On solving assignment type mixed integer linear programming problems within decision support systems

Citation for published version (APA):

Document status and date:
Published: 01/01/1983

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 05. Sep. 2019
Memorandum COSOR 83-21

On solving assignment type mixed integer linear programming problems within decision support systems

by

J. van Nunen*, A. Beulens* and J. Benders**

* Graduate School of Management, Delft
** Eindhoven University of Technology, Department of Mathematics and Computing Science

Eindhoven, The Netherlands

September 1983
Abstract

In this paper we discuss how assignment type mixed integer problems as they often occur in practical situations can be handled. It is shown that the linear programming relaxation of mixed integer problems of a certain type yields solutions with only a few non-unique assignments. We will give tight upperbounds for the number of non-unique assignments that result after solving the linear programming relaxation of the problem.

Since the number of splitted assignments is small one can use a heuristic to reach a practically good and feasible assignment.

The final part of the paper is devoted to deriving heuristics for generalized assignment type problems. These heuristics take the LP-relaxation solution as a startingpoint. Conditions are given which guarantee that the heuristics produce good feasible solutions.

Key words: Mixed integer linear programming, assignment problems, location-allocation problems, distribution problems.
1. Introduction

Many practical problems in the area of location-allocation and distribution can be modelled by mixed integer linear programs. We encountered such problems during a practical study of location-allocation problems within a Dutch brewery, see (1), (2).

There we were confronted with assignment type problems with over 60,000 zero-one variables. Later on we met similar problem structures e.g. for the storage of LPG-gas within an oil company and several types of time-table problems. It is well known that if one solves the linear programming relaxation of such mixed integer problems, one might get solutions in which only a relatively small number of zero-one variables are not integer. In (3) we gave tight upperbounds for the number of zero-one variables that are not integer in the solution of the LP-relaxation solution. In this paper we will focus on heuristics that take the solution of the LP-relaxation as a starting point and produce feasible solutions for the mixed integer problem.

Moreover we will give conditions which guarantee that the heuristic will yield a good feasible solution of the mixed integer problem.

Section 2 is devoted to a description of a location-allocation problem within a brewery. It is shown that an almost integer solution is achieved. In section 3 we derive heuristics for the generalized assignment problem and give conditions and bounds for the existence and quality of feasible solutions. Section 4 is used to give some concluding remarks and some comments.

2. A location-allocation problem within a brewery

The mixed integer problem formulation that will be given in this section is based on location-allocation problems as they appeared within a Dutch brewery. A decision support system in which mixed integer problems have to be solved relatively often, was built to support decision making with respect to production, inventory and distribution structure. We will not describe the most sophisticated models that were used, but restrict ourselves to a simplified version, to illustrate the main ideas. A simplified model formulation might be based on a distribution network as described in Figure 1.
For a detailed problem description we refer to (1), (2), (3). Each brewery contains a number of production lines. Each production line can produce one or several of the products that are considered. The products have to be transported to the warehouses. These warehouses are located at different places in the country. The buyers, who are themselves wholesalers, have to be assigned to one of the warehouses. The buyer has to receive all his products completely from the warehouse he is assigned to. The demand of the buyers is supposed to be known. In the actual situation the forecasts for the demand were rather adequate for mid-and longterm planning. One of the problem formulations could be: Which warehouses, out of a given set of potential warehouses should be used in the (near)future, and how should the available buyers, be assigned to the warehouses such that, within capacity constraints of lines and warehouses, the total costs for production handling and transportation are minimal.

In fact the above problem is solved if one determines the amount of each product that has to be transported from each production line to each warehouse and if one determines the warehouse to which a buyer is assigned.

The model input was based on four lists with names P, L, W and B for products, lines, warehouses and buyers respectively. Based on these lists the relevant input data are given in table 1. In table 2 some additional, derived input data are given. The contents of most of the data described in table 1 and 2 will speak for itself. An exception might be $\text{EGINLP}(l,p)$ which contains equalization indices for handling differences between the (planned) performance of production lines. The model can now be given by defining the decision variables and constraints.
<table>
<thead>
<tr>
<th>Table</th>
<th>Name</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBP(b,p)</td>
<td>Demand (b,p)</td>
<td>The demand of buyer b for product p.</td>
</tr>
<tr>
<td>TCBW(b,w)</td>
<td>Transportation costs (b,w)</td>
<td>The transportation costs for one ton of product from buyer b to warehouses w.</td>
</tr>
<tr>
<td>HCPW(p,w)</td>
<td>Handling costs (p,w)</td>
<td>The handling costs for one ton of product p at warehouse w.</td>
</tr>
<tr>
<td>CAPLP(l,p)</td>
<td>Capacity(l,p)</td>
<td>The production capacity of line l with respect to product p.</td>
</tr>
<tr>
<td>UPCAPW(ω)</td>
<td>Upper capacity(ω)</td>
<td>The maximal capacity of warehouse ω.</td>
</tr>
<tr>
<td>LOCAPW(ω)</td>
<td>Lower capacity(ω)</td>
<td>The minimal capacity that should be used in warehouse ω.</td>
</tr>
<tr>
<td>THRUPW(p,w)</td>
<td>Throughput time(p,w)</td>
<td>The maximal throughput time of product p in warehouse ω.</td>
</tr>
<tr>
<td>CONVPW(p,w)</td>
<td>Conversion index(p,w)</td>
<td>The conversion index for space occupation of product p in warehouse ω.</td>
</tr>
</tbody>
</table>

Table 1: Input tables for the location allocation model

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPL(p,l)</td>
<td>Time required to produce one ton of product p on line l, after equalization.</td>
</tr>
<tr>
<td>CAPBW(b,w)</td>
<td>Capacity required to store the full demand of buyer b in warehouse w.</td>
</tr>
<tr>
<td>CPLW(p,l,ω)</td>
<td>Costs for production, handling and transportation of one ton of product p from line l to warehouse ω.</td>
</tr>
<tr>
<td>CBW(b,ω)</td>
<td>Cost for handling and transportation of the demand of buyer b from warehouse ω to buyer b.</td>
</tr>
</tbody>
</table>

Table 2: Some additional input tables for the location-allocation model
As indicated there are two types of decision variables.

\( TPLW(p, l, w) \) The amount of product \( p \) that is produced at line \( l \) for transportation to warehouse \( w \).

\( ASBW(b, w) \) a 0-1 assignment variable which is 1 when buyer \( b \) is assigned to warehouses \( w \) and which is 0 otherwise.

The restrictions that have to be satisfied are listed below:

\( BALPW(p, w) \) The balancing restrictions which guarantee that the amount of product \( p \) that is transported to \( w \) is equal to the demand for product \( p \) in warehouse \( w \).

\( CAPUL(l) \) The production capacity restriction for lines, which guarantee that (equalized) production capacity for line \( l \) is not violated.

\( CAPUW(w) \) The capacity restrictions for the warehouses, which guarantee that the maximum (minimum) capacity of warehouse \( w \) is not violated.

\( ASRB(b) \) The assignment restriction which guarantee that buyer \( b \) is assigned to just one warehouse.

The problem formulation can now be restated as the determination of the decision variables \( TPLW(p, l, w) \) and \( ASBW(b, w) \) such that the objective functions COST, which contains the total production, handling and transportation cost is minimized (see table 3).

<table>
<thead>
<tr>
<th>column</th>
<th>row</th>
<th>matrix element</th>
<th>relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPLW ((p, l, w))</td>
<td>BALPW ((p, w))</td>
<td>1</td>
<td>( TPL(p, l) )</td>
</tr>
<tr>
<td></td>
<td>CAPUL ((l))</td>
<td>( TPL(p, l) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COST</td>
<td>( CPLW(p, l, w) )</td>
<td></td>
</tr>
<tr>
<td>ASBW ((b, w))</td>
<td>BALPW ((p, w))</td>
<td>(-DBP(b,p))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPUW ((w))</td>
<td>( CBW(b, w) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPLW ((w))</td>
<td>(-CBW(b, w))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASRB ((b))</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COST</td>
<td>( CBW(b, w) )</td>
<td></td>
</tr>
</tbody>
</table>

| RHS | BALPW \((p, w)\) | 0 |
| | CAPUL \((l)\) | 1 |
| | CAPUW \((w)\) | \( CAPW(w) \) |
| | CAPLW \((w)\) | \(-LOCAPW(w)\) |
| | ASRB \((b)\) | 1 |

Table 3: Definition of the non-zero coefficients in the simplex tableau of the location-allocation model
The size of the problem follows from the number of decision variables and restrictions. The number of decision variables is equal to $P \times W \times L + B \times W$ where, $P, L, W$ and $B$ stand for the number of products, lines, buyers and warehouses respectively. The number of restrictions equals $P \times W + 2 \times W + L + B$.

Based on the above data we can state the following theorem.

**Theorem 1.** If one solves the linear programming relaxation of the above location-allocation problem, then for any basic feasible solution in which every warehouse stores every product we have that the number of non-unique assignments is less than or equal to the number of fully occupied production lines plus the number of fully occupied warehouses plus the number of exactly satisfied lower bounds for the capacity of the warehouses.

For a proof see (3).

Since, in general, the number of production lines and warehouses is small, compared with the number of buyers the linear programming relaxation will produce mainly feasible integer assignments. In the brewery example with over a 1000 buyers there remain between 3 and 15 non-unique assignments. So only a very few assignments remain to be done.

In the brewery problem, in many cases, an acceptable integer solution could be found by rounding off the non-integer fractions. Such a round-off procedure was however not acceptable in all situations. For a more complete description of the decision making processes to be supported, including the non-quantitative criteria to be taken into account, and of the Decision Support System in which LP-relaxation was used see (1), (2).

3. **Heuristics for the generalized assignment problem**

The problem that was described in the previous section is of the assignment type. In this section we will use the generalized assignment problem to illustrate some of the results with respect to the quality of heuristics for assignment type problems. Let us first describe the generalized assignment problem (GAP). Suppose that $n$ jobs have to be assigned to $m$ machines with restricted capacity, and suppose that it is not allowed to assign jobs to more than one machine. Let $a_{ij}$ be the required capacity when job $j$ is executed on machine $i$, and let $b_i$ be the capacity of machine $i$. Moreover we denote by $c_{ij}$ the cost of executing job $j$ on machine $i$. 
Let $x_{ij}$ be the 0-1 variable which equals 1 if job $j$ is assigned to machine $i$ and 0 if not. Now the GAP can be formulated as follows:

**Minimize**  
$$\sum_{i,j} c_{ij} x_{ij}$$

subject to

$$\sum_{i} a_{ij} x_{ij} \leq b_{i} \quad i = 1, 2, \ldots, m \quad (1)$$

$$\sum_{j} x_{ij} = 1 \quad j = 1, 2, \ldots, n \quad (2)$$

$$x_{ij} \in \{0, 1\} \quad i = 1, 2, \ldots, m \quad j = 1, 2, \ldots, n \quad (3)$$

It is well known that this problem is NP-hard, which means that large problems are difficult to solve, i.e., not with polynomial bounded algorithms see (6). Most of the methods developed until now, to solve the GAP-problem are based on a branch and bound method, where in each node of the decision tree the 0-1 dichotomy of the variables $x_{ij}$ is used. The differences between the various methods lies in how the bounds are calculated in each node and which variable is chosen to branch on. The general idea behind all these methods is to relaxe the constraints (1), (2), or (3) by removing one of them. Recent papers on solution procedures for the GAP are Ross and Soland (8), Martello and Toth (7), and Fischer, Jaikumar and Van Wassenhoven (4). In practical problems where the total number of jobs and machines will be large the methods all require large amounts of computation times. We will propose a heuristic that is based on the linear programming relaxation of the GAP. Where the relaxation is obtained if one replaces conditions (3) by $x_{ij} \geq 0$.

The solution of the relaxed problem yields a lower bound for the solution of the GAP. But the relaxed solution will contain a number of indices $j$ for which the variables $x_{ij} \neq 0$ for several $i$.

However, an upperbound on the number of splitted jobs can be given.

**Theorem 2.** If one solves the linear programming relaxation of the GAP then the number of non-unique assignments is less than or equal to the number of machines of which the capacity is used completely.

For a proof we refer to (3).
Since the number of machines is in general relatively small compared with the number of jobs, the solution of the relaxed problem will be a good starting point for a heuristic that assigns the remaining non-uniquely assigned jobs. Moreover, the quality of the resulting solution can be judged by comparing the corresponding costs with the value of the relaxed linear programming solution.

Whether a feasible integer solution exists or not depends mainly on the amount of (over)capacity that is available at the machines. Our heuristic uses the criterion of available overcapacity in a dynamic way. To illustrate the idea let us first consider a simplified generalized assignment problem with $a_{i,j} = a_j$ for all $i$. Now the overcapacity $oc$ of all the machines equals $oc = \sum_i b_i - \sum_j a_j$.

Clearly there will exist a feasible integer solution if $(m-1)a_1 \leq oc$ where it is supposed that the jobs are ordered according to the requested capacity, so $a_1$ is the biggest job, $a_2$ the second biggest job etc.

It is easy to see that $oc \geq (m-1)a_1$ is a sufficient condition for the existence of a feasible solution, since the overcapacity is larger than $(m-1)$ times the requested capacity for the biggest job. Problems that satisfy the above condition can be called easy problems. However, in general so much overcapacity will not be available. On the other hand once the biggest jobs are assigned less overcapacity will be sufficient to guarantee that a feasible integer solution exists. This idea is used in the following theorem.

**Theorem 3.** Let $\{x_{i,j} \mid i \in I, \ j \in J\}$ be a basic feasible solution of the linear programming relaxation of the GAP. Then there will exist a feasible integer solution of the GAP if $oc \geq (m-1)a_1^*$ where $oc$ denotes the overcapacity and $a_1^*$ is the capacity occupation of the biggest splitted job in the relaxed solution.

**Proof:** Once the jobs bigger than $a_1^*$ are assigned, the proof proceeds completely along the same lines as proving that $oc \geq (m-1)a_1$ is a sufficient condition for the existence of a feasible integer solution.

Let $J^*$ be the set of splitted jobs that have to be assigned after the solution of the relaxed problem. The jobs in $J^*$ are ordered according to requested capacity. If $oc \geq (m-1)a_1^*$, then there is a machine with sufficient overcapacity where job $a_1^*$ may be assigned. Proceeding along these lines gives that all subsequent jobs in $J^*$ can be placed, and the statement will be clear.

The above theorem shows that the requested overcapacity, to call a problem easy after solving the LP-relaxation, depends on the biggest non-uniquely assigned job in the actual solution.
If only small jobs remain to be assigned, than little overcapacity is required to have an easy problem. However one does not know in advance which jobs will be splitted in the relaxed problem. Very often it will be the big jobs that are splitted and the condition \((m-1)a^x\) will not be met. In the heuristic to be developed this problem is solved by allowing that already assigned small jobs are reassigned and to try to assign big jobs first. Let us be a little bit more precise with respect to the terms small and big jobs. Suppose in the \(t\)-th iteration step of the heuristic, not assigned big jobs have to be assigned to the available overcapacity. Let \(k_t\) be the set of small jobs defined by

\[
k_t = \{ j \mid a_j \leq z_t \}
\]

with \(z_t\) such that

\[
oc + \sum_{j \in k_t} a_j \geq (m-1)a^x
\]

and \(a^x\) the biggest non-assigned job in iteration step \(t\). So by ignoring the requested capacity for the small jobs we create enough "overcapacity" for the remaining problem with big jobs to be an easy problem. Once the big jobs are assigned \(k_{t+1}\) will be smaller than \(k_t\).

This leads to the following heuristic:

1. Solve the linear programming relaxation of the GAP, and determine the biggest splitted job \(a^x\).
2. Let \(z_t\) be the border between big and small jobs and choose \(z_t\) the smallest number such that

\[
oc \geq (m-1)a - \sum_{j \in k_t} a_j
\]

\[
oc \geq (m-1)z_t
\]

with

\[
k_t = \{ j \mid a_j \leq z_t \}
\]

Note: if \(k = \emptyset\), at most \((m-1)\) jobs have to be assigned.

3. Fixate assigned big jobs. Set the \(l\) biggest splitted jobs as integer variables. Set the small jobs as continues variables.

4. Solve the mixed integer problem as described in step 3 by using a SOS-option (where SOS stands for special ordered set, SOS options are available in most commercial L.P. packages). Go back to step 2 with \(t = t+1\) if some jobs are not uniquely assigned. Otherwise go to step 5.

5. Stop.
In several practical situations the proposed heuristic worked very well. Depending on the number of machines, \( z \) as used in step 3 was chosen equal to \( m \). In problems with a large number of machines \( z \) should be smaller than \( m \) so that in each iteration step only small integer problems have to be solved. In general the heuristic converged very fast. However some comments have to be made. In some examples oc \( \leq (m-1) a^* - \sum \lambda_j \) in this case the biggest jobs are assigned first and fixated there after. Once again we would like to emphasize that the quality of the achieved solutions can be measured by comparing it with the LP-relaxed solution.

4. Some extensions and comments

Although we only described the simplified example with \( a_i = a \), it is easy to define adjusted bounds \( z_c \) if the simplification does not hold. A similar heuristic as described in the previous section was defined for the location-allocation problem as described in section 2. In fact this paper treats 2 main topics. First we showed that assignment type mixed integer programming problems can be handled by considering the linear programming relaxation of the problem. Such problems appear very often in praxis. We mentioned already time table problems and a LP-gas distribution problem. However, many other practical problems have the same structure, see e.g. (5) where a catalogue space planning problem is considered. For several extensions of the brewery problem, which allow for inventory control and more period problems we were able to prove similar results. The lesson that we can draw from this is that in many practical situations it is worthwhile to try to solve very large integer problems just by considering a LP-relaxation of the problem. Often one can prove in advance that in the resulting basic solution of the relaxed problem the majority of values will be integer.

The second topic is concerned with heuristics for assigning the remaining non-uniquely assigned variables. There it appeared that for the brewery problem a heuristic that rounded off non-integer fractions was fully acceptable. For several large test problems, the heuristic as described in the previous section, performed very well. Some problems appeared to be typical scheduling problems if for example the overcapacity was smaller than the smallest job. However, the dynamic measure for the overcapacity proved to be a good guideline for classifying problems as easy or difficult.


