Performances of parallel branch and bound algorithms with best-first search

Bernard Mansa,1,* , Catherine Roucairolb

*a School of Computer Science, Carleton University, Ottawa, K1S 5B6 Canada
b Université de Versailles-St-Quentin, 45, Avenue des États-Unis, 78000 Versailles, France

Received 17 August 1993; revised 7 September 1994

Abstract

This paper analyzes the performances of parallel branch and bound algorithm with best-first search strategy by examining various anomalies on the expected speed-up: detrimental, acceleration and detrimental acceleration.

Since the best evaluation is not always sufficient to distinguish the best node to choose with best-first search strategy, we define tie breaking rules for cases when nodes have the same value: the fifo, the lifo and the consistent rules.

The purpose of the paper is to convey, through bounds of the parallel execution for each tie breaking rule, an understanding of the nature of the anomalies, the range of their impact and a comparison of their efficiency to cope with these anomalies.

Sufficient and necessary conditions are given regarding the predisposition for each of the three classes of anomalous behavior. For comparison, we introduce a propriety of proneness to anomaly. In particular, we show that the consistent rule on best-first search Branch and Bound algorithm may be the weaker solution to cope the detrimental acceleration anomaly. Finally, we prove that the fifo rule is theoretically and practically efficient.

Keywords: Anomalies; Branch and bound algorithm; Consistency; Parallel processing; Scalability; Speed-up

1. Introduction

Branch and Bound algorithms (denoted by B&B algorithms) are the most popular techniques used to solve NP-hard combinatorial optimization problems [5]. They use in their implementation a queue of subproblems obtained by decomposition of the original problem. Following the search strategy defined, a partial subproblem (i.e. an
item of this queue) is selected, and this subproblem is again partitioned, except if it can be proved that the resulting subproblems cannot yield an optimal solution or if it can no longer be decomposed.

Consequently, the use of parallelism to speed up the execution of B&B algorithm has emerged as a way to solve larger problem instances and has attracted many researchs (for an introduction to parallel B&B, see [3]). On shared memory multiprocessors, a global priority queue of live nodes is then accessed by several processors in order to speed up exploration of the B&B search tree through the state space. However, anomalous behavior of an execution obtained by the parallel implementation could occur.

First, the analysis of the speedup $S$, i.e. the ratio of the sequential execution time to that of the parallel case, could detect three kinds of anomalies:

- **acceleration anomaly** ($S$ greater than the number of processors used),
- **deceleration anomaly** ($S$ between one and the number of processors used),
- **detrimental anomaly** ($S$ less than one).

Second, comparing two parallel executions, it is possible to use more time with $n_2$ processors than with $n_1$ processors, even though $n_1$ is less than $n_2$, i.e.:

- **detrimental acceleration anomaly** (or **detrimental scalability**).

After Fox et al. in [2], and Burton et al. [1] first results, Lai and Sahni [6, 7] pointed out conditions of detrimental scalability, so that further Lai and Sprague [8] showed conditions under which anomalies are guaranteed not to occur when the number of processors is doubled, or not even doubled. In another way, Li and Wah [10-12] focused on understanding the cause of anomaly during parallelization of the serial algorithm. The existence of subproblems with the same priority of selection has been proved to be the necessary condition of detrimental anomalies.

According to the obvious advantage of keeping acceleration anomalies possible, the interest in avoiding detrimental anomalies has been emphasized. Therefore, Li and Wah presented how a special condition on the nodes with same priority (will be formally introduced further) is sufficient to avoid degradation. Their method is attractive for Depth First Search strategy where anomalous behavior is frequent, and, thus, has been improved technically by Saletore and Kalé [15].

Since the cost of anomalies needs to be compared with the implementation price to forbid them, it is worthwhile to consider design and analysis of basic Best-First Search strategies, which deal with live nodes of same priority (i.e. of same evaluation bound), without either processing or memory overhead.

Search strategies in which the order of exploration of nodes with the same smallest evaluation in the list depend of their arrival order are introduced: the **fifo** rule (i.e. node explored is the oldest node in the list) **lifo** rule (i.e. node explored is the youngest node in the list), and **consistent** rule (i.e. node explored is the leftmost node of the search tree traversed present in the list). The greatest lower bound and least upper bound on the number of iterations for parallel implementations will be given in order to be compared to serial ones with same strategy.
The purpose of the paper is to convey, through those bounds, an understanding of the nature of the anomalies, the range of their impact and a comparison of their efficiency to cope with these anomalies.

The paper contains five sections. After a description of the B&B algorithm in Section 2, we present the common model used for the analysis of parallel best-first search algorithms in Section 3. In the next section, we study the bounds of the number of iterations and the conditions of anomalous behavior during parallelization for the three different secondary rules introduced: the fifo (Section 4.1), the lifo (Section 4.2) and the consistent (Section 4.3). We compared the previous results (Section 5) and make conclusive remarks.

2. Branch and bound algorithm

This section gives necessary definitions and properties required to analyze Branch and Bound algorithm.

The B&B algorithm uses a decomposition process (to partition a given problem in smallest subproblems), a strategy (to select the problem to be decomposed), and a bounding function (to give a lower bound of the value of the solutions in each subproblem obtained by decomposition). A subproblem which the evaluation exceeds the value of the best known solution, or proved not able to yield a better solution, can be discarded.

Let us first introduce a formal definition of B&B (using mainly the notation of Ibaraki [4]).

If PO denotes the Combinatorial Optimization problem, the decomposition process applied to PO can be represented by a rooted tree $B = (P, E)$, where $P$ is the set of nodes of $B$ corresponding to the decomposed problems, and $E$ is the set of edges of $B$ corresponding to the decomposition process. The original problem $P_0$ is the root of $B$. Given $P_i$ and $P_j \in P$, the edge $(P_i, P_j) \in E$ if and only if $P_j$ is generated by a decomposition from $P_i$. The set of terminal nodes of $B$, denoted $T$, are those partial problems solved and which do not need further decomposition. If $f$ denotes the economical function to minimize, $f(P_i)$, $P_i \in T$, denotes the value of $P_i$. We valuate $f(P_i)$ with infinity if $P_i$ is non-feasible. The level (or depth) of $P_i$, denoted $l(P_i)$, is the length of the path from $P_0$ to $P_i \in B$. The level of $P_0$ is 0. An ancestor of $P_j$, denoted $anc(P_j)$, is a node on the path from $P_0$ to $P_j$.

**Definition 2.1.** A lower bounding function $g: P \rightarrow \mathbb{R}^+ \cup \{ \infty \}$ is computed for each subproblem as is created ($\mathbb{R}^+$ denotes the set of non-negatives real numbers), satisfying the following conditions:

(a) $g(P_i) \leq f(P_j)$ $\forall P_j \in T$, $\forall P_i = anc(P_j)$,

(b) $g(P_i) = f(P_i)$ $\forall P_i \in T$,

(c) $g(P_j) \geq g(P_i)$ if $P_j$ is a son of $P_i$, $P_i, P_j \in P$. 


If a decomposed problem $P_i, P_i \in \mathcal{F}$, has a solution with the best objective function value so far, then the solution becomes the *incumbent* $z$ (the best known solution). A node of the search tree is declared "explored" (or "expanded") if it has been decomposed in a set of successors which have been all evaluated.

Lemma 2.1. For any $P_j \in \mathcal{P}$, $P_j$ can be eliminated from consideration if $g(P_j) \geq z$.

**Proof.** According to the properties of the lower bounding function (2.1), this subproblem cannot lead to a least cost solution of $P_0$, and can then be discarded. □

Therefore, we can identify the four disjoint subsets of $\mathcal{B}$, with the value of the optimal solution, denoted by $z^*$, the lower bounding function $g$ and with the cost function $f$ (Fig. 1):

- (Critical tree) $\mathcal{C} = \{P_i | g(P_i) < z^*\}$,
- (Ties nodes) $\mathcal{M} = \{P_i | g(P_i) = z^* \text{ and } P_i \notin \mathcal{F}\}$,
- (Optimal nodes) $\mathcal{O} = \{P_i | g(P_i) = z^* \text{ and } P_i \in \mathcal{F}\}$,
- (Discarded nodes) $\mathcal{D} = \{P_i | g(P_i) > z^*\}$,

where $\mathcal{B} = \mathcal{C} \cup \mathcal{M} \cup \mathcal{O} \cup \mathcal{D}$ and $\mathcal{C}, \mathcal{M}, \mathcal{O}$ and $\mathcal{D}$ are pairwise disjoint.

The set of "live nodes" $\mathcal{A}$ is the set of nodes that have been generated but not yet expanded.

**Definition 2.2.** The Best-First Search strategy $S$ selects the node of $\mathcal{A}$ with the least $g(\ )$ value.
The best-first search of the B&B tree precludes the nodes from being explored and, thus, attempts to minimize the number of subproblems expanded (see [2, 13]).

However, no rule has been formally introduced in the definition of this strategy, when two or more nodes have the same smallest lower bound value. It is worthy of attention that common implementations use an implicit heap as queue of live nodes and consequently cannot predict and cannot prescribe in which order the subproblems with same evaluation will be selected.

We define specific tie-breaking rules to deal with nodes with the same lower bound nodes. Assuming that the youngest, oldest and leftmost denote the set of nodes in the active list $\mathcal{A}$, respectively, the most recently generated, the least recently generated and the leftmost in the search tree $\mathcal{B}$.

\begin{align*}
(lifo) \quad S_l(\mathcal{A}) &= \text{set youngest ones among } \{P_i | g(P_i) = \min_{P_j \in \mathcal{A}} g(P_j)\}, \\
(fifo) \quad S_f(\mathcal{A}) &= \text{set of oldest ones among } \{P_i | g(P_i) = \min_{P_j \in \mathcal{A}} g(P_j)\}, \\
(sequence) \quad S_s(\mathcal{A}) &= \text{set of leftmost ones among } \{P_i | g(P_i) = \min_{P_j \in \mathcal{A}} g(P_j)\}.
\end{align*}

3. Parallel branch and bound algorithm

In a parallel implementation, an ideal scheduling algorithm is one which keeps all the processors busy executing essential tasks, and which minimizes the interprocessor communications. In the B&B case, the scheduling is particularly challenging since the tasks are generated dynamically. Each processor executes the decomposition process as in the serial case: it selects the node with least evaluation, expands it and inserts each generated subproblems which could lead to a better solution (a mutual exclusion process is required when changing the incumbent or when accessing the priority queue [9]). The primitive unit-time computational step is the node expansion. The termination of the algorithm is determined when the queue is empty and all the processors are idle.

However, four main assumptions are commonly required to simplify the analysis of the model [7, 8, 11].

(A1) Direct history. The bounding function and the branching scheme applied to a subproblem, $(P_i)$, only depend on the information obtained along the path from the initial node, $P_0$, to this node, $P_i$.

(A2) Synchronicity. At the same time, all the processors select a different subproblem to expand. All the processors insert the generated problems at the end of the computational step, all together.

(A3) Constant granularity. The size of work of each processor for each iteration of the algorithm is constant all over the execution.

(A4) No implementation overhead. The access time on the live nodes queue or on any shared resource is constant or null.
The first assumption is usually incorporated in B&B definition implementation. The three other assumptions introduce the notion of *iteration* of the B&B algorithm:

**Definition 3.1.** During each iteration, each processor executes a cycle of selection-expansion-insertions. If the number of live nodes is less than the number of processors, the starving processors will wait until the next iteration.

With $p$ processors, at most $p$ subproblems with the smallest evaluation will be decomposed during one iteration. A full non-determinism remains for the lifo rule and fifo rule defined: the selection function depends on the order of insertion of the nodes.

**Definition 3.2.** Under the assumptions (A1)–(A4), a B&B strategy is *consistent* if and only if at least one node expanded during the sequential execution is expanded at each iteration of the parallel one.

**Property 3.1.** The *sequence* rule is consistent for the best-first search strategy.


**Theorem 3.1.** Consistency is a sufficient condition to avoid detrimental anomalous behavior for a Best-First Search strategy.

To ensure Property 3.1, Li and Wah [11] have proposed to add a path number as a second key of priority for each node (e.g. in the *sequence* rule, the node selected is the one with the least evaluation bound and the leftmost path). The sequence consistent strategy may allow acceleration anomaly.

**Definition 3.3.** Under the assumptions (A1)–(A4), a B&B strategy is *completely consistent* if and only if one node $P_i$ is selected before another node $P_j$ under the necessary and sufficient condition that $g(P_i) \leq g(P_j)$.

**Property 3.2.** A necessary condition to allow acceleration anomaly is that the strategy is not completely consistent [11].

We introduce the notion of the minimum and the maximum number of nodes in the search tree to expand.

**Definition 3.4.** In a tree, the *distance* of a node $P_j$ to a set of nodes $\mathcal{X}$, denoted $d(P_j, \mathcal{X})$, is the minimal number of nodes on the path between $P_j$ and a node of $\mathcal{X}$. 
Proposition 3.1. With a Best-First Search strategy defined, the number of iterations of a sequential B&B algorithm $\Phi(1)$ is bounded by:

$$|\mathcal{C}| + \min_{P_j \in \mathcal{C}} d(P_j, \mathcal{C}) \leq \Phi(1) \leq |\mathcal{C} \cup \mathcal{M}|,$$

where $|\mathcal{C}|$ is the cardinality of $\mathcal{C}$.

Proof. This result can be found in [5]. However we give the complete proof to emphasize the basic ideas which will be used in the following.

Obviously, a terminal node, $P_j^* \in \mathcal{C}$, with the best objective function value $z^*$, has to be generated, and the critical tree, $\mathcal{C}$, is explored completely to prove that there is not a better solution than $z^*$. But, the exploration of nodes of $\mathcal{M}$ may be required to reach $P_j^*$, i.e. to generated it. The smallest number of such nodes can be obtained by traversing the shortest path between $\mathcal{C}$ and $\mathcal{C}$.

The right term corresponds to an execution where the rule defined yields to the best solution node the latest as possible. In this case, the father node $P_i$, which generates the node $P_j^*$ with value $z^*$, will be the latest of $\mathcal{M}$ selected of the live nodes queue $\mathcal{A}$. Thus, the worst case of the expansion of the search tree includes $\mathcal{C} \cup \mathcal{M}$ and there is a unique solution $P_j^*$, whose father $P_i$ belongs to $\mathcal{M}$. $\square$

Throughout this paper, $\Phi(1)$ and $\bar{\Phi}(1)$ will denote respectively the lower and upper bounds defined in Proposition 3.1.

Observe that if the subset $\mathcal{M}$ is empty, the Best-First Search strategy is optimal that is the number of iterations $\Phi(1)$ is equal to $\Phi(1)$, i.e. the number of subproblems of $\mathcal{C}$.

The number of iterations of a parallel B&B algorithm required to explore a sequential search tree with arbitrary Best-First Search strategy, is bounded.

Proposition 3.2. The number of iterations $\Phi(p)$ of a best-first search parallel B&B execution with $p$ processors is bounded by

$$\max \left( \frac{\Phi(1)}{p}, h_{\mathcal{C}}, \min_{P_i \in \mathcal{C}} l(P_i) \right) \leq \Phi(p) \leq \frac{(\Phi(1) - h_{\mathcal{C} \cup \mathcal{M}})}{p} + h_{\mathcal{C} \cup \mathcal{M}},$$

where $h_{\mathcal{C}}$ is the depth of the Critical tree $\mathcal{C}$, and $h_{\mathcal{C} \cup \mathcal{M}}$ is the depth of the tree $\mathcal{C} \cup \mathcal{M}$.

Proof. Clearly, the ratio of $\Phi(1)$ to the number of processors is a lower bound. But, even with an infinite number of processors, we can not generate immediately (i.e. in a constant number of iterations) the solution node. The whole decomposition process between the original problem $P_0$ and a solution node (even the nearest) has to be done, and is intrinsically sequential. Conversely, the optimality is only proved when the critical tree $\mathcal{C}$ is completely explored, that is when the deepest node of $\mathcal{C}$ has been reached.

The upper bound can be decomposed in two parts with the maximum of iterations for which each processor has a node to expand, and the maximum of iterations needed
to reach the deepest node when there is not enough work for each processor, i.e. \( h_{\text{WUM}} \).

Unfortunately, these bounds are not tight (that is, they may be unreachable). Throughout this paper, \( \Phi(p) \) and \( \Phi(p) \) will denote respectively the lower bound and the upper bound defined in Proposition 3.2.

4. B&B strategies and same priority nodes

We show the different behaviors of each of the three rules by bounding the number of iterations during an execution.

4.1. Fifo rule

The subset of nodes \( P_i \) which belong to \( \mathcal{M} \), such that \( P_i \) has \( k \) ancestors in \( \mathcal{M} \) is denoted by \( \mathcal{M}_k \). The rank in \( \mathcal{M} \) of each of the nodes of \( \mathcal{M}_k \) is defined by the value of the index \( k \).

The expansion in the search tree can be described like a wave in each rank in the path.

Lemma 4.1. During a sequential execution of a best-first search with fifo rule, all the nodes of \( \mathcal{M}_{i-1} \) have to be expanded before the exploration of a node of \( \mathcal{M}_i \), with \( i \) greater than zero.

Proof. By induction. During the sequential execution, the exploration of a node of \( \mathcal{M}_0 \) is possible if and only if all the nodes of \( \mathcal{C} \) have been expanded. Moreover, at the termination of the exploration of the last node of \( \mathcal{C} \), all the nodes of \( \mathcal{M}_0 \) are in the active list, \( \mathcal{A} \) (none has been expanded and none could be inserted now). Since his father belongs to \( \mathcal{M}_0 \), a node \( P_1 \) of \( \mathcal{M}_1 \) will be inserted in \( \mathcal{A} \), with a lower priority than any node \( P_0 \) of \( \mathcal{M}_0 \) (\( P_0 \) is older than \( P_1 \)). We can repeat the above argument and then prove the lemma.

Theorem 4.1. The node \( P^* \) with the best solution value found during the sequential execution of a best-first search with fifo rule, belongs to the subset of optimal nodes with minimal distance to the set of critical nodes: \( P^* \in \{ P_i \in \mathcal{C} \mid d(P_i, \mathcal{C}) = \min_{P_j \in \mathcal{C}} d(P_j, \mathcal{C}) \} \).

Proof. By contradiction, assume that a node \( P_i \in \mathcal{C} \) with a strictly longer path from \( \mathcal{C} \) has been generated before such a node \( P^* \). Under the condition that there is no best solution generated by the exploration of a critical node, \( P_m \) the father of \( P_1 \) and \( P_q \) the father of \( P_4 \) both belong to \( \mathcal{M} \). Let \( \mathcal{M}_m \) denote the set which incorporates \( P_m \) and \( \mathcal{M}_k \) the set which includes \( P_q \), \( k_m \) is greater than \( k_q \). According to Lemma 4.1, \( P_q \) has to be expanded before \( P_m \), which contradicts the assumption.
Thus, the following proposition gives reachable bounds on the number of sequential iterations:

**Proposition 4.1.** Let $\mathcal{M}_k$ be the set in which the father of the optimal node belongs. The number of iterations $\Phi_f(1)$ of sequential execution is between:

$$|\mathcal{G}| + \sum_{i=0}^{k^* - 1} |\mathcal{M}_i| \leq \Phi_f(1) \leq |\mathcal{G}| + \sum_{i=0}^{k^*} |\mathcal{M}_i|.$$

In the case of a parallel implementation, the fifo rule tightens the bounds (considering the difference of iterations between a parallel and a sequential execution).

**Proposition 4.2.** The number of iterations $\Phi_f(p)$ of a best-first search parallel B&B with fifo rule is lower bounded by:

$$\max \left( \frac{\Phi_f(1) - (m - 1)(h_\alpha - \min_{P_e \in \mathcal{M}} l(P_1))}{p}, \Phi_f(p) \right) \leq \Phi_f(p),$$

where $m$ is an integer which represents the number of paths in $\mathcal{M}$.

**Proof.** The parallel exploration of nodes from $\mathcal{G}$ have theoretically the same behavior than a waiting iteration owing the lack of work. However, the parallel execution traversing the sequential search tree $P_s$ may not be the best possible parallel search tree $P_p$.

During the parallel execution, the case where (Fig. 2(a)), a better improvement has been done in finding a node $P_\alpha$ with best solution value sooner than the node $P_\alpha$ found in the sequential execution, where sooner denotes that the father $P_p$ of $P_\alpha$ is inserted in the queue before the father $P_\beta$ of $P_\alpha$. Assume that an ancestor $P_{\gamma}$ of $P_\alpha$ has been expanded before the node $P_\alpha$. The ancestor $P_\beta$ of $P_\alpha$ (which belongs to $\mathcal{M}_{k_{\alpha}}$) has been expanded after $P_\alpha$, the father of $P_\alpha$ which belongs to $\mathcal{M}_{k_{\alpha}-1}$. Repeating the inductive scheme of the fifo rule again, the highest ancestor $P_{\delta_0}$ of $P_\alpha$ which belongs to $\mathcal{M}_0$ has been expanded after $P_\alpha$, the ancestor of $P_\alpha$ which belongs to $\mathcal{M}_{k_{\alpha}-k_{\beta}}$. Thus, the generation of the terminal node $P_\alpha$ instead of $P_\beta$ saved at least the exploration of nodes of the path from $P_\alpha$ to $P_\beta$.

The maximum number of discarded nodes is bounded by $k_{\alpha} - k_{\beta}$, where $k_{\alpha} = k_p - k_\alpha$. Moreover, $k_\alpha$ is bounded for $P_\alpha$ by $(h_\alpha - \min_{P_e \in \mathcal{M}} l(P_1))$, which represents the maximum difference of path length between $P_\alpha$ and $P_{\delta_0}$. According to Theorem 4.1, $k_\alpha \geq k_\beta$. The maximum of $k_\alpha - k_\beta = k_\gamma - (k_p - (h_\alpha - \min_{P_e \in \mathcal{M}} l(P_1)))$ will be upper bounded for $k_\beta = k_\gamma$. The maximum length of path from $P_\beta$ to $P_\alpha$ is equal to $k_\alpha - k_\beta$, i.e. $h_\alpha - \min_{P_e \in \mathcal{M}} l(P_1))$.

The generation of the terminal node $P_\alpha$ could also discard all the nodes of $\mathcal{M}$ in the active list $\mathcal{M}$ (like $P_\beta$). The maximum of such nodes is $(m - 1)$, which denotes one less the number of different paths in $\mathcal{M}$. The number of nodes explored has been reduced by $(m - 1)(h_\alpha - \min_{P_e \in \mathcal{M}} l(P_1))$ down to the global minimum of the parallel search.
\[ H = h_c - \min_{P \in M} l(P_i) \]

Proposition 4.3. The number of iterations \( \Phi_f(p) \) of a best-first search parallel B&B with fifo rule is upper bounded by

\[
\Phi_f(p) \leq \min \left( \Phi_f(p) + (m - 1) \left( h_c - \min_{P \in M} l(P_i) \right) - h_{\mathcal{C} \cup \mathcal{M}} + h_{\mathcal{C} \cup \mathcal{M}} \right),
\]

where \( m \) is an integer which represents the number of paths in \( \mathcal{M} \).

Proof. In the worst case of best-first search with the fifo rule, the exploration of nodes of \( \mathcal{D} \) during parallel execution can appear but have the same behavior than a waiting iteration because of the lack of work. An "expand-all-nodes" of the \( \mathcal{C} \cup \mathcal{M} \) tree, generates the upper bound on the number of iterations, i.e. 

\[ \left( |\mathcal{C} \cup \mathcal{M}| - h_{\mathcal{C} \cup \mathcal{M}} \right)/p + h_{\mathcal{C} \cup \mathcal{M}} \].

The height of \( \mathcal{C} \cup \mathcal{M} \) has to be taken in account to consider the iterations where there is not enough nodes in the active list to keep all the processors busy.

Nevertheless, \( |\mathcal{C} \cup \mathcal{M}| \) may have not to be considered in complete. According to the fifo rule breadth exploration, the worst parallel exploration will at least contain the
sequential tree. Consider the case (see Fig. 2(b)) where a node \( P_p, P_p \in \mathcal{P}_p \) generated during the parallel execution has been not generated in the sequential one, \( P_p \notin \mathcal{P}_s \). Since \( P_s, P_s \in \mathcal{C} \), a terminal node has been generated in the sequential execution, \( P_o \) has to be generated in the parallel execution before \( P_s \). Let \( P_s \) denote the father of \( P_s \), \( P_o \) belongs to \( \mathcal{N} \). Let \( P_o \) denote the father of \( P_p \) which has been expanded before \( P_s \), \( P_o \) belongs to \( \mathcal{N}_{k_s-1} \). Repeating this fifo rank scheme, \( P_o \) the highest ancestor of \( P_p \) which has been expanded before \( P_s \), belongs to \( \mathcal{N}_{k_s-1} \). Let \( P_{s0} \) denote the highest ancestor of \( P_p \) in \( \mathcal{N} \). Then the maximum of length between \( P_p \) and \( P_{s0} \) is equal to \( k_p = (k_p - k_{s0}) + (k_{s0} - k_p) = (k_s + 1) + (h_o - \min_{P_p \in \mathcal{P}} (l(P_i))) \). Finally, the maximum number of explorations along the path to \( P_p \) which does not belong to the sequential execution is equal to \( (h_o - \min_{P_p \in \mathcal{P}} (l(P_i))) \).

Repeating the lower bound argument, the number of paths for which a node is expanded only during the parallel execution is bounded by \((m - 1)\), i.e. the maximum number of paths in \( \mathcal{N} \) different to the path to \( P_s \). The number of explored nodes in the parallel search has been increased by \((m - 1) \times (h_o - \min_{P_p \in \mathcal{P}} (l(P_i)))\), up to the global maximum of the parallel search, \( \Phi(p) \).

Throughout the paper, \( \Phi_f(p) \) denotes the lower bound one the number of iterations considered in Proposition 4.2 and \( \Phi_f(p) \) denotes the upper bound on the number of iterations considered in Proposition 4.3.

The significant outcome of the two propositions is the analysis of sufficient and necessary conditions of the three main anomalies, which becomes straightforward.

First, the maximum speed-up specifies the available acceleration anomaly of the parallel implementation.

**Corollary 4.1.** The value of \( s \), the expected speedup with \( p \) processors, is upper bounded by:

\[
s \leq p \times \frac{\Phi_f(1)}{\Phi_f(1) - (m - 1)(h_o - \min_{P_p \in \mathcal{P}} l(P_i))}.
\]

**Proof.** Immediate, following Proposition 4.2. The speedup is upper bounded by the ratio between the sequential execution \( \Phi_f(1) \) and the best possible parallel execution, \( \Phi_f(p) \).

Second, the condition of a detrimental anomalous behavior specifies the availability to effectively improve the performance by using parallelism.

**Corollary 4.2.** An anomalous detrimental behavior can occur during the implementation of a parallel best-first search with fifo rule if and only if

\[
(m - 1)(h_o - \min_{P_p \in \mathcal{P}} l(P_i)) \geq (p - 1)(\Phi_f(1) - h_o - \min_{P_p \in \mathcal{P}} l(P_i)).
\]
Proof. Immediate, following Proposition 4.3. An anomalous detrimental behavior exists when the speed-up (i.e. the ratio between the sequential execution and the parallel one) is less than one. Thus, comparing the number of iterations during sequential execution, \( \Phi_f(1) \), with the number of iterations with the worst possible parallel execution, \( \Phi_f(p) \), we prove the corollary.

Third, the scalability of the parallel algorithm with the problem can be easily analyzed when comparing its availability to efficiently use an increasing number of processors.

Corollary 4.3. An anomalous detrimental acceleration behavior can occur during the implementation of a parallel best-first search with lifo rule, when increasing the number of processors from \( p_1 \) to \( p_2 \) processors \((p_2 = \rho p_1)\) if and only if

\[
p_1 > \frac{(\rho - 1) \Phi_f(1) - (\rho + 1)(m - 1)(h_w - \min_{r \in \mathcal{R}} l(P)) + (h_{\mathcal{E} \cup \mathcal{M}})}{\rho(h_{\mathcal{E} \cup \mathcal{M}})}.
\]

Proof. Immediate, following Propositions 4.2 and 4.3. An anomalous detrimental acceleration may exist when comparing executions with \( p_1 \) and with \( p_2 \) processors \((p_1 < p_2)\) if and only if \( \Phi_f(p_1) < \Phi_f(p_2) \), i.e. when the parallel execution with \( p_1 \) processors takes less iterations than the parallel execution with \( p_2 \) processors. Thus, with \( \rho = p_2/p_1 \), the ratio of increasing resources, we obviously prove the corollary.

4.2. Lifo rule

As the previous section, the best-first search with lifo rule precises the order of selection in the active list.

Proposition 4.4. The number of iterations of sequential B&B execution \( \Phi_i(1) \) is bounded by:

\[
|\mathcal{C}| + \min_{P_j \in \mathcal{E}} d(P_j, \mathcal{C}) \leq \Phi_i(1) \leq |\mathcal{E} \cup \mathcal{M}|.
\]

Proof. This result is similar to Proposition 3.1 which considers an arbitrary rule, but is different by nature. Clearly, the lifo rule inherently attempts to explored further in a path of the Branch and Bound search tree including nodes with equal values. The maximum number of iterations is reached when the initial paths enumerated in \( \mathcal{M} \) are not leading to a node with the best feasible value. Conversely, the minimum is reached when the initials paths enumerated in \( \mathcal{M} \) lead to a node with the best feasible value in a minimum of nodes expanded in \( \mathcal{M} \).
Proposition 4.5. The number of iterations $\Phi_t(p)$ of a best-first search parallel B& B with lifo rule is bounded by
\[
\max \left( \frac{\Phi_t(1) - |A|}{p} \cdot \Phi_t(p) \right) \leq \Phi_t(p) \leq \min \left( \frac{\Phi_t(1) + |A| - h_{N_1} + h_{N_2} + h_{N_3}}{p} + \Phi_t(p) \right).
\]

Proof. The difference between the number of available nodes expanded in the sequential and in the parallel execution can reach at most $|A| - 1$. This result shows that the two adverse cases are possible with the lifo rule.

First, the worst number of expansions can be iterated in the parallel case even if the sequential execution has led to an optimal exploration of the Branch and Bound search tree. If the path leading to the node with the best values has been the initial node explored, the sequential execution will be completed without visiting the useless part of $\mathcal{A}$. Nevertheless, in the parallel case, other existing processors may thus generate different nodes of $\mathcal{A}$ and insert them between an insertion of the node of the useful path and a selection operation (because of the non-deterministic order of the insertions). Such a parallel case leads to a complete exploration of the part to $\mathcal{A}$ that is unlikely to contain a solution, before working further in the useful path. This case is described with the right term.

Observe that the expression obtained is always dominated by the upper bound of parallel execution $\Phi_t(p)$ since $\Phi_t(1)$ is at least equal to the cardinality of the critical set. Nevertheless, this notation is relevant since it can describe the maximal difference of unexplored nodes in the parallel case, i.e. the anomalous case.

Second, the best number of expansions can be iterated in the parallel case even if the sequential execution has achieved the worst exploration of the Branch and Bound search tree. If the path leading to the node with the best value has been the last explored, the sequential execution will be completed with the visit of the whole useless part of $\mathcal{A}$. Nevertheless, in the parallel case, other existing processors may thus generate a node of the useful path of $\mathcal{A}$ and insert it between an insertion of the node of the useless part and a delete-min operation (because of the non-deterministic order on the insertions). Such a case (the node leading to the solution inserted the last), repeated in each parallel iteration, leads to a smaller exploration of the part of $\mathcal{A}$ that is unlikely to contain a solution. This case is described with the left term.

In the following, $\Phi_{\ell}(p)$ and $\overline{\Phi_{\ell}}(p)$ denote the lower bound and the upper bound on the number of iterations considered in Proposition 4.5. The analysis previously used for the fifo rule is repeated, and the three corollaries are deduced.

Corollary 4.4. The value of $s$, the expected speedup which can occur with $p$ processors, is upper bounded by
\[
s \leq p \times \frac{\Phi_t(1)}{\Phi_t(1) - |A|}.
\]
Corollary 4.5. An anomalous detrimental behavior can occur during the implementation of a parallel best-first search with lifo rule if and only if
\[ |\mathcal{A}| \geq (p - 1) \times (\Phi_1(1) - h_{\Phi_1}^1). \]

Corollary 4.6. An anomalous detrimental acceleration behavior can occur during the implementation of a parallel best-first search with lifo rule, when increasing the number of processors from \( p_1 \) to \( p_2 \) processors (\( p_2 = pp_1 \)) if and only if
\[ p_1 > \frac{(p - 1) \Phi_1(1) - (p + 1)(|\mathcal{A}|) + (h_{\Phi_1}^1)}{p(h_{\Phi_1}^1)}. \]

4.3. Consistent rule

Without loss of generality, we assume that the consistent rule is the sequence one, presented in Eq. (4).

Clearly, the sequential behavior is similar to the one with lifo rule (even if the cause is different by nature) whereas the parallel case is not.

Proposition 4.6. The number of iteration \( \Phi_c(p) \) required for the execution of the best-first search Branch and Bound with a consistent rule is bounded by
\[ \max \left( \frac{\Phi_1(1) - |\mathcal{A}|}{p}, \Phi_c(p) \right) \leq \Phi_c(p) \leq \min(\Phi_1(1), \Phi_c(p)). \]

Proof. The difference between the number of available nodes expanded in the sequential and in the parallel execution is upper bounded by \( |\mathcal{A}| - 1 \).

Following Theorem 3.1, the worst number of expansions iterated in the parallel case cannot exceed the sequential one.

Conversely, the best number of expansions can be achieved in the parallel case even if the sequential execution led to the worst exploration of the Branch and Bound search tree. If the path leading to the node with the best value has been the last explored (the rightmost), the sequential execution will be completed with the visit of the whole useless part of \( \mathcal{A} \). Nevertheless, in the parallel execution, the case where the useful path is the only possible path in \( \mathcal{A} \) to be explored because of the non-complete part of \( \mathcal{G} \) may occur. Such a case leads to a non-exploration of the part of \( \mathcal{A} \) that is unlikely to contain a solution, which has been not the case in the sequential execution. \( \square \)

In the following, \( \underline{\Phi_c}(p) \) and \( \overline{\Phi_c}(p) \) denote the lower bound and the upper bound considered in Proposition 4.6.

The same analysis of sufficient and necessary conditions of the three main anomalous behaviors follows.
Corollary 4.7. The value of $s$, the expected speedup which can occur with $p$ processors, is upper bounded by

$$s \leq p \times \frac{\Phi_c(1)}{\Phi_c(1) - |A|}.$$ 

Corollary 4.8. An anomalous detrimental behavior cannot occur during the implementation of a parallel best-first search with a consistent rule.

Corollary 4.9. An anomalous detrimental acceleration behavior can occur during the implementation of a parallel best-first search with consistent rule, when increasing the number of processors from $p_1$ to $p_2$ processors ($p_2 = \rho p_1$) if and only if

$$p_1 > \frac{\rho \Phi_c(1) - \rho(|\mathcal{E}|) - (\rho + 1)(|\mathcal{A}|) + (h_{\mathcal{G} \cup \mathcal{A}})}{\rho (h_{\mathcal{G} \cup \mathcal{A}})}.$$ 

5. Comparative study

The underlying causes of anomalies are known. In the previous section, we make explicit that they are depending on the tree structure of the tie nodes generated, but have a limited range of impact.

We identified the conditions in relation with a particular execution. We compared the reachable bounds for the same specific rule. The sensitivity to anomaly is dependent upon the quality of the sequential exploration. The number of iterations, $\Phi_c(1)$ and $\Phi_c(1)$, used in the parallel bounds may be quite different as shown with the presented intervals for the sequential bounds.

Nevertheless, the differences of the amount of nodes expanded are all relative on the part of the tie nodes set $\mathcal{A}$ visited or not. The maximum of the difference has been pointed out for each rule. Observe that the main condition on bounding (ensured with the global bounds on parallel execution, $\Phi(p)$ and $\Phi(p)$) is :

$$(m - 1)(h_x - \min_{P \in \mathcal{A}} l(P)) \leq |\mathcal{A}|.$$

Those terms represent the overcost in the detrimental anomalies or the gain in acceleration anomalies. We can easily deduce that, as compared to the fifo rule, different executions may induce a significant greater difference of iterations with the lifo rule.

To formulate the metric of the sensibility to anomalies, we introduce a scale of proneness to anomalous behavior.

Definition 5.1. A search strategy with a given rule $r$ is less prone to anomaly during parallelization than a strategy with rule $r'$ if the size of the interval of the possible number of iterations for the B&B execution is smaller:

$$\overline{\Phi_r(p)} - \underline{\Phi_r(p)} < \overline{\Phi_{r'}(p)} - \underline{\Phi_{r'}(p)}.$$
Proposition 5.1. The fifo rule is less prone to anomalies than the lifo rule.

Considering the several corollaries introduced, the parallel best-first search strategy with fifo rule does not show significant detrimental anomalies as comparing to the lifo rule. Another advantage of the fifo rule is given with the scalable anomaly condition presented, which is relaxed compared to the lifo rule.

The consistent rule allows acceleration anomalies but forbids detrimental ones. This confirms the main advantage of consistency described by Li and Wah [11] regarding the parallelization of a sequential program. However, we detect that the detrimental acceleration anomaly may occur with a consistent rule.

Proposition 5.2. The consistent rule allows detrimental acceleration anomaly.

Although, following Corollaries 4.6 and 4.9, we generalize the definition of proneness to anomalies for the scalable analysis. In this case, we consider the potential interval of iterations with an increasing number of processors.

Proposition 5.3. The consistent rule is more prone on scalable anomaly than the lifo rule if $\rho(\Phi_{\tau}(1) - |\mathcal{E}|) < (\rho - 1)\Phi_{\tau}(1)$, where $\rho$ denotes the increasing ratio of processors.

Therefore, the consistency cannot be considered as the final efficient solution to cope the problem of parallel anomalous behaviors since the detrimental acceleration is not avoided, and can be worst than the lifo rule (and even more than the fifo rule).

The tenet that a consistant strategy can definitely avoid anomalies no longer makes sense on machines which are scalable. This is all the more relevant for commercial applications which are time critical. Indeed, such a periodic computation of an instance of a problem cannot suffer exceptional, but fatal, anomalous behavior.

Unlike the consistent rule, the lifo and the fifo rules are based on features inherent to the execution, or to the parallel machine used (number of processors, access to the priority queue, . . . ). They are self-adaptative to the constraints of the host system. It is worth pointing out that the cost of implementing a secondary key is disproportionate as compared to a practical rule such as the fifo.

These results raise practical and theoretical perspectives.

The distribution of Branch and Bound node values is usually exponential, and thus provides a significant ratio of tie nodes (unsolved problem with a evaluation equal to the optimal solution). This has been confirmed by Quinn and Deo [14] who analyzed the upper bound with a non-constant granularity assumption. Experimental results show that the behavior of the lower bounding function on a small instance of the problem can be generally extended to larger instances. Therefore, a test of a small size problem with few processors can improve the tuning choices of the execution for a greater instance on a scalable machine.

In a more theoretical point of view, it is known that the best sequential algorithm, and the best parallel algorithm may not be known for all instances of a particular
problem. The result on the scalability of the consistent rule shows that a "good" parallelization of a sequential algorithm may not be the best parallel algorithm for the problem to be solved.

The current theory of parallel computation is rooted in concept inherited from sequential computation. The results of this paper suggest that the speedup of the parallel execution should not be the main goal of the parallelization. However, it clearly gives information on the efficiency of the accuracy of the tie-breaking rule regarding the lower bounding function. This also confirms that branching and bounding are definitely interdependent, and should not be designed separately.

Future research should explore the notion of optimality. The usual motivation cited for parallelism is a decrease in execution time. It is necessary to generalize this conventional notion to take into account the scalability. For example, following its definition, the consistency is relative to the sequential strategy. Its main result is to optimize the exploration in parallel of the sequential search tree (for example, a new sequence rule with rightmost choice instead of leftmost is also consistent). Moreover, the question of the better expansion of a Branch and Bound search tree is not considered in this approach which only leads to a sequential guideline of the parallel exploration with a non-negligible overhead. Considering a resolution of a specific problem, the parallel implementation of B&B algorithm will be optimal if and only if it \textit{minimizes the maximum amount of time required by a processor}, that is which minimizes the computational time of the last busy processor.

In this context, the impact of a non-consistent strategy such as the fifo rule is theoretically and practically relevant.

References