Minimising the number of tool switches with tools of different sizes.

Csaba Raduly-Baka[1], Timo Knuutila[2*] and Olli S. Nevalainen[2]
1) Elcoteq Design Center Oy, Joensuunkatu 13, FIN-24100 Salo, Finland
2) Department of information technology and TUCS, University of Turku, FIN-20014 Turku, Finland
*) Corresponding author: knuutila@cs.utu.fi, phone: +358-2-33386365, fax: +358-2-3338600

***Abstract*— In this paper we address the combined problem of job-ordering and tool placement, where each tool can occupy more than one slot of the primary storage magazine. The capacity of the magazine is limited so that all the tools neccessary in the production cannot fit into the magazine at the same time, and the cost of magazine reorganization depends linearly on the number of tool moves. Our task is to find the order of processing the jobs and the positions to put the tools in the magazine, so that the total cost of switching tools from one job to the next is minimized. We introduce a new heuristic for the problem. The algorithm hybridizes an efficient tool switching algorithm based on the matrix permutation problem and a novel two level storage management algorithm. We compare the proposed solution method to previous approaches from the literature. Our comparisons indicate that the new algorithm procudes results with costs almost a third of the costs produced by algorithms previously known in this field.**

***Keywords:* tool management; tool switching; two level storage management; flexible manufacturing; heuristics; combinatorial optimization**

I. Introduction

The problem of increasing the production efficiency in flexible manufacturing systems by computational means has become an important aspect of manufacturing with the arrival of computer numerically controlled (CNC) machinery. While the kind of machinery is an excellent means of increasing the degree of automatization in the production, it still leaves numerous questions open with respect to its optimal usage. Some of these deal with the tool management which has clear effect on the operating cost and the productivity of the manufacturing process.

In particular, we consider the case where a CNC machine processes a number of jobs with a number of changeable tools. It is imposed that each job requires some subset of tools for its processing and that the machine has a *primary* tool magazine with a lower capacity than the number of all different tools [2]. When proceeding to the next job, the tools used by the next job must placed in the magazine, if not already there. Because not all the tools fit into the tool magazine some of the tools in the magazine may have to be removed to make room for the tools of the new job.

Flexible manufacturing systems are used in various industries to achieve efficiency in low volume high-mix product manufacturing. One example is, in the electronics industry, the assembly of printed circuit boards (PCB). Each PCB contains a number of components, and different PCB types are assembled by the system [5]. One can, in this context, interpret the component types (stored in component reels, sticks, etc.) as ”tools” and the feeder unit as a ”magazine”.

Various aspects of the tool management have been discussed in literature. One aspect is the ordering of the jobs so that the number of required tool switches is minimized. This problem is also known as the *tool switching problem*. The problem is NP-hard even in the simplified case where all the tools are of the same size, and a number of heuristic solutions have been proposed in the literature [1]. The *tool-loading problem* considers a case where the job sequence is given, and the goal is to find a placement and removal order of tools, that the total cost of required tool changes is minimized. This problem can be solved optimally in the case of equal tool sizes using the KTNS (Keep Tool Needed Soonest) algorithm, as showed by Tang and Denardo [4].

For different tool sizes, the tool-loading problem becomes NP-hard as showed by Matzliach and Tzur in [6]. This problem is addressed in the literature also as the *Dynamic Storage Management Problem*, or *DSMP*. Because of different tool sizes, the storage area will become fragmented after processing a number of jobs, and the question arises how to place the tools to minimize the fragmentation. The online version of this problem also appears in operating systems, where the system memory is shared among various tasks [3]. The offline version has been discussed in the context of flexible manufacturing systems [7].

The *DSMP* has been addressed in literature as the two level storage management problem in [6] and [7]. In this problem there are two magazines holding tools, one with a limited capacity internal to the machine, the other with an unlimited capacity external to the machine. The primary (internal) magazine can hold only a limited number of tools, and if a required tool is not in the primary storage area the machine must pick it up from the secondary (external) storage area. This may also involve the removal of some tools from the primary storage area, see Matzliach and Tzur in [6] and Hirvikorpi *et al.* in [7].

The combined problem of job-ordering, tool placing and tool-loading problems has been studied previously only for equal tool sizes [9]. The problem is easy in the sense that the tool-loading and placing can be solved optimally for it. In real life manufacturing systems it is likely that the tool sizes are unequal, see [8] for a number of heuristics for solving the problem.

In this paper we consider a new approach to the job-ordering, tool loading and placement problem with unequal tool sizes. Our approach to the problem is similar to that described by Tzur and Altman in [8], except that we consider a different combination of job ordering and storage management algorithms. Our storage management algorithm is based on the heuristic by Hirvikorpi *et al.* in [7], with small variations required by the nature of this problem. For the job-ordering we modify the heuristic introduced by Djellab *et al.* in [9]. This heuristic has been used very successfully to optimize instances of job-ordering problem with equal tool sizes [11]. The original Djellab heuristic is based on the KTNS tool-loading heuristic which we replace with our own tool-loading and placement heuristic.

The rest of this paper is organized as follows. In Section 2 we formally specify the problem and its assumptions. In section 3 the Djellab tool switching heuristic is briefly described, a detailed description can be found in [9]. Our use of the Djellab method is neutral in the sense that it does not take any position on the (equal or unequal) sizes of the tools. In Section 4, a modified version of the storage management heuristic of Hirvikorpi *et al.* [7] is described. In Section 5 we give the combined heuristics for the tool switching problem with different tool sizes. Section 6 summarizes the results of numerical tests with the new algorithm, here we observe that the new algorithm outperforms existing solution by a factor of 3, for large problem instances. Section 7 contains concluding remarks on the subject.

II. Problem description

The problem of *Job ordering, tool Loading and tool Placement* with unequal tool sizes (*JLP* for further abbreviation) has the following parameters:

- N - the number of jobs to be processed.
- M - the number of different tools available.
- C - the capacity of the tool magazine (number of slots) used as the primary storage area.
- T - set of tools, where each tool $t \in T$ has a transfer cost $c_t \in \mathbb{R}^+$ which is the cost to move, remove or insert the tool into the magazine, and a size $s_t \in \mathbb{N}$ which is the number of slots occupied by the tool.
- A - a job-tool incidence matrix. The size of A is $N \times M$ and the element a_{ij} of A is 1 if the *i*th job requires the *j*th tool, and 0 otherwise.

The goal is to find an ordering of the jobs (job scheduling) and the placement and loading order of the tools, so that the total cost of switching the tools between jobs is minimized. This problem statement omits the lifetime limitations of the tools. We assume that the tools do not wear and therefore can reside in the magazine as long as they are needed.

The *JLP* problem is NP-hard as shown by Matzliach and Tzur in [6], even in the case where the order of the jobs has been fixed. The best known heuristic for this problem is the *Aladdin* heuristic proposed by Tzur and Altman in [8]. The *Aladdin* algorithm combines the job-ordering and the tool-loading problems into a single heuristic.

We propose an algorithm, in which a job-ordering algorithm is used to sequence the jobs and a two level storage management algorithm is used to minimize the tool switching cost of each job sequence. Tzur and Altman experimented in [8] with various two-stage heuristics, where the job ordering and tool switching stages were separated. Their tests indicated that two-stage approaches gave weaker results than a unified one-stage solution, namely the *Aladdin* algorithm. We will see that, by choosing the heuristics carefully, one can create an efficient two-stage approach. We also note that the choice of storage management heuristic is crucial: most of the job-ordering heuristics presented in [8] can be improved simply by a different choice of the tool storage management algorithm.

III. The Djellab-Gourgand tool switching heuristic for equal tool sizes

The novel heuristic algorithm proposed by Djellab and Gourgand in [9] uses a hypergraph representation of the tool switching problem. The algorithm solves a weighted version of the matrix permutation problem [11]. As presented by Djellab *et al.* in [9], the algorithm gives excellent results for the tool switching problem when the tool sizes are equal.

A major strength of the heuristic by Djellab *et al.* [9] is that it is able to consider also an initial partial order of the jobs. The fact that certain jobs can be processed only after other jobs is commonly ignored in the tool switching literature.

The Djellab-Gourgand heuristic consists of two major parts:

- Given a priority order in which the jobs are considered for insertion, create a job sequence so that the number of tool switches using the KTNS policy is minimized. This procedure is called the *Best Insertion* (*BI*) heuristic.
- In the second part, iteratively try to improve the results by generating random priority orders and using the BI heuristic with these priority orders. This procedure is called the *Iterative Best Insertion* (*IBI*) heuristic.

In our implementation we modified the Djellab-Gourgand heuristic to allow different tool sizes. This is necessary due to the fact that the KTNS policy which is used in the original algorithm assumes equal tool sizes in order to be optimal.

The reader is refered to [9] for a detailed description of the original Djellab-Gourgand heuristic. Basically we apply here the same algorithm with the difference that we do not use the KTNS policy for tool placement, but the *IMTM* storage management algorithm proposed in section 4.

IV. The storage management heuristic

The problem of storage management with different tool sizes and fixed job ordering has been proved to be NP-hard by Matzliach and Tzur in [6]. A number of algorithms have been proposed to solve this problem. In the case of moveable tool positions Matzliach and Tzur in [6] proposed an algorithm. The problem becomes even more complicated in the case when the tools cannot be moved unless involving the tool switching cost to each move. This problem has been recently addressed by Hirvikorpi *et al.* in [7] and they give heuristics to solve the problem.

We suppose in the storage management problem (with different tool sizes and magazine reorganization costs) that there are N jobs to be processed in a predefined order, and the jobs use in total M different tools. The task is to determine a tool switching and placement strategy, which minimizes the total cost of tool changes. Tool t consumes s_t slots of the magazine, and the cost of removing the tool t into the magazine is c_t. The $N \times M$ binary matrix A tells whether the tool t is used ($A(j,t) = 1$) or not ($A(j,t) = 0$) in a job j. The capacity of the tool magazine is C slots.

In the present paper we implement a heuristic based on the *SMMT-2* heuristic introduced by Hirvikorpi *et al.* in [7]. We name this algorithm *Iterative Multi-Tool Manager* or *IMTM* algorithm. In our implementation we have improved for efficiency reasons the method *SMMT-2* uses for choosing tools for removal. This was necessary because storage management decisions are made repeatedly in the *IBI* procedure (of the Djellab-Gourgand heuristic) a great number of times and they thus become a bottleneck of the joint solution algorithm of the *JLP* problem. As to the results produced by the new heuristic and that of [7], according to our evaluations the solutions are of similar quality, but the new heuristic runs about 190 times faster.

A. *Removal cost matrix*

When cleaning the magazine for new tools, the decisions on which tools to remove are central to the *IMTM* algorithm. In order to save computation time, we precompute a matrix R of size $N \times M$, containing the removal cost for each tool of each job. Whenever the algorithm has to decide which tool to remove at a certain job, it uses the removal cost matrix to choose the tool.

The elements of R are calculated in a way that resembles how the KTNS rule is used to choose the tools to be removed, but now the removal costs are taken into consideration. For each job j and tool t let $dist(j,t)$ be the distance to the next job using the tool. If job j uses this tool, the distance is 0, otherwise it is the number of jobs until the tool is used again. Formally,

$$dist(j,t) = \begin{cases} 0 & \text{if } A(j,t) = 1 \\ m & \text{if } A(j+m,t) = 1 \text{ and } A(j+i,t) = 0 \\ & \text{for all } i \in [0,m), \text{ and } m > 0 \\ N & \text{if } A(j+i,t) = 0 \text{ for all } i \in [0, N-j) \end{cases}$$

If the tool is not used in any of the forthcoming jobs, the distance is the number of jobs N, so that no other (used) tool can have this large distance. The tool removal costs (R) are then given by the following formula which weights the tool removal costs by their distance to the next use. In this way we reuse the principle of the KTNS algorithm for tools of different sizes.

$$R(j,t) = \begin{cases} c_t/dist(j,t) & \text{if } dist(j,t) > 0 \\ \alpha \cdot c_t & \text{if } dist(j,t) = 0 \end{cases}$$

Here α is an arbitrary constant greater than 2, to give the highest removal cost for tools used by the current job (with distance 0). The storage management algorithm uses the R matrix when searching for the block of consequtive magazine slots with the lowest removal cost, to make room for new tools. Naturally R is only one (heuristic) measure for favourable actions to be taken when solving the storage management problem.

B. *Outline of the* IMTM *algorithm*

The *IMTM* algorithm builds a content of the tool magazine for each job by considering the magazine from the previous job, and inserting the tools needed by the job but not already in the magazine.

The tools are always inserted one by one, into the smallest available space they can fit. If the magazine does not contain enough empty space, the algorithm removes from the magazine some tools, which are not needed by the current job, to make room for the new tool. The choice of the tools to be removed is made according to the lowest removal cost value defined by R.

If no room can be made by removing unused tools, the algorithm removes from the magazine a tool which is required by the current job, and restarts, trying to insert the new tools (including the removed one) needed by the current job. This last step will cause a rearrangement of the tools in the magazine.

The algorithm tries to build a magazine with limited capacity for the current job using the tool magazine setting of the previous job. The algorithm iterates through all jobs and applies a magazine building procedure for each job. The magazine is considered to be empty before the first job.

C. *Inserting in a free area*

The storage management algorithm first looks for available empty spaces in the magazine for the new tool. This is done by the *InsertFreeArea* procedure. Given the size of the tool to be inserted, the *InsertFreeArea* searches the magazine to the shortest sequence of free slots, where the sequence contains at least as many slots as required by the tool size.

If such an area is not found, the storage management proceeds with the next step by trying to insert the new tool over some removable tools.

D. *Inserting in a removable area*

In order to insert a new tool over some removable tools, the algorithm must choose which tool can be removed. This choice is made according to the removal cost values stored in the R matrix.

Function

```
InsertRemovableArea(magazine, job, tool, ts): Boolean
```

implements the insertion of a new tool `tool` with size `ts` into slots containing removable tools using `magazine` (an array of slots) to store the tools. The algorithm differentiates between tools which are used by the current job `job`, and tools which are not (and can thus be removed). The implementation separates the magazine space into removable and non-removable areas. Given a tool with size `ts` and a removable area containing enough slots for the tool, the algorithm searches for a sequence of `ts` slots with a lowest total removal cost. This is done by checking the removal cost by the aid of our matrix `R` for each possible insertion location of the new tool.

Procedure `InsertRemovableArea` uses function

```
FindMinimumCostPlace(
    magazine, job, start, end, ts, cost): int
```

to find the lowest cost insertion slots in a removable area. Here `start` is the start of the area where insertion is considered, `end` is the last slot where insertion can occur, and `cost` is the cost of making free space at the returned position.

Tool removal costs are computed with function

```
GetToolRemovalCost(magazine, job, pos, len): int
```

which calculates the removal cost of tools stored in `len` consecutive slots starting at `pos`. This is done by simply summing up the values `R[job, t]` where `t` ranges from `pos` to `pos + len - 1`. Routine `FindMinimumCostPlace` applies `GetToolRemovalCost` for each possible starting position, and returns the one with the minimal cost.

The implemementation of the main insertion routine is illustrated in the pseudocode below.

```
InsertRemovableArea(magazine, job, tool, ts): Boolean
pos = 0 // index of the slot where insertion can occur
cost = 0 // current cost of insertion at pos
for j = 1 to C
   if tool at magazine[j] is not used
      next = index of the first used tool after j
      i = FindMinimumCostPlace(
              magazine, job, start, next - 1, ts, c)
      // c is the cost of inserting the tool at position i
      if i > 0 and c < cost then // suitable for insertion
        set pos to i and cost to c
end end end
if pos > 0 then // there is a slot where we can insert
   remove tools from magazine[pos...pos + ts - 1]
   store tool at magazine[pos]
end
return (pos > 0)
```

Algorithm `InsertRemovableArea` can thus insert tools anywhere in a removable area, even into the middle of it. This means that the magazine can become fragmented after a number of tools have been placed over removable tools. This problem is somewhat handled by considering the tools in their decreasing order of size. Nevertheless, the algorithm might fail to insert the tool, and control is returned to a higher level in the storage management algorithm where we reconsider the tool insertion into a fragmented magazine space.

E. *Inserting multiple tools*

Let us now consider a higher level of hierarchy of the *IMTM* algorithm. As stated in the outline of the storage management algorithm, a number of new tools are inserted into the magazine for each job. This is done by function

```
InsertMultiTool(magazine, job, T): Boolean
```

which does the insertions tool by tool. The algorithm first tries to insert using `InsertFreeArea`, and if that fails then with `InsertRemovableArea`.

If both of these attempts fail for the current job (no place for new tools), `InsertMultiTool` returns ’false’ and the *IMTM* storage management algorithm tries to rearrange the used tools using function

```
IteratedInsertMultiTool(magazine, job, T): Boolean
```

This routine considers the tools used by the current job, and the ones inherited from the previous magazine in the increasing order of their removal costs. It removes at each step a tool from the magazine and inserts it in the set of new tools T (the ones not in the magazine but needed by the current job) which will be passed to routine *InsertMultiTool* again.

```
IteratedInsertMultiTool(magazine, job, T)
prevMagazine = magazine
if not InsertMultiTool(magazine, job, T) then
   MT = set of tools from the magazine needed by job
   sort MT by increasing order of removal cost from R
   for j = 1 to |MT|
      tool = MT[j]
      remove tool from prevMagazine
      add tool to T // set of new tools
      magazine = prevMagazine
      if InsertMultiTool(magazine, job, T) then
         return
   end end
   //insert T as one block at the start
   clear magazine
   insert T into magazine
end
```

After moving the tool from the magazine into the set of new tools, `IteratedInsertMultiTool` employs again the `InsertMultiTool` procedure to insert the set of T tools into the magazine. This loop is iterated on until all the tools have been inserted, or all the needed tools have been removed from the magazine, and they are all considered as new tools.

It is easy to see that the `IteratedInsertMultiTool` procedure cannot fail. If there is no possibility to insert the new tools into the magazine, all the tools needed by the current job will end up in T. These are either inserted into the lowest removal cost places (as the last step of the for loop) or the magazine is emptied and the tools are inserted as one block. The *IMTM* algorithm employs the `IteratedInsertMultiTool` procedure to insert the new tools needed for the current job into the magazine.

F. *Generating the magazine set-up for each job*

At the highest level of the algorithm hierarchy, the storage management algorithm iterates over the set of `N` jobs, and builds a magazine set-up for each job using the algorithm

```
InsertJobTools(magazine[], T, N)
```

which returns as the output an array of magazine set-ups. This is used to compute the total cost of tool switching for a particular sequence S of the jobs.

```
InsertJobTools(magazine[], T, N)
for j = 1 to N
  Tj = tools needed by job j.
  magazine[j] = magazine[j - 1]
  IteratedInsertMultiTool(magazine[j], j, Tj)
end
```

G. *Comparison to the* SMMT-2

Altough the storage management algorithm described above is based on the ideas introduced by Hirvikorpi *et al.* in [7], there are several fundamental differences between this algorithm and the *SMMT-2* algorithm of Hirvikorpi. Both algorithms produce results of similar quality, while the new algorithm does that about 200 times faster in similar conditions on the same data set. This allows us to use it as a storage management algorithm embedded into the Djellab-Gourgand job ordering heuristics.

The main differences between the *IMTM* algorithm and the *SMMT-2* algorithm presented by Hirvikorpi *et al.* in [7] are the following.

The *SMMT-2* algorithm tries to insert all the tools for a job in a single block. This might succeed in some cases, but for the upcoming jobs it might cause additional fragmentation.

The *SMMT-2* algorithm considers tools to be inserted in a random order. In the *IMTM* algorithm we consider tools in the decreasing order of their size, placing first tools with larger size.

However, the main difference lies on how tools for removal are chosen. The *SMMT-2* chooses the lowest removal cost tool to be removed, and then tries to place a new tool in its place. By removing the tools with the lowest removal cost, one does not always produce enough space for a new tool, and then other tools have to be removed. This causes problems especially when, for example, the two lowest cost removable tools are not in adjacent slots, so their removal does not free up relevant space. This means that we do not always remove the tool with the lowest removal cost.

When the *SMMT-2* algorithm runs out of possibilities of placing the new tools, it will try to place them as a single continuous block, relying on a less efficient algorithm. In the *IMTM* algorithm we remove a tool which is in the magazine from the previous job, and then try the same algorithm to insert the tools required for the current job and not in the magazine.

In *SMMT-2* the tool removal cost is calculated in every step, which adds up to the complexity of the algorithm. In the *IMTM* algorithm we use a precomputed matrix of tool removal costs.

V. The combined heuristics

The *Best Insertion* (*BI*) routine of the job ordering algorithm by Djellab *et al.* in [9] does not utilize any knowledge of tool sizes. It is practically a gap minimization algorithm, which takes the job/tool incidence matrix, and searches for a job permutation, for which the number of horizontal gaps is minimized. These gaps represent tools which are removed from the magazine to make room for other tools. We use the *Best Insertion* routine as such in the *IMTM* algorithm.

The *Iterative Best Insertion* (*IBI*) algorithm [9] relies on the KTNS algorithm to evaluate the cost of job permutations found by the *Best Insertion* algorithm. In our implementation, for ordering jobs with unequal tool sizes, we replace the KTNS algorithm with the *IMTM* algorithm. We name the new algorithms with the replaced storage management *Best Insertion** and *Iterative Best Insertion** respectively. In this way the *Iterative Best Insertion** algorithm will return a job sequence which has the lowest cost according to the *IMTM* algorithm, from all sequences found by *Best Insertion**. Because the *IMTM* algorithm is used at each step in the iteration of the *Iterative Best Insertion** algorithm, the speed of the *IMTM* algorithm is crucial. Our evaluation showed that the *IMTM* algorithm is fast enough so that the Djellab-Gourgand heuristic remains usable in what comes to the time consumption.

Besides the replacement of the KTNS algorithm by the *IMTM* algorithm, all the other aspects of the job ordering remain as described by Djellab *et al.* in [9]. We name the new job ordering algorithm using tools with unequal sizes *DG+*.

TABLE I
Problem instances

N	M	Min	Max	C
10	10	2	4	12, 15, 20, 25
15	20	2	6	18, 25, 30, 35
30	40	5	15	45, 50, 55, 60
40	60	7	20	60, 65, 70, 75

VI. Computational results

We compared the *DG+* algorithm presented in this paper to the Aladdin algorithm introduced by Tzur and Altman in [8]. We used the original implementation of the Aladdin algorithm, which was kindly provided to us by Dr. M. Tzur.

The Aladdin algorithm evaluation in [8] counts the number of tool switches instead of the cost of these switches. Indeed, in some of the manufacturing environments, the impact on the manufacturing cost is the number of switches (steps to refill the magazine), and not proportional to the size of the tools. In our comparison we used this switch counting method both for Aladdin (as it was originally) and the *DG+* algorithm.

The original Aladdin evaluation in [8] used random tool size sets for each instance. There were 10 instances for each job/tool number configuration, and there were a total of 4 job/tool configurations as follows: (10, 10), (15, 20), (30, 40) and (40, 60). Each job/tool configuration was tested with 4 different magazine capacities. We fixed the tool size distributions in our evaluation for both the Aladdin and for the *DG+* algorithm. Practically this means that we gave the same set of tools as an input for both the Aladin algorithm and the *DG+* algorithm. Table I summarizes the problem instances used in our test. The problem types are characterized by the following parameters:

- *N* - The number of jobs to be processed.
- *M* - The number of tools used to process these jobs.
- *Min* - The minimum number of tools used by a job.
- *Max* - The maximum number of tools used by a job.
- *C* - The capacity of the tool magazine.

It is important to note, that the comparison does not take account the initial tool setup. This means, that practically only the tool removals are counted when the number switches are evaluated.

In our test we used various tool size distributions, of 3 different tool sizes, occupying 1, 2 or 3 slots. The tool size frequency is indicated in each table. For example (1/3, 1/3, 1/3) means that each tool size has been used in equal proportion. The tables (II, III, IV, V) containing the test results are organized as follows. The first column contains the type of the

TABLE II

Results for (1/3, 1/3, 1/3) tool size frequencies

Instance		Switching cost		Running time	
Type	C	Aladdin	DG+	Aladdin	DG+
(10, 10, 2, 4)	12	13.100	4.300	0.150	0.030
(10, 10, 2, 4)	15	7.600	2.300	0.200	0.020
(10, 10, 2, 4)	20	0.600	0.700	0.290	0.020
(10, 10, 2, 4)	25	0.000	0.000	0.350	0.030
(15, 20, 2, 6)	18	43.100	12.300	0.721	0.110
(15, 20, 2, 6)	25	29.100	6.000	0.701	0.110
(15, 20, 2, 6)	30	12.500	3.200	0.711	0.110
(15, 20, 2, 6)	35	1.800	1.200	0.751	0.120
(30, 40, 5, 15)	45	243.200	62.600	1.892	1.832
(30, 40, 5, 15)	50	226.800	48.100	1.852	1.793
(30, 40, 5, 15)	55	201.400	35.800	1.892	1.763
(30, 40, 5, 15)	60	168.700	25.100	1.912	1.642
(40, 60, 7, 20)	60	472.100	137.900	4.736	5.999
(40, 60, 7, 20)	65	451.500	116.200	4.506	5.958
(40, 60, 7, 20)	70	437.700	99.600	4.536	6.029
(40, 60, 7, 20)	75	421.500	83.800	4.426	5.918

TABLE III

Results for (1/5, 1/5, 3/5) tool size frequencies

Instance		Switching cost		Running time	
Type	C	Aladdin	DG+	Aladdin	DG+
(10, 10, 2, 4)	12	15.400	5.500	0.550	0.051
(10, 10, 2, 4)	15	9.700	3.700	0.450	0.050
(10, 10, 2, 4)	20	2.100	1.800	0.500	0.040
(10, 10, 2, 4)	25	0.000	0.000	0.490	0.030
(15, 20, 2, 6)	18	48.700	17.400	0.751	0.130
(15, 20, 2, 6)	25	37.700	10.300	0.871	0.130
(15, 20, 2, 6)	30	27.100	7.200	0.901	0.121
(15, 20, 2, 6)	35	17.900	4.200	0.851	0.130
(30, 40, 5, 15)	45	265.100	84.800	1.952	2.063
(30, 40, 5, 15)	50	249.900	71.300	1.942	2.013
(30, 40, 5, 15)	55	232.100	58.700	1.912	2.053
(30, 40, 5, 15)	60	207.900	45.900	1.952	2.003
(40, 60, 7, 20)	60	516.200	184.600	4.706	6.709
(40, 60, 7, 20)	65	491.400	163.800	4.636	6.970
(40, 60, 7, 20)	70	474.300	144.100	4.626	7.031
(40, 60, 7, 20)	75	457.600	125.400	4.606	7.020

TABLE IV

Results for (1/5, 3/5, 1/5) tool size frequencies

Instance		Switching cost		Running time	
Type	C	Aladdin	DG+	Aladdin	DG+
(10, 10, 2, 4)	12	10.900	4.100	0.460	0.030
(10, 10, 2, 4)	15	5.100	2.000	0.450	0.030
(10, 10, 2, 4)	20	0.000	0.000	0.460	0.040
(10, 10, 2, 4)	25	0.000	0.000	0.480	0.030
(15, 20, 2, 6)	18	44.100	14.200	0.731	0.130
(15, 20, 2, 6)	25	27.500	7.200	0.711	0.120
(15, 20, 2, 6)	30	13.200	3.700	0.741	0.111
(15, 20, 2, 6)	35	2.900	1.300	0.731	0.130
(30, 40, 5, 15)	45	247.700	62.300	1.872	1.923
(30, 40, 5, 15)	50	217.700	47.600	1.802	1.942
(30, 40, 5, 15)	55	185.900	35.900	1.822	1.833
(30, 40, 5, 15)	60	148.100	26.300	1.882	1.753
(40, 60, 7, 20)	60	485.500	140.000	4.376	6.419
(40, 60, 7, 20)	65	464.800	120.300	4.276	6.399
(40, 60, 7, 20)	70	439.300	104.100	4.236	6.389
(40, 60, 7, 20)	75	405.500	87.700	4.236	6.390

TABLE V

Results for (3/5, 1/5, 1/5) tool size frequencies

Instance		Switching cost		Running time	
Type	C	Aladdin	DG+	Aladdin	DG+
(10, 10, 2, 4)	12	6.200	1.700	0.450	0.030
(10, 10, 2, 4)	15	0.600	0.600	0.450	0.020
(10, 10, 2, 4)	20	1.600	0.000	0.470	0.020
(10, 10, 2, 4)	25	0.000	0.000	0.470	0.030
(15, 20, 2, 6)	18	36.400	8.000	0.741	0.120
(15, 20, 2, 6)	25	10.700	2.800	0.721	0.110
(15, 20, 2, 6)	30	0.500	0.500	0.741	0.100
(15, 20, 2, 6)	35	0.000	0.000	0.751	0.121
(30, 40, 5, 15)	45	206.200	31.200	1.862	1.642
(30, 40, 5, 15)	50	150.400	17.800	1.832	1.542
(30, 40, 5, 15)	55	72.400	8.500	1.932	1.412
(30, 40, 5, 15)	60	11.900	2.900	2.072	1.312
(40, 60, 7, 20)	60	433.100	83.400	4.105	5.548
(40, 60, 7, 20)	65	409.700	66.100	4.055	5.258
(40, 60, 7, 20)	70	385.600	50.500	3.995	5.137
(40, 60, 7, 20)	75	313.500	36.600	4.075	4.927

problem instance. The second column contains the capacity of the magazine. The third and fourth columns contain the tool switching costs computed by Aladin and *DG+* respectively. The fifth and sixth column contain the running times for Aladin and *DG+* respectively.

Tables 2, 3, 4 and 5 present the comparison of results between the Aladdin algorithm and the *DG+* algorithm. There are 3 different tool sizes, tools occupying 1, 2 and 3 magazine slots. The frequency array specifies the number of tools with a given size. The running time of the algorithms was evaluated in milliseconds.

It is observed that DG+ outperformed Aladdin in most of the cases. In small problem instances the results are similar (in one case better for Aladdin). The DG+ algorithm performed excellently in cases of large instances.

VII. Conclusions

We introduced a new heuristic approach for the combined job scheduling, tool loading and tool placement problem, with unequal tool sizes. The main idea behind this algorithm is to use an efficient storage management algorithm with a previously known job ordering algorithm [9].

The new storage management algorithm is based on the ideas introduced in [7], but here we applied a different policy on how tools are removed, and how the fragmented magazine is rearranged. The combination of these algorithms has not been addressed previously in the literature.

We compared the results of the new algorithm with the results of the Aladdin algorithm introduced in [8]. We found that combining high performance heuristics from both job scheduling and tool placement problems resulted in good quality and time performance. Our results indicate that this algorithm makes a remarkable improvement over previously known approaches. Further consideration of tool wearing could be added to the problem statement of *JLP* problem.

A further study could investigate the combination of other job scheduling algorithms with the *IMTM* algorithm and evaluate the result provided by the combination.

VIII. Acknowledgement

We are grateful to Dr. Michal Tzur for providing us with the implementation of the Aladdin algorithm.

References

[1] Y. Crama and J. van de Klundert, The approximability of tool management problems, Technical Report, Maastricht Economic Research School on Technology and Organizations (1996).

[2] Y. Crama and J. van de Klundert, The worst-case performance of approximation algorithms for tool management problems. Naval Research Logistics, 46, pp. 445-462 (1999).

[3] A. S. Tanenbaum, Modern operating systems, Prentice Hall Ic., 2nd edition, 200-201, 2001.

[4] C.S. Tang and E.V. Denardo, Models arising from a flexible manufacturing machine, part I: Minimization of the number of tool switches, Operations Research, Vol. 36, No. 5, pp. 767-777 (1988).

[5] M. Johnsson, Operational and tactical level optimization in printed circuit board assembly, PhD thesis, University of Turku (1999).

[6] B. Matzliach and M. Tzur, Storage management of items in two levels of availability, European Journal of Operational Research (2000) 121 pp. 363-379.

[7] Mika Hirvikorpi, Kari Salonen, Timo Knuutila, Olli Nevalainen, General Two Level Storage Management Problem - A reconsideration of the KTNS-Rule, European Journal of Operational Research (in print).

[8] M. Tzur and A. Altman, Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes, IIE Transactions (2004) 36, pp. 95-100.

[9] H. Djellab, K. Djellab, M. Gourgand, A new heuristic based on the hypergraph representation for the tool switching problem. International Journal of Production Economics, (2000) 64, pp. 165-176.

[10] L.T. Kou, Polynomial complete consequtive information retrieval problems, SIAM Journal of Computing 6 (1) (1997) pp. 67-75.

[11] K. Salonen, Cs. Raduly-Baka, O. Nevalainen, A note on the tool switching problem of a flexible machine, Special Issue of Computers and Industrial Engineering, Selected papers from 32nd ICC&IE in Limerick, (to appear).