In search of an improved BoM and MRP algorithm

Conference Paper · June 2009

DOI: 10.1109/ITI.2009.5196167 · Source: DBLP

3 authors:

Zlatko Stapic
University of Zagreb
46 PUBLICATIONS **248** CITATIONS
SEE PROFILE

Tihomir Orehovački
Juraj Dobrila University of Pula
127 PUBLICATIONS **689** CITATIONS
SEE PROFILE

Alen Lovrencic
University of Zagreb
44 PUBLICATIONS **129** CITATIONS
SEE PROFILE

In Search of an Improved BoM and MRP Algorithm

Zlatko Stapic, Tihomir Orehovacki, Alen Lovrencic
Faculty of Organization and Informatics
Pavlinska 2, 42 000 Varaždin, Croatia
zlatko.stapic@foi.hr, tihomir.orehovacki@foi.hr, alen.lovrencic@foi.hr

Abstract. *In facing the problem of building a complex yet fast and resource efficient Enterprise Resource Planning (ERP) or Extended Resource Planning (XRP) system the first obstacle is the creation of a Bill of Materials (BoM) algorithm, which is used to calculate material needs according to the production plan. Taking into consideration different possibilities of BoM implementation, we compared linked lists, tree structure and matrix implementation in relation to algorithm complexity. As we came to the conclusion that none of them suits the needs of modern systems, we proposed a new BoM algorithm called multiple linked list implementation. It combines the advantages of other implementations with a new approach based on two complex data structures used to represent both the data and relationships between products, parts or components used in production.*

Keywords. Material Requirements Planning, Bill of Materials, BoM algorithm, data structures, algorithm complexity

1. Introduction

In the late 1960's, when the International Business Machines Company (IBM) presented the first Requirements Planning System (RPS), the concept of Bill of Materials (BoM) was also introduced [10]. A few years later, in the early 1970's, the concept of Material Requirements Planning (MRP) appeared and immediately gained scientific attention leading to the popularity of this widely used method for efficient planning of necessary production materials [14]. As dissolution of BoM is one of the first steps in conceiving the MRP algorithm, BoM became the basis of this and every subsequent system for needs planning.

Over the last thirty years, the process of planning as well as the concept of MRP has been expanded and, as a result, MRP algorithms have become more complex and have come to take into account Master Planning, scheduling and routing [19]. Furthermore, an MRP algorithm has been a fundamental element of all planning systems, starting from IBM's RPS to this century's Enterprise Resource Planning (ERP) and Extended Resource Planning (XRP). As these systems are becoming more and more complex, it is of crucial importance that MRP algorithms should be as fast and as less demanding as possible. These criteria also apply to BoM dissolution algorithms, as they are at the core of MRP algorithms.

The primary purpose of this paper is to present a series of different approaches to BoM dissolution and basic needs computation algorithms. In the paper, linked lists implementation, tree structure representation and matrix implementation are described, presented in pseudo code and subsequently compared. In addition to these traditional implementations we also propose a new implementation based on several advantages of other algorithms combined with the new idea of data representation using two complex data structures. The main comparison criterion will be algorithm time complexity, while all the presented algorithms will be based on the use of dynamic structure of linked lists.

2. Definitions

2.1. The Bill of Materials

The simplest definition a *bill of materials* is that of a list that contains information about materials, parts, sub-assemblies and components which should be used to produce the final product [18].

The BoM can be defined and represented in different ways. The most common representation of BoM is a *tree structure*, the root of which is the final product, with the descendants of each node representing the components or materials necessary to produce it. It is important to emphasize the fact that in such a representation each product variant is treated as a separate product.

Over the past years, several different BoM models have been defined by different authors,

such as the identical-part BoM, plus-minus BoM and multiple BoM by Scheer [15], the modular BoM, variant BoM and generic BoM by Van Veen and Wortman [17] and the object-oriented BoM by Chang and Fisher [2]. However, the point of departure for this paper will be Scheer's definition of BoM based on two types of entities, *parts* and their *relationships* [15], as shown in figure 1.

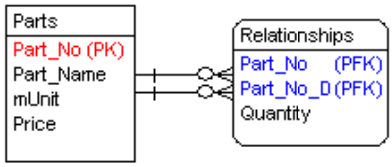

Figure 1. Bill of materials ERA model

BoM data structured in this way is usually read and stored in RAM for further processing.

2.2. Data structures used

Although different BoM processing systems have their own processing algorithms, the first step is common to all of them. The data should first be transformed into a suitable *data structure*. This data structure provides basis for the subsequent processing and as such completely determines the algorithm resource usage and its complexity.

In the following paragraphs we will show several most common data structures to represent the information about the components and build parts, along with other information (such as quantity) that should be used to produce product *X*.

The following figures (2, 3 and 4) show three different ways of presenting the same information on *product X*.

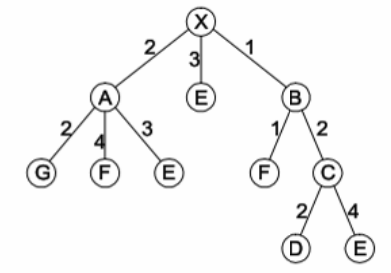

Figure 2. Tree structure representation

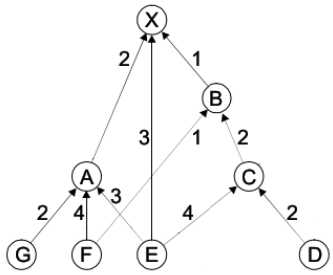

Figure 3. Gozinto-graph representation

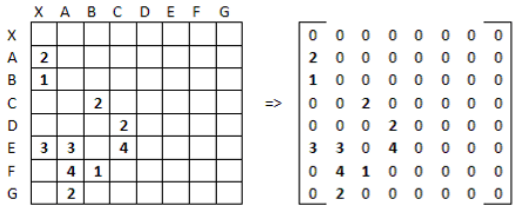

Figure 4. Matrix representation

Product *X*, shown in the figures, consists of two components *A*, three parts *E* and one component *B*. In addition, it can be seen that components *A* and *B* also consist of different parts or sub-components. In order to understand the third representation properly we should observe columns as products consisting of several other components or parts, with quantity stated in rows.

Although all the three *data structures* represent the same product, they require completely different BoM processing algorithms. In order to determine the best BoM processing algorithm, or to propose a new one, we will compare them according to their complexity.

2.3. Algorithm complexity

We distinguish two measures of software efficiency: the time it takes to execute an algorithm, and the amount of memory needed to calculate and store the results. Accordingly, there are two measures of algorithm complexity – time and space. An MRP algorithm is a time-consuming algorithm, and therefore we are particularly interested in estimating the complexity of implementation in terms of the time needed to perform operations on the data.

Time complexity of a program is measured by the number of basic operations to be performed in order to calculate the result. The complexity unit is given to each command within the algorithm, so the time complexity of the program is calculated as the *total complexity of all commands* in it.

Program complexity should be calculated on the basis of the value of input parameters. Therefore, we are interested in seeing how quickly the time resource usage of the algorithm will grow while increasing the input values [3]. We also introduce mathematical notation called the order of magnitude (or the Big-O) notation that approximately describes the growth rate complexity function of the algorithm [4]. We will use it to make our results more readable.

3. Comparing implementations

3.1. Linked lists implementation

According to theorists in this field, linked lists are probably the most commonly used dynamic data structure in program design [12]. This data structure is usually implemented by pointers and can be used in several different ways. Singly linked lists, queues, circular lists, doubly linked lists and others can be found in [5], [8] and [9], although all the lists represent data elements that contain one or more pointers to their neighboring elements. Such implementation of linked lists is shown in figure 5.

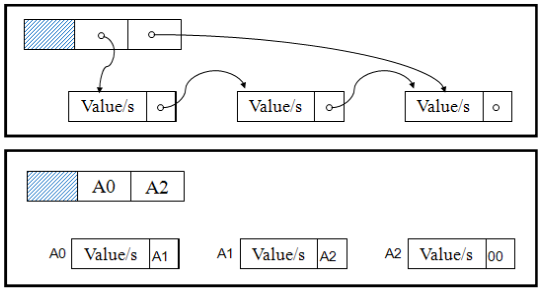

Figure 5. Singly linked lists

While representing following linked lists data structures, we will use a second notation (with the address pointing between elements), in order to ensure that the figures are readable.

Algorithms that are used to create linked lists and manipulate the data they contain are well known, so there is no need for them to be analyzed here. The complete list of operations that are used to work with linked lists [13], along with their complexity, is shown in table 1.

Further optimization by using doubly linked lists could improve the complexity of *Previous* operation to *O(1)*, but we do not need it in our algorithms.

Operation	Complexity
First	O(1)
End	O(1)
Next	O(1)
Previous	O(N)
Locate	O(N)
Insert	O(1)
Delete	O(1)
Retrieve	O(1)
MakeNull	O(1)

Table 1. List of operations and complexity

Using the presented operations, we can define our own BoM algorithm that will create a linked list containing calculated material requirements. There are several important notes to be introduced in order to make the algorithms readable:

- *N* will represent the number of different elements contained in the bill of material;
- *O(...)* will be the pessimistic (the worst-case) number of steps the algorithm takes;
- *table* – data in a database describing all relationships between elements (*max* N^2);
- *descendant* – material or part needed in the production of other elements;
- *requirements* – list of requirements in the production of final products (*max N*);
- algorithms will be described by pseudo code to preserve simplicity.

Including all the above-mentioned principles, our described algorithm could be as follows:

```
Create empty List                    MakeNull   O(1)
For each element in requirements     While      O(N)
  Input data in List                 Insert     O(1)
Next element in requirements         Next       O(1)
For each List element                While      O(N^4)
  For each descendant element        While      O(N^2)
    Input data in List               Insert     O(1)
  Next descendant element            Next       O(1)
Next List element                    Next       O(1)
```

Code 1. Linked lists BoM algorithm

As the size of the *table* is determined by the number of elements contained in the bill of material, and in the worst case it could be (N^2), the inner loop in the second algorithm *For each descendant element* has the complexity of $O(N^2)$.

Consequently, as it can be seen in pseudo code 1, in the worst case of N^2 relationships, algorithm complexity is determined by the outer *for each* loop with the complexity of $O(N^4)$. According to this pessimistic complexity, such a *polynomial* algorithm is not very efficient.

3.2. Tree structure implementation

We can define a structured tree as a hierarchical data structure which is represented by nodes connected by edges. Nodes are used to represent certain data, and edges show how the nodes are related [6]. Although trees are simplified instances of a more general structure called a *graph* (see figure 3), the implementation of a Bill of Materials by using a structured tree is both natural and possible, as shown in figure 2. When it comes to algorithm implementation, different approaches can be used, but we will focus on using pointers and linked lists. Figure 6 shows the linked list implementation of *product X* BoM represented by a structured tree. The list of operations and their complexity can be found in [1].

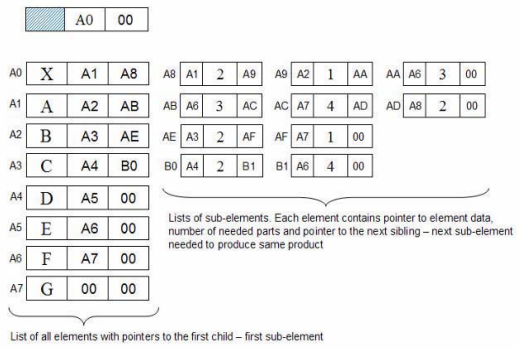

Figure 6. Tree structure BoM representation

Taking into consideration the list of operations and implementations shown in the previous figure, we can define pseudo code and determine its approximate complexity.

```
Create empty Tree                 Init      O(1)
For each row in table             While     O(N^3)
  If Label exists                 Locate    O(N)
    Insert next sibling data      Create    O(1)
  Else
    Insert data in main list      Create    O(1)
    Insert first child data       Create    O(1)
Read next row                     Wend      O(1)

Create empty List                 MakeNull  O(1)
For each element in requirements  While     O(N^2)
  Find element in main list       Locate    O(N)
  Insert all desc elem in List    Insert    O(N)
Next element in requirements      Next      O(1)
For each List element             While     O(N^3)
  Find element in main list       Locate    O(N)
  Insert all desc elem in List    Insert    O(N)
Next List element                 Next      O(1)
```

Code 2. Structured tree BoM algorithm

It is evident that the pessimistic complexity of the structured tree BoM algorithm presented in pseudo code 2 is also polynomial, $O(N^3)$. However, compared to the previous implementation, the complexity of this algorithm is significantly lower. Although it is possible to apply different optimization routines to this algorithm, it is not possible to improve it so as to make it usable in situations other than simple needs calculation.

3.3. Matrix implementation

While processing BoM, matrix algebra can also be used [7]. Before stating the results defined in [7], we should introduce the symbols used. B_0 (requirements vector), I (identity matrix), S (resources vector), MSP (product structure matrix), MUP (overall requirements matrix), KPP (production final needs) and M^{-1} (inversed matrix) are the symbols used in the following equations:

$$MUP = (I\text{-}MSP)^{-1} \quad (1)$$

$$KPP = MUP\,(B_0\text{-}S) = (I\text{-}MSP)^{-1}\,(B_0\text{-}S) \quad (2)$$

Taking into consideration algorithm theory and developed matrix algorithms [7][16], we can conclude that the most complex matrix algorithm is the one for inversed matrix calculation. According to [16], the complexity of optimized *p-adic* algorithm for inversed matrix calculation is $O(N^4(\log N)^2)$.

If we ignore all the other calculations besides inversion, the complexity of this matrix implementation of the BoM algorithm is also polynomial and much greater than $O(N^3)$, as calculated in the previous section.

3.4. Multiple linked lists implementation

Taking into consideration the calculated time complexity of all the three implementations, we have come to the conclusion that the presented algorithms, in their current form, have different and important shortcomings and could not be used in modern ERP or XRP systems without serious optimization and enhancement.

The solution we would therefore like to propose is called *multiple linked lists implementation*. It is based on advantages of previously analyzed algorithms. In other words, we propose to use:

- linked lists (in terms of speed and direct memory access to the data describing BoM), and
- matrix data representation (in terms of data availability and semantic cohesion between data presented in rows and columns).

As we have already stated, the only problem regarding matrix-based implementation (see figure 4) is the need for inverse matrix calculation (see formula 2). We tried to avoid this calculation by creating *complex data structures*, and thus prepare the data for a simple and efficient algorithm.

The proposed complex data structure consists of two substructures for elements which will be connected and thus form matrix representation.

The first data structure (DS1) contains information about the element name (and other important data) and five pointers to connect elements with (see figure 7):

- first and last element (component or part) to be used when producing this element;
- first and last element (component or part) containing this element in its structure;
- next element, part or product that can be found in BoM.

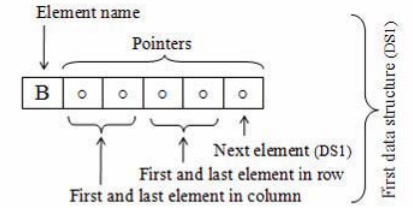

Figure 7. First data structure

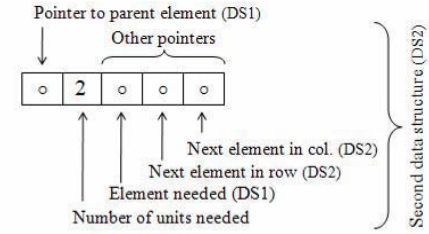

Figure 8. Second data structure

On the other hand, the second data structure (DS2) contains information about tree nodes, parts of components used in production:

- pointer to the product which contains this part or component;
- number of parts needed in production;
- pointer to the basic information on product, part or component;
- pointer to the next list element containing information on where else to use this part or component;
- pointer to the next list element containing information on another part needed to produce the mentioned product;

Combining these two data structures, it is possible to create *multiple linked lists BoM implementation,* which will contain all the information needed for a new algorithm to calculate all material requirements in a planned production cycle. An example of multiple linked lists containing information and all the necessary pointers is shown in figure 9.

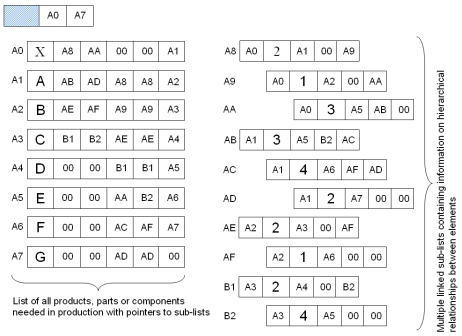

Figure 9. Multiple linked lists implementation

There are several improvements that new algorithms based on these structures could benefit from. First of all, it is easy to create and fill a data structure out of data contained in a database or another data source. Secondly, it is also easy to access the information on all parts needed in the production of a particular product by simple movement through the list representing the proper column. Finally, it is easy to find all products that depend on one particular part because all the necessary data on that subject is presented in a list representing the proper row.

A basic dissolution algorithm that uses the data structures presented above and list operations described in table 1 could be described with the following pseudo code.

```
Create empty list                 MakeNull   O(1)
For each row in table             While      O(N^3)
   If Label exists                Locate     O(N)
     Insert data in sub-list      Create     O(1)
   Else
     Insert data in main list     Create     O(1)
     Insert data in sub-list      Create     O(1)
Read next row
```

```
Create empty List                  MakeNull   O(1)
For each element in Requirements   While     O(N^3)
  Find element in main list        Locate    O(N)
  Insert element in List           Create    O(1)
  For each sub-elements            While     O(N^2)
     Insert element in List        Create    O(1)
Next element in Requirements       Next      O(1)
```

Code 3. Basic dissolution algorithm

According to pseudo code 3, the complexity of the proposed algorithm is formally polynomial $O(N^3)$, which places this algorithm in the same category as the previously presented tree structure implementation.

On the other hand, the benefits of using this algorithm are obvious, as the new data structures enable us to find other dependant products for any particular part or component being observed.

4. Conclusion

In this paper we presented and analyzed several implementations of BoM and MRP algorithms. The performed analyses show that time complexity of different algorithms ranges in the polynomial area near the approximate value of $O(N^3)$.

Taking into consideration the previously recognized advantages of linked lists, tree structure and matrix implementation we also proposed new *multiple linked lists implementation*, which results in better data availability and semantic cohesion between data presented in rows and columns (a benefit accepted from matrix implementation) and direct memory access to the data (a benefit accepted from linked lists implementation). The stated benefits are gained with no increase in terms of time complexity.

Further research in this area is necessary and could be performed in order to optimize the presented algorithm and analyze its characteristics on different real-life test cases.

5. References

[1] Aho AV, Hopcroft JE, Ulman JD. Data Structures and Algorithms, 2nd edition. Reading: Addison-Wesley; 1987.

[2] Chung Y, Fischer G. A. Conceptual Structure and Issues for an Object Oriented Bill of Materials (BOM) Data Model. Computers and Industrial Engineering, 1994; 26(2): 321-339.

[3] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to Algorithms, Cambridge: The MIT Press; 2001.

[4] Dale N. C++ Data Structures, Third Edition, Sudbury: Jones and Barlett Publishers; 2003.

[5] Deshpande PS, Kakde OG. C & Data Structures. Rockland: Charles River Media; 2004.

[6] Drozdek A, Data Structures and Algorithms in C++, Second Edition. Pacific Grove: Brooks/Cole; 2001.

[7] Hell M. Primjena matrične algebre kod temeljnog računa nad sastavnicama, White paper, 2007.

[8] Horowitz E, Sahni S, Anderson-Freed S. Fundamentals of Data Structures in C. New York: W.H. Freeman & Co.; 1992.

[9] Kruse RL, Ryba AJ. Data Structures and Program Design in C++, New Jersey: Prentice Hall; 2000.

[10] Mabert VA. The early road to material requirements planning. Journal of Operations Management, 2007; 25(2): 346 – 356.

[12] Parker A. Algorithms and Data Structures in C++, Boca Raton: CRC Press; 1993.

[13] Preiss BR. Data Structures and Algorithms with Object-Oriented Design Patterns in C++, New York: John Wiley & Sons; 1999.

[14] Ram B, Naghshineh-Pour MR, Yu X. Material requirements planning with flexible bills-of-material. International Journal of Production Research, 2006; 44(2): 399 – 415.

[15] Scheer AW. Business Proces Engineering. Berlin-Heidelberg: Springer-Verlag; 1998.

[16] Sedgewick R. Algorithms, Reading: Addison-Wesley; 1988.

[17] Van Veen EA, Wortmann JC. New Developments in Generative BOM Processing Systems. Production Planning & Control, 1992; 3(3): 327-335.

[18] Vegetti M, Henning GP, Leone HP. An object-oriented model for complex bills of materials in process industries. Brazilian Journal of Chemical Engineering, 2002; 19(4): 491 – 497.

[19] Yenisey MM. A flow-network approach for equilibrium of material requirements planning. International Journal of Production Economics, 2006; 102(2): 317-332.