | **0.74** | 0.42 |
| 2 | **0.66** | 0.31 | **0.61** | 0.34 | **0.66** | 0.35 | **0.90** | 0.27 | **0.16** | 0.30 | **0.27** | 0.42 |
| 3 | **0.52** | 0.24 | **0.43** | 0.25 | **0.43** | 0.28 | **0.96** | 0.14 | **0.12** | 0.22 | **0.33** | 0.44 |
| 4 | **0.71** | 0.24 | **0.65** | 0.22 | **0.69** | 0.22 | **0.90** | 0.21 | **0.06** | 0.12 | **0.24** | 0.37 |
| 5 | **0.47** | 0.35 | **0.52** | 0.39 | **0.49** | 0.36 | **0.76** | - | **0.16** | 0.24 | **0.49** | - |

* Average total relative number of decomposed nodes.

The method of calculating the index is , where  is the total number of nodes decomposed using algorithm i. Table 14 shows the average and standard deviation of the index calculated for the problems in every quintile according to each of the algorithms. The results of this index are also in line with the two previous indices (Table 6).

**Table 14 - Average Total Relative Number of Nodes Decomposed Using a B&B Algorithm**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem Set No. | Algorithms | | | | | | | | | | | | |
| 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | |
| **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation |
| 1 | **0.54** | 0.31 | **0.55** | 0.31 | **0.54** | 0.30 | **0.85** | 0.24 | **0.30** | 0.18 | **0.77** | 0.29 |
| 2 | **0.75** | 0.31 | **0.74** | 0.31 | **0.72** | 0.33 | **0.93** | 0.21 | **0.10** | 0.12 | **0.33** | 0.37 |
| 3 | **0.63** | 0.36 | **0.63** | 0.36 | **0.61** | 0.37 | **0.90** | 0.22 | **0.22** | 0.33 | **0.55** | 0.42 |
| 4 | **0.72** | 0.39 | **0.72** | 0.38 | **0.69** | 0.39 | **0.96** | 0.14 | **0.05** | 0.08 | **0.35** | 0.43 |
| 5 | **0.57** | 0.42 | **0.57** | 0.41 | **0.62** | 0.43 | **-** | - | **0.32** | 0.41 | **-** | - |

* Average total relative run time.

Method of calculating the index is , where  is the total run time using Algorithm . Table 15 shows the average and standard deviation of the index calculated for the problems in each quintile according to each of the algorithms. These results are also in line with the three previous indices and Algorithm 5 has on average significantly better performance in each of the quintiles.

**Table 15 - Average Total Relative Run Time Using a B&B Algorithm**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem Set No. | Algorithms | | | | | | | | | | | | |
| 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | |
| **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation | **Average** | Standard deviation |
| 1 | **0.60** | 0.45 | **0.57** | 0.44 | **0.57** | 0.44 | **0.66** | 0.43 | **0.48** | 0.46 | **0.80** | 0.37 |
| 2 | **0.52** | 0.27 | **0.50** | 0.26 | **0.57** | 0.33 | **0.89** | 0.28 | **0.06** | 0.12 | **0.24** | 0.36 |
| 3 | **0.50** | 0.26 | **0.42** | 0.21 | **0.42** | 0.24 | **0.89** | 0.23 | **0.17** | 0.27 | **0.50** | 0.41 |
| 4 | **0.55** | 0.29 | **0.51** | 0.26 | **0.56** | 0.33 | **0.92** | 0.14 | **0.05** | 0.09 | **0.34** | 0.44 |
| 5 | **0.42** | 0.34 | **0.47** | 0.39 | **0.48** | 0.38 | **-** | - | **0.23** | 0.32 | **-** | - |

Theoretically, the algorithms are different from each other only in the way the decision is implemented at the allocation nodes. The theoretical complexity of examining a problem under a given resource allocation is identical for all the algorithms and is bounded by . The theoretical complexity of the formulation of all possible resource allocations is bounded by . The theoretical complexity of formulating resource allocations as needed is bounded by . In other words, Algorithms 5 and 6 theoretically have greater computational complexity than the rest of the algorithms, but empirically, Algorithm 5 performance was faster than the rest of the algorithms.

**5. Discussion and Conclusions**

Section 2 dealt with various implementation options for B&B principles in algorithms for finding one optimal solution for an RSSP. Based on a mapping and evaluation of the various alternatives, the combinations chosen for implementation and comparison were presented. To summarize the chapter, a theoretical and empirical comparison was made between the algorithms in the literature and new algorithms outlined in that section.

Comparing the average performance for each of the five quintiles for every algorithm and every index, collected in Table 16 below, focused on three aspects: (1) amount of work and time until finding a first solution, and its distance from the optimal solution; (2) amount of work and time until finding an optimal solution; (3) amount of work and time until the algorithm's work was finished.

**Table 16 - Properties and Average Relative Performance Indices, of B&B algorithms**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm: | | | 1 | 2 | 3 | 4 | 5 | 6 |
| Method to implement the decision at ordering nodes | Method to implement the decision at allocation nodes | All options | X | X | X | X |  |  |
| One option |  |  |  |  | X | X |
| Order of ordering nodes | Arbitrarily | X |  |  |  |  |  |
| By load |  | X | X | X | X | X |
| Method to implement the decision at ordering nodes | All options |  |  | X | X | X | X |
| One option | X | X |  |  |  |  |
| Search strategy | | By depth | X | X | X |  | X |  |
| Depth and bound |  |  |  | X |  | X |
| Until the first solution | | Nodes that were decomposed | 0.80 | 0.81 | 0.92 | 0.92 | **0.37** | **0.37** |
| Duration of run | 0.72 | 0.74 | 0.85 | 0.93 | **0.35** | **0.35** |
| Relation to optimum | 1.30 | 1.30 | 1.15 | 1.15 | **1.09** | **1.09** |
| Until the optimal solution | | Nodes that were decomposed | 0.74 | 0.74 | 0.73 | 0.89 | **0.19** | 0.38 |
| Duration of run | 0.62 | 0.59 | 0.61 | 0.87 | **0.22** | 0.42 |
| Until work is completed | | Nodes that were decomposed | 0.64 | 0.64 | 0.64 | 0.91\* | **0.20** | 0.50\* |
| Duration of run | 0.52 | 0.49 | 0.52 | 0.84\* | **0.20** | 0.47\* |

\*-Average calculated based on four out of five quintiles

It was found that Algorithms 5 and 6 which combine an implementation for one option at the allocation stage, ordering resources by work load on the resource and the shared resources, and implementation of all the options at the ordering nodes, performed better for each of the indices and for all the complexity sets of problems that were solved. For indices work load and performance time until the end of the search, Algorithm 5 performed significantly better than Algorithm 6. The difference between the two algorithms is the search strategy; Algorithm 5 uses depth first while Algorithm 6 combines depth first and bound value. Since both examined search strategies are the same until a first solution for a problem is found, their performance is similar for finding a first solution.

The difference between the two fastest algorithms until finding a first solution (5, 6) and between the two slowest algorithms (3, 4) is the way the decision is implemented at an allocation node. It can be seen that when all the options at an allocation node are implemented as expected, it takes more time until the first solution is found than implementing one option. This is because every contains at least one possible solution to the problem, and an algorithm that implements one option at the allocation nodes invests less work until a first solution is found, and therefore it finds it faster.

In terms of the distance of the first solution from the optimal solution, the two best algorithms (6, 5) are also the fastest algorithms until a first solution is found. Algorithms that produced the greatest distance between the first solution and the optimal solution (2, 1) implement all the options at the allocation nodes and one possibility at ordering nodes. The only difference between them, in the method of ordering the resources, did not produce a significant difference in their performance. Algorithms that implement all the options at allocation nodes investigate in the same order, therefore the difference in the index results between these algorithms is due to the way the decision is implemented at the ordering nodes. With algorithms that implement all the options at ordering nodes (3-6), the chances increased that a solution would be found closer to the optimal solution, since each time a branch is selected with a better lower bound; this is compared to the algorithms that implement one option at ordering nodes (2, 1) where the option is selected randomly from among all the options that were not yet investigated. The difference in the index results between the algorithms implementing all the options at the allocation nodes and the algorithms implementing one option at allocation nodes is due to the fact that in the problem there could be several with a minimal lower bound. This would cause the examination order of in these algorithms to be different.

At the second level, where the amount of work invested until the optimal solution to the problem is found, and the speed of the work by each of the algorithms are examined, it can be seen that results were obtained that were very close in terms of the amount of work invested and in terms of the running speed. According to the results, the best algorithms are the two algorithms that implement one option at the allocation nodes (6, 5). The most effective algorithm at this level is an algorithm that implements one option at allocation nodes, ordering resources according to workload on the resource and the shared resources, implementation of every option at ordering nodes, and depth first searching (5). The conclusion obtained from these results is that the method of implementation of all the options at allocation nodes is wasteful and not worthwhile, despite the theoretical complexity of the method.

At the last level, in which the total amount of work invested until the end of the algorithm run (and the speed of the work) are examined, it can be seen that the best algorithm is the one that performed best at the previous levels, Algorithm 5. Algorithms that implement a search strategy of depth first and lower bound (6, 4) cannot be compared since the most complex problems in the set did not manage to complete the algorithm run due to memory limitation.

The algorithm that had the best performance at all the levels is the one that implements one option at allocation nodes, arranging ordering nodes by workload on the resource and the shared resources, implementation of all the options at ordering nodes, and depth first search strategy (5).

**Appendices**

**Appendix A - Algorithm for Formulating Resource Allocations for All the Operations as Needed**

**Algorithm for formulating resource allocations for all the operations ():**

OperList – List of operations with execution method with minimal performance time for each one

Node, PreNode – Node: List of operations with the chosen performance method, depth of the node and lower bound value of the node

NodeList – List of nodes that have not been examined

SPList – List of SP that have been examined

ObjFunT – Value of the best solution found by B&B

1. OperList composition

2. Preliminary Node composition based on OperList 🡪 NodeList

3. As long as NodeList is not empty:

a. Remove node in advance NodeList 🡪 Node

b. Calculate lower bound of the Node

c. If the value of the lower bound of Node is lower than ObjFunT, or ObFunT is equal to zero then

• Formulation of SP based on the performance method of the Node

• If SP has not yet been examined, then send SP for solution via B&B.

d. Node 🡪 PreNode

• Delete Node from NodeList and formulate a new Node:

1. Find the depth first operation of node PreNode with more than one execution method.

2. For every execution method of the operation, formulate a new Node

3. Calculate a lower bound for Node

4. Add Node to NodeList

4. Sort NodeList in increasing order of the lower bound

**Appendix B - Propositions and Proofs for the Methods Tree**

**Proposition 1**: In order to get, in RSSP problem , from node  to a sequence of methods with a minimal critical path, from among all possible method sequences branching from  , for every operation  , the performance method must be selected with the minimal performance duration.

**Proof 1**: We'll construct a new problem  equivalent to , which has for every  only one performance method, and it is the one that is determined on the methods tree of  from the root until . It is obvious that a set of method sequences for the problem  is identical to a set of method sequences laid out from  in problem  . Since in  for every  there is a single execution method, then it is also the minimal execution method of . Therefore, selecting a minimal execution method for every  in problem  leads to the same sequence of methods as in selecting the minimal execution method in  for  in problem  . And since according to Proposition 22-4 thus is obtained in  , a sequence of methods with a minimal critical path, and therefore necessarily also from  in problem  is obtained in the same way a sequence of methods with a minimal critical path.

**Proposition 2**: An algorithm for formulating a resource allocation for all operations (  ) lays out the sequences of methods for the operations in a non-descending order of their bound value ( ) . (I.e. of the critical lengths of paths of the method sequences.)

**Proof 2**: At any stage in the work process of an algorithm  , there is a set of nodes (where  is a typical node) that for them the bound was calculated. According to the definition of a bound, it is equal to the length of the minimal critical path of any sequence of methods laid out from  . If we select to lay out from , then algorithm  leads first to a sequence of methods with an identical bound to the value of the bound in . Since in the process of descending the method tree, a bound cannot descend, and because with a minimal bound is selected, then the sequence of methods that we would reach would be with the minimal bond from those that have not yet been laid out