EVALUATION OF A NEW DECISION-AID PARAMETER FOR JOB SHOP SCHEDULING UNDER UNCERTAINTIES

Zakaria YAHOUNI\textsuperscript{1}, Nasser MEBARKI\textsuperscript{1} and Zaki SARI\textsuperscript{2}

\textbf{Abstract.} This paper addresses the groups of permutable operations method. This method is a flexible scheduling approach to hedge against uncertainties and is composed of a proactive/reactive phase. The proactive phase consists of computing a set of solutions (schedule) to a scheduling problem, leaving the choice of executing one of these solutions during the reactive phase according to the current state of the shop floor. During the reactive phase, the remaining decisions have to be made in real-time. The \textit{worst-case} evaluation of the remaining solutions is a decision-aid parameter used during the reactive phase in order to control the final schedule from exceeding a \textit{worst-case} performance. While the existing literature only tackles the \textit{worst-case} evaluation of the groups of permutable operations, this paper deals with its \textit{best-case} evaluation. For solving this problem, a new lower bound calculating this parameter in polynomial time is proposed. The computational efficiency of this parameter in a reactive algorithm exhibits very good performance. Moreover, the experiments show the robustness of this evaluation allowing this parameter to be used in an unstable job shop environment.

\textbf{Keywords:} Scheduling, Decision-aid system, Lower bounds, Job shop, Makespan

\textbf{Mathematics Subject Classification.} 90B35, 90B36, 68M20

\textsuperscript{1} LS2N, Laboratoire des Sciences du Numérique de Nantes, UMR CNRS 6004, Nantes, France; e-mail: firstName.LastName@ls2n.fr
\textsuperscript{2} MELT, Manufacturing Engineering Laboratory of Tlemcen, Algeria; e-mail: z.sari@mail.univ-tlemcen.dz

\copyright EDP Sciences 2001
1. Introduction

Most of the classical scheduling studies assume that all data and parameters of the problem are fully known. However, in practice, manufacturing systems are not so deterministic. During the execution of the offline generated schedule, many disturbances may occur, like arrival of new jobs, uncertain task duration, machine breakdown, etc., [9, 20, 31]. These disturbances will in most cases, deteriorate the expected performances.

This issue has received considerable critical attention over the past years, [5, 10, 26, 27] are among who first considered this problem. To address this issue, different approaches have been proposed in the literature. These approaches have been classified by [20, 30, 43] into three main classes: proactive, reactive and proactive-reactive approaches.

The proactive approaches rely generally on estimating and anticipating the perturbations before executing a schedule [23]. One of the most known proactive methods in project scheduling is the classical critical chain method used in several project management pieces of software [28]. This method introduces slacks on the operations of the critical path to absorb the delays caused by the perturbations. Another proactive method is just-in-case scheduling [45]. This method identifies the activities most likely to fail based on the available uncertainty information. The weakness of this kind of approaches is that it deals with a prediction of the system instead of the current state of the shop.

The second class encompasses the reactive approaches. Over the past years, much effort has been devoted to developing reactive methods that use generally a real-time control heuristics [19, 35, 46, 47]. These real-time control methods build dynamically the schedule based on the current state of the system [18]. Although, these reactive methods consider the current state of the shop floor contrarily to the proactive methods, their performance is good on a very high disturbed environment and generally poor on a low disturbed environment [13].

The third class combines the proactive and reactive approaches. The work of [43] addresses the relative importance of such combination to hedge against uncertainties. The basic idea of these approaches is to develop during the offline phase of the scheduling process, a flexible schedule with some degree of freedom, allowing during the online reactive phase (execution phase) to revise and adapt the schedule according to the new state of the shop.

The right choice of the scheduling approach depends on the scheduling environment and some assumptions that are taken into account when analyzing manufacturing systems [49]. The proactive-reactive approaches seem to be the most effective approaches to absorb production uncertainties while maintaining a good performance [48]. These approaches focus on the proper combination between a proactive flexible method and a reactive algorithm capable of revising the flexible schedule without deteriorating too much the expected performances. Some of these methods can be found in [8, 25, 32, 37, 40, 48]. Groups of Permutable Operations (GoPO) is one of the most studied proactive-reactive methods [2, 3, 9, 13, 22].
This method was created by LAAS-CNRS laboratory of Toulouse, France [21] and is implemented in an industrial software [41]. It is composed of two phases:

- A proactive phase which aims at computing a flexible solution offline. This solution is represented by groups of permutable operations such that each permutation between two operations gives a feasible schedule.
- A reactive phase in which the proactive schedule is processed online on the shop floor. This phase consists of choosing during the execution of a GoPO solution, the order of operations to be executed in each group of permutable operations. The decision of ordering the operations in each group can be chosen either using a reactive algorithm or by a human, named the operator. The schedule executed during this phase is called realized schedule.

This flexible method has an interesting property; once the groups are generated during the proactive phase, the worst schedule called the worst-case (which represents the worst permutation in each group), can be computed in polynomial time [1, 2, 22]. This allows to guarantee a minimal quality of the realized schedule (after taking all the decisions). Due to this property, the worst-case can be used during the reactive phase of GoPO in a decision-aid algorithm/system in order to schedule the operations inside each group according to the current state of the shop. The literature review has primarily focused on this parameter. However, making a decision during the reactive phase of GoPO using only the worst-case parameter can lead prioritizing lower quality solutions.

In this paper, we raised this issue by proposing a new parameter that can be used for the reactive phase of GoPO. This parameter consists of evaluating the best-case performance which corresponds to the best possible permutation in each group, such that the final realized schedule is an optimal solution of GoPO. The literature research has not focused on this parameter (best-case) as its computation is NP-hard and can be too time-consuming to be used in real-time in a decision-aid system. But a lower bound of this parameter should be of good interest for different reasons:

- Associated with the worst-case value, it permits to better represent the quality of GoPO as denoted by [22]. This quality can be measured by a range of all possible performances [worst-case ... best-case].
- It also permits to know, once the groups are generated during the proactive phase, the best achievable performance in case of no disruptions. For example, if the scheduling objective is to minimize the tardiness, it permits to know if there is no schedule that has no late job which can be of great interest for the decision maker.
- Most importantly, it permits to know, during the execution of a GoPO solution (reactive phase), the next optimal (or near-optimal) operation to be chosen in each group of permutable operations.

Two main contributions are presented in this work: the first one concerns the computation of the best-case of GoPO. As this problem is challenging, an efficient
polynomial lower bound for the *best-case makespan* performance has been proposed. Three steps are used for the computation of this new lower bound: first, we adapt the literature findings to GoPO. Then, this lower bound is tightened using the groups precedence property and the last step is the improvement of this lower bound using a precedence constraint property between the operations. The second contribution of the work presented in this article concerns the study of the usefulness of the *best-case* parameter in a decision-aid system. For this, the *best-case* parameter has been implemented in a decision-aid reactive algorithm and experimented in both deterministic and non-deterministic environments. This experimental implementation of the *best-case* parameter exhibits good performance.

The rest of the article is structured as follows: in section 2 and 3, GoPO is described and the problematic of the presented work is discussed. In section 4, the lower bound for the *best-case* of GoPO is presented. Section 5 is devoted to the reactive phase of GoPO where a new reactive algorithm using the *best-case* parameter has been evaluated on well-known instances of the job shop problem. In the same section, the reactive decision-aid algorithm which uses the *best-case* parameter is studied on both deterministic and non-deterministic environments. Finally, main conclusions are summarized in the last section.

2. Groups of Permutable Operations

We consider the job shop problem with release dates and precedence constraints \((J/r_{1},\text{Pred}/f\) according to the classification of [29]) where we have \(n\) jobs \(J_1, J_2, ..., J_n\) to be processed on \(m\) machines \(M_1, M_2, ..., M_m\), each machine can treat only one operation at a time. Job \(i\) consists of \(n_i\) operations. Associated with every operation \(O_i\): a machine allocation \(\mu_i\), a release date \(r_i\), a processing time \(p_i\) and a completion time \(C_i\). \(\Gamma_i^{-}\) and \(\Gamma_i^{+}\) denote respectively the predecessor and the successor of a given operation. Generally, the job shop problem uses a regular objective function \(f\) that is a non-decreasing function of the \(C_i\). In this work, we focus on the *makespan* objective \(C_{\text{max}} = \max(C_i)\).

GoPO is defined as a sequence of groups (of permutable operations) on each machine to be performed in a particular given order. The group containing operation \(O_i\) is denoted \(G_i\). Every group contains one or many operations that can be executed in an arbitrary order. On a given machine, the group after (resp. before) the group containing the operation \(O_i\) is denoted by \(G_i^{+}\) (resp. \(G_i^{-}\)). A GoPO schedule is feasible if for each group, all the permutations among all the operations of the same group give a feasible schedule (i.e. a schedule which satisfies all the constraints of the problem).

To illustrate this problem, let us study a job shop example of three jobs and three machines as described in table 1. Fig. 1 presents a feasible GoPO solving this problem. This GoPO is made of seven groups: two groups of two operations and five groups of one operation. The first machine is composed of two groups, the group \(G_6\) containing \(O_6\) and its predecessor group \(G_5^{-}\) containing the
permutable operations \( O_1 \) and \( O_7 \). To generate these groups of permutable operations, a greedy algorithm (described in [22]) may be used for the construction of the groups. This algorithm starts with an initial schedule with one operation per each group (this initial schedule is usually generated using simple heuristics). Then, the algorithm tries to merge each two successive groups according to different criteria\(^1\) until no group merging is possible.

Table 1. Example of a job shop problem

<table>
<thead>
<tr>
<th>( J_i )</th>
<th>( J_1 )</th>
<th>( J_2 )</th>
<th>( J_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O_i )</td>
<td>( O_1 )</td>
<td>( O_2 )</td>
<td>( O_3 )</td>
</tr>
<tr>
<td>( p_i )</td>
<td>( M_1 )</td>
<td>( M_2 )</td>
<td>( M_3 )</td>
</tr>
<tr>
<td>( \Gamma_i^- )</td>
<td>/</td>
<td>( O_1 )</td>
<td>( O_2 )</td>
</tr>
<tr>
<td>( p_i )</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 1. Groups of Permutable Operations solution.

In this paper, we focus on the execution phase of GoPO; we suppose that the groups are already given. The execution of a GoPO schedule can be viewed as a sequence of decisions: each decision consists in choosing an operation to execute in a group when this group is composed of more than one operation. The number of decisions that need to be taken is expressed as follows:

\[
\sum_{v_i} (|G_i| - 1)
\]

For instance, for the solution described on fig. 1 and table 1, there are two decisions to be taken: on \( M_1 \), at the beginning of the scheduling, either operation \( O_1 \) or \( O_7 \) has to be executed. Let us suppose that the decision taken is to schedule

\(^1\) Such that the groups are not related with a precedence constraint and the worst-case permutation in this group does not exceed a given threshold
There is another decision to be taken on $M_3$ : scheduling operation $O_5$ or $O_8$ first. Fig. 2 represents all the different semi-active schedules that can be obtained after taking these two decisions. Note that these schedules do not always have the same quality (performance): the best schedule has a $C_{\text{max}} = 10$ as shown on fig. 2.a and the worst schedule has a $C_{\text{max}} = 12$ as shown on fig. 2.c and fig. 2.d.

Fig. 2. Set of Semi-active schedules.

### 3. Problematic

The main problematic of this paper is to find the best schedule (best-case) to be used as a decision-aid parameter during the decision phase of GoPO. This problem is computationally challenging as it is a combinatorial problem and the number of possible schedules that can be described by a GoPO solution is $\prod_{i \leq k} (|G_i|)!$.

---

\[\text{A feasible schedule is called semi-active if no operation can be finishing earlier without changing the order of the operations on the machines.}\]
The most effective and commonly operational research tool for solving optimally this problem is to use an exact method like the branch and bound algorithm. However, an exact method can be very costly and time-consuming in a decision-aid system, especially for solving large scale NP-hard combinatorial optimization problems. Instead, an effective lower bound can be computed in real-time to produce a near optimal (or optimal) anticipation of the best schedule that can be achieved.

To compute this lower bound and to avoid the exhaustive enumeration of all possible schedules, we can use a symmetrical calculation of the worst-case evaluation [2, 3]. This new evaluation can be measured by computing the best-case-earliest-starting-time of operations and groups.

The computation of this best-case-earliest-starting-time problem is similar to the computation of the longest path in a disjunctive graph [11, 15]. A disjunctive graph is defined as $G = (N, A, B)$ [42], $N$ denotes the set of nodes (operations) including two fictitious operations; a source operation $O_0$ and sink operation $O_{(n \times m) + 1}$ ($n \times m$ being the number of operations), $A$ is the set of conjunctive arcs representing the precedence constraints between operations and denoted by $O_i \rightarrow O_j$ (i.e., $O_i$ precedes $O_j$). $B$ represents the set of disjunctive arcs for each pair of operations that must be performed in the same group and denoted by $O_i \leftrightarrow O_j$ which means that either $O_i \rightarrow O_j$ or $O_j \rightarrow O_i$. A feasible semi-active schedule can be obtained as soon as all the disjunctive arcs are selected.

Fig. 3(a) and Fig. 3(b) present respectively the disjunctive graph representation of the initial job shop problem described in table 1 and the groups of permutable operations shown in fig. 1. These two figures show that the GoPO example is a partial schedule of the initial job shop problem, where some disjunctive arcs are selected. In this GoPO representation, there are two remaining disjunctive arcs $O_1 \leftrightarrow O_7$ and $O_5 \leftrightarrow O_8$. The selection of these arcs leads to one of the schedules presented in fig. 2.

The literature review on the job shop lower bound problem are based on two main techniques identified by [44]: the adjustment conjunctive constraints based techniques and the classical one-machine relaxation-based techniques [4, 16, 17, 38]. These techniques can be the starting point of the GoPO lower bound computation.

However, computing a lower bound for the best-case in a GoPO problem is quite different from computing a lower bound for the optimal schedule of the general job shop problem, because of the conjunctive constraints between groups on the same machines as shown in figure 3. Moreover, due to these conjunctive constraints, the best-case is not necessarily the optimal schedule of the initial general job shop problem. In the next section, this lower bound calculation is presented in three steps: a job shop lower bound adaptation, the groups’ conjunctive constraints improvement and a precedence constraint improvement.
4. LOWER BOUND OF THE best-case SCHEDULE

4.1. THE best-case-earliest-starting/completion-time OF AN OPERATION

The lower bound of the best-case-earliest-completion-time of an operation \( (\chi_i) \) corresponds to the smallest value of \( C_i \) in every semi-active schedule described by GoPO. To compute \( \chi_i \), we need first to calculate the lower bound of the best-case-earliest-starting-time of this operation \( (\theta_i) \).

In the first step, we can compute such a lower bound using a relaxation on the resources by making the assumption that each resource has an infinite capacity. In the GoPO problem, \( \theta_i \) is computed as the maximum of the best-case-earliest-completion-time (lower bound) \( (\chi_j) \) of all its predecessors: for an operation \( O_i \), its predecessors include the predecessors given by the problem \( (\Gamma^-_i) \) and also the operations of the previous group on the same machine (each operations in \( G^-_i \)).

In the job shop example described in fig. 1, the predecessors of operation \( O_6 \) (executed on \( M_1 \) ) are: operation \( O_5 \) (executed on \( M_4 \) ) because of the precedence constraints, and the operations \( O_3 \) and \( O_7 \) executed on the same machine \( M_1 \) because they are in the previous group \( G^-_6 \). So, we have:
\[
\begin{align*}
\theta_i &= \max(r_i, \max_{j \in G_i^-} \chi_j, \max_{j \in \Gamma_i^-} \chi_j) \\
\chi_i &= \theta_i + p_i
\end{align*}
\]

Calculating \( \theta_i \) using equation (1) is equivalent to the head computation of operation \( O_i \) as explained in [15]. [44] used a similar idea based on a one-machine relaxation using the precedence constraints property between operations.

In a second step, we use the precedence constraints between groups to improve the computation of the lower bound; in this case, an operation in a given group \( G_i \) cannot be executed before all the operations of its previous group \( G_i^- \) have been executed. As a consequence, an operation can only begin after the optimal \( \max(C_i) \) of the previous group.

Thus, it needs the computation of the optimal completion time of a group (named as \( \gamma(G_i) \)). We have previously computed \( \theta_i \) as a release date so we can generate a 1\(|r|C_{\text{max}} \) problem instance that corresponds to our problem, with \( r_i = \theta_i \). This problem is polynomially solvable by ordering the operations in ascending release dates [12, 33].

Thus, the improved lower bound for \( O_i \) is computed as follows:

\[
\begin{align*}
\theta_i &= \max(r_i, \gamma(G_i^-), \max_{j \in \Gamma_i} \chi_j) \\
\chi_i &= \theta_i + p_i
\end{align*}
\]

\( \gamma(G_j) = C_{\text{max}} \) of 1\(|r_j|C_{\text{max}}, \forall O_j \in G_j, r_j = \theta_j \)

Table 2 illustrates the computation of the lower bound of all the groups and operations best-case-earliest-completion-times in our job shop example. The computation shows that \( \theta_9 = 9 \) which is not the optimal completion time of \( O_9 (= 10) \) as shown in fig. 2.a. and fig. 2.b.

**Table 2. Computation of the Lower bounds using equation(2)**

<table>
<thead>
<tr>
<th>( O_i )</th>
<th>( O_1 )</th>
<th>( O_7 )</th>
<th>( O_4 )</th>
<th>( O_2 )</th>
<th>( O_5 )</th>
<th>( O_8 )</th>
<th>( O_3 )</th>
<th>( O_6 )</th>
<th>( O_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \theta_i )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>( \chi_i )</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>( \gamma(G_i) )</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

4.2. Precedence constraint improvement

Using equation (2) for the computation of the best-case-earliest-completion-time of operation nine (\( \theta_9 \)) has given an error gap of one. It is noticeable that for the calculation of this lower bound, we put \( O_2 \) and \( O_8 \) on their both best-starting-time, as well as \( O_7 \) and \( O_1 \) transitively, which is inconsistent. We know
for a fact that $O_7$ and $O_1$ cannot start together at their best - starting - time at the same time. In this section, we present a precedence constraint rule to improve $\theta_i$ using an adjustment technique based on the precedence consistency of operations belonging to the same group.

This improvement is based on a property about any two operations ($O_i$ and $O_j$) that belongs to two successive groups ($G_i$ and $G_j / G_j = G_i^+$) on the same machine and have their predecessors (direct or indirect) in the same group ($\Gamma_i^-$ and $\Gamma_j^- \in G_k$). In this case, one of the operations $O_i^-$ or $O_j^-$ has to start after the other one. Executing one of these operations at first and delaying the other one will have an influence on the completion time of $G_j$. Fig. 4 shows this precedence constraint property for our job shop example.

![Figure 4. Precedence consistency between $O_2$ and $O_9$](image)

From the example shown in fig. 4, we have on Machine $M_2$, operation $O_9$ that has to start after $O_8$ because of the precedence constraint, and after operation $O_2$ because of the group precedence on the same machine. Both predecessors of these operations, $O_1$ and $O_7$ are in the same group. Executing one of them first will have an influence on the starting time of operation $O_9$.

To take into account this situation, we propose an adjustment technique for $\chi_i$ that requires two times the calculation of $\chi_i$ using equation (2); in our job shop example $\chi_9$ will be computed one time for the case when $O_1$ is executed before $O_7$ and the other one when $O_7$ is executed before $O_1$. The new $\chi_9$ will be the minimum value between these two evaluations. The results of this improvement are shown on table 3. In this case, the lower bound’s improvement gives the optimal value for the $C_{\text{max}}$.

Computing $\chi_i$ using equation (2) is polynomial and has a complexity of $O(n^2)$. In a GoPo, this precedence constraints property may be found at most $n \times m$ times. So we have a complexity of $O(n^3)$.
Table 3. Improved Lower bounds results

<table>
<thead>
<tr>
<th></th>
<th>( r_1 &lt; r_7 )</th>
<th>( r_7 &lt; r_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O_i )</td>
<td>( \theta^1_i )</td>
<td>( \chi^1_i )</td>
</tr>
<tr>
<td>( O_1 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( O_7 )</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>( O_4 )</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>( O_2 )</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>( O_5 )</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>( O_8 )</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>( O_3 )</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>( O_6 )</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>( O_9 )</td>
<td>7</td>
<td>10</td>
</tr>
</tbody>
</table>

4.3. Heads and tails for the best-case schedule

The classical computation of a job shop lower bound is generally based on the calculation of:

- The \textit{best-case-earliest-starting-time}, called \textit{Head}, which represents the longest path from \( O^* \) and the operation \( O_i \) in a disjunctive graph.
- The \textit{best-case-latest-completion-time} of an operation, called \textit{Tail}, which represents the distance between the latest completion time of an operation and the end of the schedule.

So any improvement of these two values will generally improve the lower bound for the objective function. In a job shop problem, the classical concept of the one-machine relaxation is used; each machine is considered alone and heads and tails have to be found for each one-machine-problem. The maximum value of these evaluations represents the lower bound of the job shop problem.

In a GoPO problem, we can use this relaxation more efficiently relying on the concept of one-group-one-machine-problems relaxation. In this relaxation each group is considered as a one-machine and efficient heads and tails are computed for each group.

Using equation (2), \( \theta_i \) is a valid head and must be quite effective. Because of the symmetry of heads and tails, tails can be computed as \( \theta_i \) using a reversed version of equation (2): rather than starting the computation at the beginning of the scheduling problem, the computation begins at the end. So, replacing predecessor by successor, the new formulation is:

\[
\begin{align*}
\theta'_i &= \max(\gamma^i_{G_i}, \max_{j \in G^i} \chi'_j) \\
\chi'_i &= \theta'_i + p_i \\
\gamma^i_{G_j} &= C_{\max} \alpha \text{ for } r_j \leq C_{\max}, \forall O_j \in G_j, r_j = \theta'_j
\end{align*}
\]
With $\theta_i$ being a valid head and $\theta'_i$ being a valid tail for the group relaxation. The one-group-one-machine-problems are then generated. Then, as for the classical lower bound for the job shop problem, we optimally evaluate the one-group-one-machine-problems using the branch and bound method of [14]. The maximal evaluation is a lower bound of the best-case schedule of GoPO. In the rest of the article, $LB_{best-case}$ and $LB_{best-case}(O_i)$ denote respectively the lower bound of the best-case schedule (for all semi-active schedules generated by the GoPo solution), and the lower bound of the best-case schedule when $O_i$ is executed first in its group.

5. Reactive phase of GoPO

During the reactive phase, each time a group containing more than one operation has to be executed, a lower bound of the best-case of each operation of this group is calculated. This lower bound ($LB_{best-case}(O_i)$) can be either used in a decision reactive algorithm or can be proposed in a decision-aid system.

Similarly to the best-case, the worst-case of an operation ($worst-case(O_i)$) represents the worst realized schedule if operation $O_i$ is sequenced first in its group. These two parameters ($LB_{best-case}(O_i)$ and $worst-case(O_i)$) are available to help the decision maker to look for an alternative schedule in case of a disturbance, without the need of rescheduling again. In this context, $LB_{best-case}(O_i)$ should be of good interest for selecting only the schedules with the highest performances.

The algorithm $RA_{best-case}$ described below is a reactive algorithm which simulates the decision process based only on $LB_{best-case}(O_i)$ parameter.

\[
L(G) = \{G_1, G_2, \ldots G_n\};
\]
\[
(G_i \text{ is a group containing more than one operation}) \quad LB_{best-case} := \text{ maximum value;}
\]
\[
\text{for } \text{ every group } G_i \text{ in } L(G) \text{ do}
\]
\[
\quad \text{ while } \text{ Card}(G_i) > 1 \text{ do}
\]
\[
\quad \quad \text{ for } \text{ every operation } O_i \text{ in } G_i \text{ do}
\]
\[
\quad \quad \quad \text{ - Put } O_i \text{ first ;}
\]
\[
\quad \quad \quad \text{ - Calculate } LB_{best-case}(O_i) ;
\]
\[
\quad \quad \text{ end}
\]
\[
\quad \quad - LB_{best-case} := \min_{O_i \in G_i}(LB_{best-case}, LB_{best-case} - case(O_i));
\]
\[
\quad \quad \text{ (Ties are broken using } \theta_i) - \text{ Remove } O_i \text{ from } G_i;
\]
\[
\quad \text{ end}
\]
\[
\quad - \text{ remove } G_i \text{ from } L(G).;
\]
\[
\text{ end}
\]
\[
\text{ Algorithm 1: } RA_{best-case} \text{ for the reactive phase of GoPO}
\]

As an illustration of how $RA_{best-case}$ works, let us enumerate all the groups of our job shop GoPO example.
$G_1 : \{O_1, O_7\}, G_2 : \{O_5, O_8\}$

$L := \{G_1, G_2\}$

The reactive procedure is executed on the first group of the list $G_1 : \{O_1, O_7\}$:

- $LB_{\text{best-case}}(O_1) = 10$ and $LB_{\text{best-case}}(O_7) = 12$ (using the result in table 3)
- Remove $O_1$ from the group and update $LB_{\text{best-case}} = \min(10, 12) = 10$
- The current group contains only $O_7$ and is removed from the list.

Then the reactive procedure is executed on the second group of the list $G_2 : \{O_5, O_8\}$:

- $LB_{\text{best-case}}(O_5) = 10$ and $LB_{\text{best-case}}(O_8) = 11$
- Remove $O_5$ from the group
- The current group contains only $O_8$ and is removed from the list.

At the end of the reactive phase, the realized schedule generated by $RA_{\text{best-case}}$ has an optimal value of 10. Of course, in this example no disturbances occurred and therefore the final schedule generated by $RA_{\text{best-case}}$ (based on the lower bound) is the optimal schedule of GoPO.

Similarly to $RA_{\text{best-case}}$, we create two other reactive algorithms, both based on the $\text{worst-case}(O_i)$ parameter; the first one called $RA_{\text{worst-case-1}}$ is similar to the algorithm described above where the variable $LB_{\text{best-case}}(O_i)$ is replaced by $\text{worst-case}(O_i)$. The second reactive algorithm is called $RA_{\text{worst-case-2}}$ and is similar to $RA_{\text{worst-case-1}}$ with an additional condition; ties between operations having the same value of $\text{worst-case}(O_i)$ are broken using $LB_{\text{best-case}}(O_i)$. The following steps illustrates $RA_{\text{worst-case-2}}$ on our job shop example for the two groups, $G_1$ at first then $G_2$:

- $\text{worst-case}(O_1) = 11$ and $\text{worst-case}(O_7) = 12$ (using the polynomial algorithm of [2] described also in [3]), therefore $O_1$ is sequenced before $O_7$ using $RA_{\text{worst-case-2}}$.
- Remove $O_1$ from the group and update $\text{worst-case} = \min(11, 12) = 11$ and go to the second group.
- $\text{worst-case}(O_5) = 10$ and $\text{worst-case}(O_8) = 11$ (if for example $\text{worst-case}(O_5) = \text{worst-case}(O_8)$ then $LB_{\text{best-case}}(O_5)$ and $LB_{\text{best-case}}(O_8)$ will be calculated to determine which operation should be sequenced first.
- In this case, $O_5$ is sequenced before $O_8$ and the realized schedule will have a makespan of 10.

In the rest of this section, we assess the relative importance of the three algorithms to their overall performances on a well-known job shop instances.

5.1. Protocol of the experiment

We took a set of benchmark instances called La01 to La40 from [34] with well-known optimal solutions for the makespan objective [11, 22, 36]. These forty instances (i.e., from La01 to La40) are widely used in the job shop literature [7, 39]. These are classical job shop instances of different sizes (5 instances for each size) with a range from 10 x 5 to 30 x 10 where $n \times m$ represents $n$ jobs and
\(m\) machines. These instances can be downloaded from [6] and each job of these instances can start at time zero.

For each instance, we generate a GoPO schedule with a maximum possible flexibility and such that the best-case is not only the optimal schedule of GoPO but also the optimal schedule of the instance.\(^3\) To do so, we use an adaptation of the algorithm of [22] mentioned in section 2. This algorithm has the following steps:

- **Step 0**: Construct a schedule solving the job shop problem. This schedule is called an *initial schedule* and can be calculated using a branch and bound algorithm or a simple heuristic. In our case, for each of the forty Lawrence instances we took the *optimal schedule* found in [11, 36, 39].
- **Step 1**: Initialize the groups in the machines such that each operation \(O_i\) is contained by a group \(G_i\).
- **Step 2**: Calculate the worst-case for each two successive groups that can be merged. If there is no possible merging between the groups in all the machines (due to precedence constraints), finish the algorithm. Otherwise, go to Step 3.
- **Step 3**: Merge the two groups having the minimum worst-case and go back to Step 2.

For each GoPO problem type, we compare the *initial schedule* (optimal) with the *realized schedule* obtained by the three algorithms \(RA_{\text{worst-case-1}}, RA_{\text{worst-case-2}}\) and \(RA_{\text{best-case}}\). The algorithms were coded using the JAVA language and the experiments are made on an Intel(R) Core(TM) i5 CPU.

### 5.2. Results and discussion

The results of these experiments are exposed in table 4 and table 5. Table 4 presents for each instance, the number of groups generated, the number of decisions for the reactive phase, the best-case, Brucker’s lower bound [11], the lower bound of the best-case \((LB_{\text{best-case}})\) using equation 2 and equation 3 and the gap error between \(LB_{\text{best-case}}\) and the best-case.

Table 5 presents for each reactive algorithm \((RA_{\text{worst-case-1}}, RA_{\text{worst-case-2}}, RA_{\text{best-case}})\) the gap error between the makespan of the realized schedule and the initial schedule. For the three algorithms and for all instances, the realized schedule were found in almost one second.

Based on the results shown on table 4, the lower bound computation provided a near-optimal solution with an average gap error less than 1%. Whereas the average gap error of Brucker’s lower bound is 2.74%. Moreover, the proposed lower bound gave the optimal solution for twenty-three instances out of forty (57%). Most of these instances are the same as the optimal instances found by Brucker and are problems of 10(jobs) X 5(machines), 15X5, 20X10, 30X10. This is not that surprising, because the proposed lower bound is based on an adaptation of the

---

\(^3\)In this way, we can measure the efficiency of the best-case lower bound without using a branch and bound algorithm.
Table 4. Lower bound results for the best-case parameter ($LB_{\text{best-case}}$)

<table>
<thead>
<tr>
<th>n X m</th>
<th>Instances</th>
<th>Optimal solution</th>
<th>Brucker’s Lower bound</th>
<th>Lower bound ($LB_{\text{best-case}}$)</th>
<th>Gap error</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 X 5</td>
<td>La01</td>
<td>19</td>
<td>666</td>
<td>666</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>La02</td>
<td>18</td>
<td>655</td>
<td>655</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>La03</td>
<td>15</td>
<td>597</td>
<td>588</td>
<td>1.51%</td>
</tr>
<tr>
<td></td>
<td>La04</td>
<td>14</td>
<td>590</td>
<td>588</td>
<td>0.34%</td>
</tr>
<tr>
<td></td>
<td>La05</td>
<td>15</td>
<td>593</td>
<td>593</td>
<td>0%</td>
</tr>
<tr>
<td>15 X 5</td>
<td>La06</td>
<td>19</td>
<td>926</td>
<td>926</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>La07</td>
<td>17</td>
<td>890</td>
<td>890</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>La08</td>
<td>16</td>
<td>863</td>
<td>863</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>La09</td>
<td>16</td>
<td>951</td>
<td>951</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>La10</td>
<td>19</td>
<td>958</td>
<td>958</td>
<td>0%</td>
</tr>
<tr>
<td>20 X 5</td>
<td>La11</td>
<td>20</td>
<td>1222</td>
<td>1222</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>La12</td>
<td>21</td>
<td>1039</td>
<td>1039</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>La13</td>
<td>18</td>
<td>1150</td>
<td>1150</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>La14</td>
<td>19</td>
<td>1292</td>
<td>1292</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>La15</td>
<td>22</td>
<td>1207</td>
<td>1207</td>
<td>0%</td>
</tr>
<tr>
<td>10 X 10</td>
<td>La16</td>
<td>43</td>
<td>945</td>
<td>875</td>
<td>931</td>
</tr>
<tr>
<td></td>
<td>La17</td>
<td>36</td>
<td>784</td>
<td>739</td>
<td>751</td>
</tr>
<tr>
<td></td>
<td>La18</td>
<td>40</td>
<td>848</td>
<td>770</td>
<td>815</td>
</tr>
<tr>
<td></td>
<td>La19</td>
<td>39</td>
<td>842</td>
<td>709</td>
<td>796</td>
</tr>
<tr>
<td></td>
<td>La20</td>
<td>39</td>
<td>902</td>
<td>807</td>
<td>850</td>
</tr>
<tr>
<td>15 X 10</td>
<td>La21</td>
<td>54</td>
<td>1046</td>
<td>995</td>
<td>1045</td>
</tr>
<tr>
<td></td>
<td>La22</td>
<td>49</td>
<td>927</td>
<td>913</td>
<td>927</td>
</tr>
<tr>
<td></td>
<td>La23</td>
<td>54</td>
<td>1032</td>
<td>1032</td>
<td>1032</td>
</tr>
<tr>
<td></td>
<td>La24</td>
<td>50</td>
<td>935</td>
<td>881</td>
<td>914</td>
</tr>
<tr>
<td></td>
<td>La25</td>
<td>50</td>
<td>977</td>
<td>894</td>
<td>954</td>
</tr>
<tr>
<td>20 X 10</td>
<td>La26</td>
<td>53</td>
<td>1218</td>
<td>1218</td>
<td>1218</td>
</tr>
<tr>
<td></td>
<td>La27</td>
<td>56</td>
<td>1252</td>
<td>1235</td>
<td>1231</td>
</tr>
<tr>
<td></td>
<td>La28</td>
<td>63</td>
<td>1273</td>
<td>1216</td>
<td>1273</td>
</tr>
<tr>
<td></td>
<td>La29</td>
<td>60</td>
<td>1202</td>
<td>1114</td>
<td>1189</td>
</tr>
<tr>
<td></td>
<td>La30</td>
<td>58</td>
<td>1355</td>
<td>1355</td>
<td>1355</td>
</tr>
<tr>
<td>30 X 10</td>
<td>La31</td>
<td>69</td>
<td>1784</td>
<td>1784</td>
<td>1784</td>
</tr>
<tr>
<td></td>
<td>La32</td>
<td>64</td>
<td>1850</td>
<td>1850</td>
<td>1850</td>
</tr>
<tr>
<td></td>
<td>La33</td>
<td>64</td>
<td>1719</td>
<td>1719</td>
<td>1719</td>
</tr>
<tr>
<td></td>
<td>La34</td>
<td>72</td>
<td>1721</td>
<td>1721</td>
<td>1721</td>
</tr>
<tr>
<td></td>
<td>La35</td>
<td>73</td>
<td>1888</td>
<td>1888</td>
<td>1888</td>
</tr>
<tr>
<td>15 X 15</td>
<td>La36</td>
<td>88</td>
<td>1268</td>
<td>1224</td>
<td>1244</td>
</tr>
<tr>
<td></td>
<td>La37</td>
<td>84</td>
<td>1397</td>
<td>1355</td>
<td>1392</td>
</tr>
<tr>
<td></td>
<td>La38</td>
<td>88</td>
<td>1196</td>
<td>1077</td>
<td>1147</td>
</tr>
<tr>
<td></td>
<td>La39</td>
<td>86</td>
<td>1233</td>
<td>1221</td>
<td>1230</td>
</tr>
<tr>
<td></td>
<td>La40</td>
<td>86</td>
<td>1222</td>
<td>1170</td>
<td>1207</td>
</tr>
</tbody>
</table>

Average 0.95%
Table 5. Comparison between the three reactive algorithms

<table>
<thead>
<tr>
<th></th>
<th>RA_worst-case-1</th>
<th>RA_worst-case-2</th>
<th>RA_best-case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gap error (%)</td>
<td>Gap error (%)</td>
<td>Gap error (%)</td>
</tr>
<tr>
<td>La01</td>
<td>155 23.27%</td>
<td>155 23.27%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La02</td>
<td>187 28.55%</td>
<td>187 28.55%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La03</td>
<td>200 33.50%</td>
<td>282 47.24%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La04</td>
<td>184 31.19%</td>
<td>157 26.61%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La05</td>
<td>55   9.27%</td>
<td>55   9.27%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La06</td>
<td>63   6.80%</td>
<td>63   6.80%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La07</td>
<td>132 14.83%</td>
<td>183 20.56%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La08</td>
<td>153 17.73%</td>
<td>93   10.78%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La09</td>
<td>225 23.66%</td>
<td>97   10.20%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La10</td>
<td>71   7.41%</td>
<td>30   3.13%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La11</td>
<td>58   4.75%</td>
<td>58   4.75%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La12</td>
<td>252 24.25%</td>
<td>119 11.45%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La13</td>
<td>351 30.52%</td>
<td>351 30.52%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La14</td>
<td>0    0%</td>
<td>0    0%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La15</td>
<td>272 22.54%</td>
<td>272 22.54%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La16</td>
<td>239 25.29%</td>
<td>301 31.85%</td>
<td>74 7.83%</td>
</tr>
<tr>
<td>La17</td>
<td>129 16.45%</td>
<td>119 15.18%</td>
<td>15 1.91%</td>
</tr>
<tr>
<td>La18</td>
<td>432 50.94%</td>
<td>302 35.61%</td>
<td>43 5.07%</td>
</tr>
<tr>
<td>La19</td>
<td>437 51.90%</td>
<td>354 42.04%</td>
<td>5 0.59%</td>
</tr>
<tr>
<td>La20</td>
<td>373 41.35%</td>
<td>451 50.00%</td>
<td>22 2.44%</td>
</tr>
<tr>
<td>La21</td>
<td>317 30.31%</td>
<td>317 30.31%</td>
<td>21 2.01%</td>
</tr>
<tr>
<td>La22</td>
<td>824 88.89%</td>
<td>481 51.89%</td>
<td>29 3.13%</td>
</tr>
<tr>
<td>La23</td>
<td>522 50.58%</td>
<td>417 40.41%</td>
<td>62 6.01%</td>
</tr>
<tr>
<td>La24</td>
<td>181 19.36%</td>
<td>220 23.53%</td>
<td>71 7.59%</td>
</tr>
<tr>
<td>La25</td>
<td>206 21.08%</td>
<td>271 27.74%</td>
<td>30 3.07%</td>
</tr>
<tr>
<td>La26</td>
<td>747 61.33%</td>
<td>715 58.70%</td>
<td>38 3.12%</td>
</tr>
<tr>
<td>La27</td>
<td>495 39.54%</td>
<td>402 32.11%</td>
<td>26 2.08%</td>
</tr>
<tr>
<td>La28</td>
<td>326 25.61%</td>
<td>284 22.31%</td>
<td>1 0.08%</td>
</tr>
<tr>
<td>La29</td>
<td>645 53.66%</td>
<td>410 34.11%</td>
<td>29 2.41%</td>
</tr>
<tr>
<td>La30</td>
<td>527 38.89%</td>
<td>266 19.63%</td>
<td>25 1.85%</td>
</tr>
<tr>
<td>La31</td>
<td>416 23.32%</td>
<td>324 18.16%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La32</td>
<td>681 36.81%</td>
<td>370 20.00%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La33</td>
<td>774 45.03%</td>
<td>677 39.38%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La34</td>
<td>352 20.45%</td>
<td>336 19.52%</td>
<td>0 0%</td>
</tr>
<tr>
<td>La35</td>
<td>562 29.77%</td>
<td>328 17.37%</td>
<td>31 1.64%</td>
</tr>
<tr>
<td>La36</td>
<td>647 51.03%</td>
<td>530 41.80%</td>
<td>41 3.23%</td>
</tr>
<tr>
<td>La37</td>
<td>579 41.45%</td>
<td>400 28.63%</td>
<td>40 2.86%</td>
</tr>
<tr>
<td>La38</td>
<td>746 62.37%</td>
<td>457 38.21%</td>
<td>45 3.76%</td>
</tr>
<tr>
<td>La39</td>
<td>461 37.39%</td>
<td>321 26.03%</td>
<td>38 3.08%</td>
</tr>
<tr>
<td>La40</td>
<td>691 56.55%</td>
<td>460 37.64%</td>
<td>59 4.83%</td>
</tr>
</tbody>
</table>

\[ \sum \quad 14667 \begin{array}{cccc} 11615 \end{array} 745 \]

Average 32.44% 26.45% 1.71%
classical job shop lower bound. Moreover, the proposed lower bound has been computed for less than one second on each instance.

The results presented in table 5 showed that the reactive algorithm using our improved lower bound (RA\textsubscript{best-case}) dominates all the instances with a performance about fifteen times (11615/745) better than RA\textsubscript{worst-case-2} and twenty times (14667/745) better than that of RA\textsubscript{worst-case-1}.

RA\textsubscript{best-case} gave optimal results for all problems with five machines and all problems with thirty jobs and ten machines except La35; nineteen instances were solved optimally compared to only one instance (La14) by RA\textsubscript{worst-case-1} and RA\textsubscript{worst-case-2}. This can be explained by the accuracy of the lower bound calculated for these instances as shown in table 4.

Compared to RA\textsubscript{worst-case-1}, RA\textsubscript{worst-case-2} that uses the best-case parameter for breaking ties (RA\textsubscript{worst-case-2}) performs better; Its average gap error on the forty instances was 26.45% compared to 32.44% for the reactive algorithm that uses only the worst-case parameter (RA\textsubscript{worst-case-1}). But these two algorithms have a poor performance compared to RA\textsubscript{best-case} which provided a realized schedule with an average gap error of only 1.71%. These results suggest that a good decision in a group of permutable operations cannot be relied primarily on the worst-case parameter.

Overall, this suggestion is not that surprising in regards of what is expected from the worst-case parameter; This parameter is more effective in controlling the performance of the final schedule from not exceeding a performance threshold than guaranteeing an optimal or near-optimal final schedule.

5.2.1. Performance of the reactive algorithm RA\textsubscript{best-case} on a non deterministic environment

In this section, we investigate the robustness of the reactive algorithm RA\textsubscript{best-case} in a disturbed environment.

For this, we try to quantify the performance of the realized schedule in case of bad decisions. A bad decision is called so when the operation chosen first from a group of permutable operations is not the one giving the best-case. For example, in the job shop presented in section 2, choosing operation O\textsubscript{7} (LB-best-case(O\textsubscript{7})=12) to be executed first in G\textsubscript{1} instead of choosing O\textsubscript{1} (LB-best-case(O\textsubscript{1})=10) is considered as a bad decision. This situation may appear whenever a perturbation has occurred on the shop, or simply due to a lack of concentration by the operator. We focus on determining the number of bad decisions required to deteriorate the makespan of the realized schedule with a factor $\rho$ from the optimal schedule (the percentage of performance lost because of a non-optimal schedule). The degraded performance of the best-case can be computed as follows: best-case = optimal-solution * (1 + $\rho\%$).

To determine the number of bad decisions required to deteriorate the makespan, we incorporate these bad decisions randomly in the reactive algorithm RA\textsubscript{best-case} during the decision-aid process. This algorithm was executed fifty times on each square instance (La16 to La20); as these instances have been found to be the
most difficult instances in practice for job shop problems [24]. Then, the number of bad decisions has been measured according to $\rho\%$ as shown in fig. 5.

![Figure 5. Number of bad decisions to deteriorate the makespan with a distance factor $\rho$ from the optimal schedule.](image)

The results obtained showed that for an optimal realized schedule ($\text{best-case} = \text{optimal} - \text{solution} \times (1 + 0\%)$), four bad decisions out of 57 decisions (4/57) does not have any impact on the optimal realized schedule obtained for La17. For the other instances: La16, La18, La20 and La19, the number of bad decisions are respectively 3/57, 2/60, 2/61 and 1/60. It means that, for La19, two bad decisions are enough to deteriorate the makespan.

For all other variations of $\rho$, La16 gave the maximum number of possible bad decisions. For example, when $\rho = 100\%$, the reactive algorithm was allowed to take 22 bad decisions, which represents 38% of the total decisions. It can be noticed that the average number of bad decisions for all the five instances almost varies linearly with regards to $\rho$ with a coefficient of 3% (the percentage number of bad decisions with regards to the total number of decisions). Therefore, the percentage number of bad decisions may be expressed by the following equation:

$$\approx 3\% + (\rho / 10\% \times 3\%)$$

It can be suggested that this computation can be used as a post-evaluation performance deviation check in order to trigger the operator concentration before taking further decisions.

6. Conclusion

In this article we have proposed an evaluation of the base-case parameter in a GoPO schedule. This evaluation relies on a computation of an efficient lower
bound, which is calculated in three steps. The first step is an application of the general job shop lower bound to the GoPO problem. The second step uses the precedence property between groups in GoPO to improve this lower bound. This step needs the calculation of a one-machine problem adapted to GoPO. For this, we proposed the one-machine-one-group relaxation such that the lower bound of each group is considered separately. Then, in the final step, we introduced a new precedence constraint property to tighten this lower bound.

This lower bound can be used as a decision-aid parameter during the reactive phase of GoPO. To evaluate the efficiency of this parameter, three reactive algorithms for GoPO are proposed: the first two algorithms rely primarily on the worst-case parameter, whereas the third algorithm relies only on the best-case parameter. These three algorithms have been implemented in a job shop benchmark problem using the makespan objective. The comparative results show the benefits of the proposed parameter (best-case) which exhibits clearly better performance than the worst-case parameter.

The experimentation of the best-case reactive algorithm has been extended to a disturbed environment. The results obtained has confirmed that the proposed parameter could be of good use during the reactive phase of GoPO in order to maintain a good performance in presence of disruptions.

However, there is still room for improvement:

- The proposed lower bound should also be implemented in a branch and bound method in order to evaluate the exact best possible performance of a flexible schedule.
- The calculation of this parameter may be generalized on other regular objectives or multi-objective approaches.

For further research, the proposed parameter should be implemented in a real decision-aid system involving the human temper factor in order to confirm and extend the experimental theoretical results.

REFERENCES


