BRANCH AND BOUND ALGORITHM – SOFTWARE PROPOSAL FOR SCHEDULES OPTIMISATION

Pawel Nowak¹, Maciej Nowak²

Warsaw University of Technology, Civil Engineering Faculty, Warsaw, Poland
E-mail: ¹p.nowak@il.pw.edu.pl, ²maciej.nowak@gmail.com

Abstract. Branch-and-bound algorithms were developed by Łomnicki and Brown in the middle of the twentieth century. In order to improve the use of branch-and-bound algorithms authors present a new and simple software by which the calculations could be reduced to the minimum. Brown-Łomnicki Algorithm is presented and its adjustment for computer platform. Authors present also some screen shots and chosen information about the software. This paper considers a machines and equipment work scheduling problem of minimizing the unnecessary pauses in their work. This important scheduling problem needs to be solved in modern construction systems.

Keywords: schedule, software, Brown-Łomnicki, algorithm.

Introduction

Scheduling of machines, working on the same job in different sections of the constructed building is an important element of construction management. Managers should schedule such a work properly, to avoid unnecessary breaks in their works (Companys 2007). Branch and Bound algorithm is a “tree” used technique for solving optimization problem using search technique. Optimization problem is trying to find a schedule, which minimise breaks of the machines work. This solution can be represented as a state-space tree. Each branch represents all the possible next steps. The goal is to find a path which returns the maximum profit (in our case – minimal breaks and the shorter schedule). The basic idea is to have a fast way to compute the upper bound and lower bound of the profit of a sub-tree (Clausen and Traff 1994). If a sub-tree is know to have its upper bound lower than the lower bound of another sub-tree, then this sub-tree can be rejected because its best result is worse than the worst result of that another sub-tree.

Mathematical model of Branch and Bound algorithm based on the process of choosing the next “node” to expand – which means which schedule is optimal at the relevant stage (Stawowy and Mazur 1996). This model should also include a function for deciding when to stop a search path before reaching the end of the path. Abandoning searches early attempts to minimize computational efforts to find the minimal solution. The software presented in the following chapters works efficiently, minimizing number of steps while finding the optimal solution. The basis of Branch and Bound algorithms is a ranking function (Horowitz et al. 2008). The ranking function assigns a value to each node in the graph. At each step, a branch and bound algorithm uses the ranking function to decide which node to expand next. The usual DFS (Depth First Search) and BFS (Breadth First Search) (Horowitz et al. 2008) exploration algorithms perform a blind search of the graph. The ranking function, here called c(x), ranks nodes based on the cost (in case of the scheduling – could be based on the time time) of a minimal solution reachable from node x. The problem with this ranking function is that the minimal solution must be known ahead of time. Instead, the ranking function uses an estimate, g(x), of the cost (or time) of a minimal solution reachable from node x.

Using g(x) to rank nodes may require exploring unnecessary nodes in the graph (schedule) especially when g(x) is estimated not correctly. The final element of the ranking function measures the cost (time) of reaching a node from the root node.

After including a function of h(x), the ranking function c(x) is (Companys 2007):

\[ c(x) = f(h(x)) + g(x). \]  
(1)

In formula 1 f determines how much significant to give the cost (or time) of reaching node x. If \( f(h(x)) = 0 \) then the algorithm makes long, deep searches of the graph. If \( f(h(x)) > 0 \) then the algorithm considers nodes
close to the root before making long potentially fruitless forays into the graph. A node is live if its sub graph might contain a minimal solution. Live nodes are potential candidates for exploration. A node is dead if its sub graph cannot contain a minimal solution (Horowitz et al. 2008).

If the estimated cost of a minimal solution reachable from x is less than the actual minimal solution reachable from x then a node can be killed when the estimated cost is greater than the least upper bound on a solution. In symbols, if \( c(x)^{\text{est}} \geq \text{upper} \) then an optimal solution is not reachable from x (assuming upper is the least upper bound on a solution).

Solution of the problem requires multi-calculations. Software presented in the next chapter minimizes the calculations to the absolute minimum.

**Brown-Lomnicki Algorithm**

Solving NP-hard discrete optimization problems to optimality is often an immense job requiring very efficient algorithms, and the Branch and Bound algorithm is one of the main tools in construction of these. A Branch and Bound algorithm searches the complete space of solutions for a given problem for the best solution. Explicit enumeration is normally impossible due to the exponentially increasing number of potential solutions.

The use of bounds for the function to be optimized combined with the value of the current best solution enables the algorithm to search parts of the solution space only implicitly (Dominguez-Marin 2003, Brown and Churchill 2003). The Brown-Lomnicki algorithm is designed to put n jobs done by m machines in the optimal order. The nodes in this algorithm represent the permutations of already chosen r jobs and the estimated time of finishing the work in this order. Assumptions for using this algorithm are:

- the machines works on every job in the same order (1,2, ...,m),
- every machine can work on one job at the time and every job can use one machine only at one time (Jaworski 1999).

The Brown-Lomnicki algorithm uses the ranking function in the following form (2–5):

\[
\xi(G(w_j)) = \max(\xi_1, \xi_2, ..., \xi_m),
\]

\[
\xi_j = T_{ij}^p + A + Bmin,
\]

where \( (T_{ij}^p) \) is the \( h(x) \) function and \( (A + Bmin) \) is the estimation function – \( g(x) \), and

\[
A = \sum_{j \in w_r} t_{ij},
\]

\[
Bmin = \min \sum_{j \in w_r, i \in J} t_{ij},
\]

where \( i \) – machine number \( (i = 1, 2, ..., m) \), \( j \) – job number \( (j = 1, 2, ..., n) \), \( w_r \) – permutation of r elements of \( j = (1,2, ..., n) \).  

**Software Proposal for Schedules Optimization**

The software was programmed in Java (Hall et al. 2009, Harris and Ross 2006, Stelting 2005, Tyiagi et al. 2004, Janicki and Izydorczyk 2001, Schildt 2005) an object oriented programming language (Eckel 2006; Sahni 2005; Graham 2004). The concept is based on a simple block diagram presented on Fig 1. The main object, foundation for further calculations, is called the Permutation.

It consists of:

- assumed order of first \( r \) jobs – \( K \) set,
- \( \xi \) value.

The software receives the data from the user: number of machines, jobs and times of work for each machine on each job (Fig 2). It initially creates \( n \) Permutations and calculates \( \xi \) for each of them.

In the next step it searches for Permutations with the lowest \( \xi \) and checks if any of them could be branched, which basically means – if their K sets have less then \( (n–1) \) elements. This step is repeated every time new branches are added. The application ends calculations and presents the solution when all Permutations with the lowest \( \xi \) are the leaves – their K sets have \( (n–1) \) elements. Screenshots on the Figs 3 and 4 show the running program: calculations details and efficiency of the software. The application is programmed to use additional optimization techniques to limit computer resources usage during calculations (these are not described in this paper).

**Practical Example**

The software was tested on the example of construction of the housing estate – 10 different (size – from 150 m\(^2\) to 250 m\(^2\) and cubature from 350 m\(^3\) to 410 m\(^3\)) single family detached houses.

Construction of each house requires 15 tasks to be performed (Fig 5), that were divided between 6 working gangs. The works division was as follows (Fig 6):

- gang 1: 1. Earthworks,
- gang 2: 2. Concrete foundations, 3. Concrete cellar walls, 4. Concrete cellar ceiling,
- gang 4: 4. First floor ceiling, 10. Stairs,

Working hours for each gang for house construction are presented in Table 1.

Fig 7 shows general schedule of 10 houses construction by 6 gangs – before optimization approx. 8900 hours.

Fig 8 shows the schedule after optimization with use of the suggested software.
Fig 1. Block diagram for the software proposal.

Fig 2. Branch and bound algorithm, data entering.
Fig 3. Branch and bound algorithm, calculations

Fig 4. Branch and bound algorithm, last step – results

Fig 5. Relevant works layout on the chosen house (15 tasks, approx. 1900 working hours)
Table 1. Working hours for each gang for house construction.

<table>
<thead>
<tr>
<th>Working gangs</th>
<th>House nr / working hours of each gang</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Gang 1</td>
<td>123</td>
</tr>
<tr>
<td>Gang 2</td>
<td>601</td>
</tr>
<tr>
<td>Gang 3</td>
<td>527</td>
</tr>
<tr>
<td>Gang 4</td>
<td>330</td>
</tr>
<tr>
<td>Gang 5</td>
<td>237</td>
</tr>
<tr>
<td>Gang 6</td>
<td>102</td>
</tr>
</tbody>
</table>

Fig 6. Relevant works of six working gangs (6 gangs, approx. 1900 working hours)

Fig 7. Schedule of works before optimization – 6 gangs, 8900 working hours (each house works marked with relevant color)

Fig 8. Schedule of works after optimization – 6 gangs, 8070 working hours (each house works marked with relevant color)
Conclusions

A large number of real-world planning problems called combinatorial optimization problems share the following properties: they are optimization problems, are easy to state, and have a finite but usually very large number of feasible solutions. The Branch and Bound algorithm presented takes advantage of this possibility for obtaining an optimal solution as early as possible in the branch-and-bound search. The presented iteration has three main components: selection of the node to process, bound calculation, and branching. At any point during the solution process, the status of the solution with respect to the search of the solution space is described by a pool of yet unexplored subset of this and the best solution found so far. The presented software limits number of calculations to the minimum. Moreover the obtained solution is truly optimal, which means it is equal to a strict mathematical solution of this optimization case, while the volume of calculations is in most cases significantly limited.

References