High Performance Peer-to-Peer Distributed Computing with Application to Constrained Two-dimensional Guillotine Cutting Problem

Mhand Hifi,Toufik Saadi and Nawel Haddadou
Laboratoire MIS, Axe Optimisation Discrète et Réoptimisation,
Université de Picardie JulesVerne,
80000, Amiens, France,
hifi@u-picardie.fr; toufik.saadi@u-picardie.fr

***Abstract*—This paper proposes a parallel peer-to-peer (noted P2P) cooperative algorithm for approximately solving the constrained fixed-orientation two-staged two-dimensional cutting problem. The resolution process is based in three mechanisms: a beam-search search strategy, a strip generation filling procedure, and a upper bound applied for refining the selected paths.**

The algorithm explores, in parallel, a subset of elite nodes where each processor develops its own path according to its internal lists. the algorithms adapts to the number of processors available on the peer-to-peer platform by backuping the plateform the partial solutions. The computational investigation on the P2Pdc environment shows the good efficiency of the proposed algorithm.

***Keywords*-beam search stategy, branch-and-bound, cooperative, dynamic programming, heuristics, integer programming, peer-to-peer, optimization.**

I. Introduction

Several real industrial applications require the allocation of a set of rectangular items to a larger rectangular stock unit. For instance, when filling the pages of a newspaper, the editor has to arrange articles and advertisements, presented as rectangular areas, on pages of fixed dimensions. In addition, industrial applications require cutting or packing the largest number of items into a rectangular unit as to minimize the waste of the rectangular stock unit or to maximize the packed pieces. These cutting (or packing) problems are difficult combinatorial optimization problems, known in the literature as two-dimensional cutting (TDC) stock problems [9], [28].

Many TDC problems can be modeled as Integer Linear Programs (ILP) but solving them exactly is very difficult. For instance, the ILP models of the 2-staged TDC (2TDC) problem can not be solved effectively applying exact methods; that is, no real life sized problems can be solved within a reasonable runtime. Therefore, the only viable recourse is the development of hybrid/cooperative and special parallel methods. A cooperative method can be viewed as a combination of ILP with combinatorial solution procedures.

This paper investigates the use of a parallel cooperative method that approximately solves a variant of 2TDC: the constrained fixed-orientation guillotine 2TDC. The problem consists of cutting, from a large rectangle R of dimensions $L \times W$, a number of every small rectangle type (or piece or item) $i,\ i \in I = \{1, \ldots, n\}$, where item i is characterized by its dimensions $l_i \times w_i$, its demand b_i, and results in a profit c_i. Item $i,\ i \in I$ has a fixed orientation; that is, an item of dimensions $l \times w$ is different from an item of size $w \times l$ when $l \neq w$. In addition, in the final cutting pattern, each piece is produced by at most two guillotine cuts. A guillotine cut divides a rectangle from one of its edges to the opposite edge while being parallel to the two remaining edges. For instance, each piece in the example of Figure 1.(a) is obtained by two guillotine cuts:

(i) R is first divided into a set of *horizontal strips*, and
(ii) each generated horizontal strip is subsequently considered individually and chopped across its width.

The two-stage guillotine pattern of Figure 1.(a) is *exact* or *without trimming* since no additional cutting stage is required to extract each piece. On the other hand, the two-stage guillotine pattern of Figure 1.(b) is *non-exact* or *with trimming* since an additional cutting stage is necessary for extracting some of the pieces [2], [3], [12], [24]. Herein, $L,\ W,\ l_i,\ w_i$ and $b_i,\ i \in I$, are strictly positive integers, and the first cut is horizontal. Considering the case when the first cut is vertical is straightforward. It consists simply in inverting the dimenions of the items.

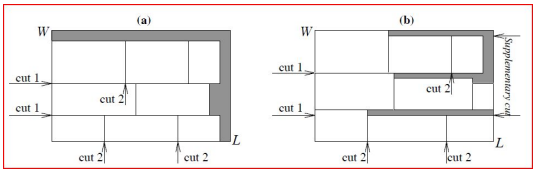

Figure 1. (a) An exact two-staged guillotine pattern (b) A non-exact two-staged guillotine pattern

The paper is organized as follows. Section II briefly exposes previous related work. Section III presents a strip generation procedure, and summarizes the beam-search algorithm proposed in [15].

Section IV describes the framework of the parallel algo-

The present work has been funded by ANR-07-CIS7-011 grant

rithm which can be viewed as a parallel implementation of a truncated branch-and-bound. Section V evaluates and discusses the performance of the proposed algorithm on several instances of the literature. Finally, Section VI summarizes the paper.

II. Background

This section provides relevant background information on 2TDC related literature and on standard beam search.

A. 2TDC Literature

The 2TDC problem was first introduced by Gilmore and Gomory [11], [12]. It has since received growing attention due to its wide spectrum of real-world applications [9], [28]. For the *unconstrained* 2TDC, few approximate and exact approaches are known in the literature [6], [14], [23]. For the *constrained* 2TDC, Hifi and Roucairol [19] proposed an approximate and an exact algorithm. The approximate algorithm chooses and combines some strips trough a strip packing heuristic whereas the exact algorithm combines the strips using a bottom-up strategy. Lodi and Monaci [22] developed integer linear programming formulations. Belov and Scheithauer [7] presented a combination of the two last approaches into a branch and price/cut algorithm. For the same problem, Hifi and M'Hallah [17] designed an exact branch and bound procedure in which a bottom-up strategy, some new lower/upper bounds, and new pruning strategies are used. With regard to heuristic methods, Hifi and M'Hallah [16] combined greedy type procedures with hill climbing strategies. Alvarez-Valdes *et al.* [2] put forward an approach which is based on constructing, improving and then combining solutions within the framework of GRASP methodology as well as the evolutionary approach known as Path Relinking. Hifi *and al.* [15], [18] devised beam search based heuristics which generate strips then combine a subset of them to fill the initial stock rectangle. The best combination maximizes the sum of the profits of the pieces included in the rectangle while respecting the upper demand constraint for each piece type. Finally, Hifi and Saadi [18] constructed a parallel algorithm whose principle follows the master-slave paradigm. This paper proposes a peer-to-peer parallel cooperative algorithm.

B. Beam search

Beam search is a truncated tree search procedure that was introduced in the context of scheduling [25], but has since been successfully applied to many other combinatorial optimization problems. It avoids exhaustive enumeration by performing a partial search of the solution space. In fact, at each level of the search tree, a subset of elite nodes is selected for further branching whereas the complementary subset of nodes is discarded forever. The selected *set of elite nodes* has at most β nodes where β is a prefixed *beam width*.

Figure 2. A standard beam search

1) *Initialization Step*
   a) Let β be the beam width.
   b) Set $B = \{\mu_0\}$ and $B_\beta = \emptyset$, where B is the set of nodes to be investigated, B_β the set of nodes branched out of the nodes in B, and μ_0 the root node of the search tree.
   c) If an initial feasible solution is available, set $z^\star$ to its objective function value; otherwise, set $z^\star = -\infty$.
2) *Iterative Step*
   `Repeat`
   a) Choose a node $\mu \in B$; branch out μ; remove μ from B and insert the created nodes (i.e., the offsprings of μ) into B_β.
   b) If a node μ of B_β is a leaf, then
      i) compute its objective function value z_μ;
      ii) if $z_\mu > z^\star$, update $z^\star$ and the incumbent solution;
      iii) remove μ from B_β.
   c) Assess the potential of each node of B_β using an evaluation operator.
   d) Rank the nodes of B_β in a non-increasing order of their values.
   e) Insert the $\min\{\beta, |B_\beta|\}$ best nodes of B_β into B; and set $B_\beta = \emptyset$.
   `Until` $B = \emptyset$.

Figure 2 gives a pseudo-code of a standard best-first beam search for a maximization problem. As with branch and bound, a lower bound can be used to fathom nodes. Indeed, if an initial feasible solution is available, then it is set as the incumbent solution and its value is assigned to $z^\star$. On the other hand, if no initial feasible solution is available, $z^\star$ is set to $-\infty$. Each node of B (which is the set of nodes to be further investigated) generates a set of offspring nodes, and appends them to B_β. If a node μ of B_β is a leaf (i.e, no further branching is possible out of μ), then its objective function value z_μ is computed and compared to $z^\star$. If $z_\mu > z^\star$, then the incumbent solution is set to the leaf node; $z^\star$ is then updated: $z^\star = z_\mu$; and μ is removed from B_β. The nodes of B_β are assessed using an evaluation operator, and ranked in a non-ascending order of their values. The first β nodes of B_β are then chosen as the elite nodes and transferred to B; whereas the remaining nodes of B_β are fathomed resulting in B_β being reset to the empty set. This process is reiterated until no further branching is possible; that is, until $B = \emptyset$.

III. Beam search algorithm for 2TDC

This section describes the main principle of the exact strip generation procedure, used for generating a set of optimal strips. Next, it reviews some existing filling procedures. Third, it explains how the strip generation and filling procedures are used within a serial beam search that approximately solves the problem at hand.

A. A strip generation procedure

The strip generation procedure (SGP) generates a set of *general optimal horizontal* strips. For a given strip of dimensions (L, ω), where $\omega \in \{w_1, \ldots, w_n\}$, SGP generates a general optimal horizontal strip −with pieces whose widths are less than or equal to the strip's width− according to the optimal solution of the following bounded knapsack problem:

$$BK_{L,\omega} \begin{cases} f_\omega(L) = \max & \sum_{i \in S_\omega} c_i x_i \\ \text{subject to} & \sum_{i \in S_\omega} l_i x_i \leq L \\ & x_i \leq b_i,\ x_i \text{ integer},\ i \in S_\omega, \end{cases}$$

where $S_\omega = \{k \in I \mid w_k \leq \omega\}$ denotes the set of pieces assigned to the strip (L, ω). x_i denotes the number of times piece type i appears in (L, ω) without exceeding the upper demand b_i of piece type i. Finally, $f_\omega(L)$ is the solution value of strip (L, ω).

Let $\overline{w}_1 < \ldots < \overline{w}_m$ be the set of distinct widths of the n pieces; that is, for $i = 1, \ldots, n,\ w_i \in \{\overline{w}_1, \ldots, \overline{w}_m\}$. As detailed in [15], solving $BK_{L,\overline{w}_m}$, using dynamic programming, generates all general optimal strips of width $\omega = 1, \ldots, \overline{w}_m$.

B. A basic filling procedure

The basic filling procedure (BFP) is a hybrid procedure that constructs a feasible solution for a sub-rectangle (L, Y) where $Y \leq W$ and there exists an integer $r \leq m$ such that $\overline{w}_r \leq Y < \overline{w}_{r+1}$. The cutting pattern associated to the subrectangle (L, Y), whose profit is $h_L(Y)$, is the optimal solution of the following ILP:

$$IP_{(L,Y)} \begin{cases} h_L(Y) = \max & \sum_{j=1}^{r} f_{\overline{w}_j}(L) y_j \\ \text{subject to} & \sum_{j=1}^{r} \overline{w}_j y_j \leq Y \\ & \sum_{j=1}^{r} \delta_{ij} y_j \leq b_i,\ i \in I \\ & y_j \leq a_j,\ y_j \text{ integer},\ j = 1, \ldots, r, \end{cases}$$

where $\delta_{ij},\ i \in I,\ j = 1, \ldots, r$, denotes the number of occurrences of the i^{th} piece in strip $(L, \overline{w}_j)$, and

$$a_j = \min\{\left\lfloor \frac{Y}{\overline{w}_j} \right\rfloor, \min_{i \in S_{\overline{w}_j}} \left\lfloor \frac{b_i}{\delta_{ij}} \right\rfloor \text{ with } \delta_{ij} > 0\}.$$

However, $IP_{(L,Y)}$ is NP-hard; thus, can not be solved exactly. Subsequently, $IP_{(L,Y)}$ is generally solved using the following approximate procedure.

1) Set $b_i^{res} = b_i$, for $i \in I$, where b_i^{res} is the residual demand for piece type i.
2) Select the feasible strip k with the highest profit to usage ratio; that is
$$\frac{f_{\overline{w}_k}(L)}{\sum_{i \in S_{\overline{w}_k}} \delta_{ik}} = \max_{j=1,\ldots,r} \frac{f_{\overline{w}_j}(L)}{\sum_{i \in S_{\overline{w}_j}} \delta_{ij}}.$$
3) Position strip k on (L, Y).
4) For every piece i in strip k, if $b_i^{res} < \delta_{ik}$, reduce δ_{ik} to b_i^{res}.
5) Update the current demand of every piece type i in strip k : $b_i^{res} = b_i^{res} - \delta_{ik}$.
6) Reorder the pieces in strip k in a non-increasing order of their widths.
7) Fill the remaining region of strip k using the *bottom-left guillotine procedure* (BLGP) explained below.
8) Update the remaining width of the subrectangle: $Y = Y - w_k$.
9) If there exists $i \in I$ such that $b_i^{res} > 0$ and $w_i \leq Y$, then
   a) if all r strips have been packed, apply BLGP on (L, Y) until no piece whose residual demand is strictly positive can be further packed, and stop;
   b) else goto 2.

   Else stop.

In fact, every time BFP considers a strip (L, y) for packing into the subrectangle (L, Y), it checks if the demand for each piece $i \in I$ is not exceeded. When packing a strip (L, y) into (L, Y) results in violating the demand constraint, BFP resorts to removing the surplus pieces as indicated in Figure 3.(b). This results in creating holes in the packed strip as shown in Figure 3.(c). To overcome this glitch, BFP reorders the pieces of each strip in a non-increasing order of their widths and fills the newly created hole using the greedy BLGP, as illustrated in Figure 3.(d).

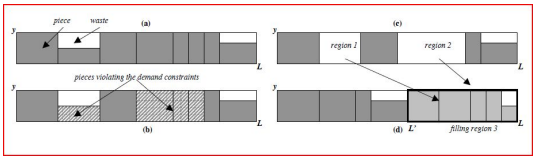

Figure 3. Description of BFP

To fill a substrip of dimensions $(L - L', y)$, BLGP solves the following knapsack problem:

$$\max\{\sum_{i \in I} c_i x_i \mid \sum_{i \in I} l_i x_i \leq L - L',\ w_i \leq y,\ x_i \leq b_i^{rest},\ x_i \text{ integer}\},$$

where $x_i,\ i \in I$, is the number of times piece type i appears in the substrip $(L - L', y)$.

In addition to filling substrips, BLGP can be applied as a heuristic to packing the items in R. Its solution value is a lower bound to the optimum for the problem at hand.

Figure 4. SBS: Beam search algorithm for 2TDC

`Input`: An instance of 2TDC problem.
`Output`: An approximate solution value $z^\star$.

Initialization Step

- Set $B = \left\{ [((L,0),(L,W)),\mathbf{b}] \right\}$ and $B_\beta = \emptyset$.
- Create all general strips associated to (L,W) by solving $BK_{L,\overline{w}_m}$.
- Solve the bounded knapsack associated with $\overline{U}_{(L,W)}$ to obtain the upper bounds $\overline{U}_{(L,Y)},\ Y = 0,\ldots,W$.
- Set $z^\star$ equal to an approximate solution value of $IP_{(L,W)}$.

Iterative Step

`Repeat`

The selection phase

- Select from B a node $\mu = \left[((L,W-Y),(L,Y)),\mathbf{b}_\mu^{res}\right]$ using a best-first search strategy.

The expansion phase

1) Branch out of μ generating the corresponding general strips (after updating the residual demands of the piece types) by solving $BK_{L,\overline{w}_r}$, where $\overline{w}_r \leq Y < \overline{w}_{r+1}$.
2) For each general strip $(L,\overline{w}_j),\ j = 1,\ldots,r$,
   a) create a new node $\nu_j = \left[((L,W-Y+\overline{w}_j),(L,Y-\overline{w}_j)),\mathbf{b}_{\nu_j}^{res}\right]$,
   b) perform a global evaluation based on the *upper bound* and *BFP's solution*; and
   c) update the best current solution value: $z^\star = \max\left\{z^\star,\ z_{\nu_j}^{Local} + BFP_{\nu_j}\right\}$.
3) Update the best potential value Γ using Equation 3.

The filtering phase

1) For each created new node $\nu_j = \left[((L,W-Y+\overline{w}_j),(L,Y-\overline{w}_j)),\mathbf{b}_{\nu_j}^{res}\right],\ j = 1,\ldots,r$,
   - if $z^\star < \overline{U}_{\nu_j} + z_{\nu_j}^{Local}$, insert ν_j into B_β, and compute the gap between $z_{\nu_j}^{Local}$ and Γ;
   - else discard ν_j.
2) Select the $min\{\beta, |B_\beta|\}$ nodes, and insert them in B.
3) Remove the node μ from B and reduce the set B_β to the empty set.

`Until` $B = \emptyset$.

- Exit with the best solution whose value is $z^\star$.

C. Beam search for 2TDC

Figure 4 describes the main steps of SBS, a best-first BS applied to 2TDC using a serial approach. SBS has three phases: initialization, iterative, and stopping criterion.

During the initialization phase, SBS solves $BK_{L,\overline{w}_m}$; thus, creates all the optimal general strips. It then solves approximately $IP_{(L,W)}$ using both BFP and BLGP, and retains the solution having the best value $z^\star$ as the incumbent's solution value.

Applying the iterative phase of SBS requires defining the nodes of the search tree and the branching mechanism. Herein, each node is characterized by the pair of sub-rectangles $(L,W-Y)$ and (L,Y), where $Y \leq W$, and by a vector of residual demand $\mathbf{b}^{res}$. The first component $(L,W-Y)$ is represented by its *partial feasible solution* obtained by combining a set of successive created strips, while the second component (L,Y) can be viewed as a complementary part that remains to be filled. The element $b_i^{res},\ i \in I$, of $\mathbf{b}^{res}$ represents the residual demand of piece type i given the partial feasible solution of the component $(L,W-Y)$. Thus, the root node is composed of the pair of sub-rectangles $(L,0)$ and (L,W) because no strip has been packed and the subrectangle (L,W) is yet to be packed, and by the residual demand vector $\mathbf{b}^{res} = \mathbf{b}$, where $\mathbf{b} = (b_1,\ldots,b_n)$, since no item has yet been assigned to the subrectangle $(L,0)$.

Branching out of a node $\mu = \left[\left((L,W-Y),(L,Y)\right),\mathbf{b}_\mu^{res}\right]$, where $Y \leq W$, is equivalent to packing a strip of dimensions $(L,\beta),\beta \leq Y,\ \overline{w}_r \leq \beta < \overline{w}_{r+1},\ r \leq m$, into the complementary part (L,Y). There are at most r branches emanating out of μ. Each branch $j,\ j = 1,\ldots,r$, corresponds to packing a strip $(L,\overline{w}_j)$, and results in an offspring node $\nu_j = \left[\left((L,W-Y+\overline{w}_j),(L,Y-\overline{w}_j)\right),\mathbf{b}_{\nu_j}^{res}\right]$. Specifically, branching out of μ involves the following steps.

1) Update the residual demand for each piece type $i,\ i \in I$.
2) Generate r *optimal general strips* by solving $BK_{L,\overline{w}_j},\ j = 1,\ldots,r$, with the updated demand for all piece types, using *dynamic programming*.
3) For each obtained strip $(L,\overline{w}_j), j = 1,\ldots,r$, create a branch out of μ as follows.
   a) Create the offspring node $\nu_j = \left[(L,W-Y+\overline{w}_j),(L,Y-\overline{w}_j),\mathbf{b}_{\nu_j}^{res}\right]$ by packing the strip $(L,\overline{w}_j)$ into the subrectangle (L,Y).
   b) Compute $z_{\nu_j}^{Local}$, the value of the *feasible solution* associated with the sub-rectangle $(L,W-Y+\overline{w}_j)$, where $z_{\nu_j}^{Local}$ is the sum of $z_{\mu_j}^{Local}$ and the optimal solution value of $BK_{L,\overline{w}_j}$.
   c) Apply the filling procedure BFP to $(L,Y-\overline{w}_j)$ to obtain a lower bound on the value of the *complementary sub-rectangle* $(L,Y-\overline{w}_j)$.
   d) Determine $\overline{U}_{(L,Y-\overline{w}_j)}$, a tight upper bound on the value of the *complementary sub-rectangle* $(L,Y-\overline{w}_j)$, where $\overline{U}_{(L,Y-\overline{w}_j)}$ is the optimal solution value of a bounded knapsack problem 1 and $f_{\overline{w}_p}(L),\ p = 1,\ldots,j$, is the profit generated by the strip $(L,\overline{w}_p)$ while t_p is the number of occurrences of that strip in $(L,\overline{w}_j)$. All upper bounds $\overline{U}_{(L,\omega)},\ \omega = 0,1,\ldots,W$, are obtained when computing $\overline{U}_{(L,W)}$ via dynamic program-

ming.

e) Compute $z_{\nu_j}^{Global}$, an upper bound on the value of ν_j by setting $z_{\nu_j}^{Global} = z_{\nu_j}^{Local} + \overline{U}_{(L,Y-\overline{w}_j)}$.

$$\overline{U}_{(L,Y-\overline{w}_j)} = \max\Big\{ \sum_{p=1}^{j} f_{\overline{w}_p}(L) t_p \mid \sum_{p=1}^{j} \overline{w}_p t_p \leq \overline{w}_j \Big\}, \quad (1)$$

Any node ν_j, $j = 1, \ldots, r$, emanating out of μ is discarded if

$$z_{\nu_j}^{Global} \leq z^{\star}. \quad (2)$$

The surviving nodes are then entered in B_β, and their best potential value Γ is computed:

$$\Gamma = \max_{\nu_j \in B_\beta} \{ z_{\nu_j}^{Global} \}. \quad (3)$$

Last, for each $\nu_j \in B_\beta$, the gap between $z_{\nu_j}^{Local}$ and Γ is computed, and the $min\{\beta, |B_\beta|\}$ nodes with the largest gaps are selected for further expansion while the others are permanently discarded.

The third and last phase defines the stopping criterion of SBS as $B = \emptyset$. In fact, this occurs when either all non-discarded nodes have complementary rectangles that can not fit any piece whose residual demand is strictly positive or all piece types have no residual demand. Thus, these nodes are leaves and can not enter B.

IV. A PARALLEL P2P GLOBAL BEAM SEARCH FOR 2TDC

This section is organized as follows. First, it discusses the parts of the beam search algorithm which can be implemented in parallel. Second, it indicates the design elements that the parallel algorithm has to take into consideration. Third, it presents the data structure used by the parallel algorithm. Fourth and last, it summarizes the main steps of the parallel algorithm. in addition, it presents some peer-to-peer elements of the algorithm and the data structures used in our implementation

A. Characteristics of the beam search for 2TDC

SBS can be successfully implemented as a parallel algorithm. Indeed, the most expensive tasks in terms of computational time are:

1) choosing the best promising node during the selection phase;
2) generating a set of optimal strips, calculating the complementary upper bounds, and getting complementary feasible and global solutions during the expansion phase; and
3) selecting the subset of elite nodes for further branching during the filtering phase.

Distributing the computational effort required by these three tasks among several processors reduces the overall computational time. Thus, a viable approach to this problem is a cooperative parallel algorithm that exploits the structure of the tree, its branch and fathom search, and bounding scheme that estimates the optimal solution of the problem. The cooperative parallel algorithm explores in parallel η nodes of the developed tree. Each processor guides its search-resolution process by maintaining a list of the nodes that have not yet been explored, and uses a best-first search strategy for selecting the successive sets of elite nodes.

B. Parallel and P2P design consideration

One of the challenges in efficiently parallelizing SBS is that the computational effort required by the selection, expansion, and filtering phases is highly variable and is, in some instances, unpredictable. This challenge could be overcome for the selection phase by sorting the node list in a specific order. However, dealing with this challenge for the expansion phase is a more complicated issue. Indeed, this phase (i) generates −at each selection− all the distinct general optimal strips, (ii) computes lower and upper bounds, and (iii) updates the best internal solution. Finally, the computational effort required by the filtering phase is generally unpredictable as it depends on the incumbent's solution value and on the quality of the bounds. Thus, the parallelization of SBS must incorporate mechanisms for *dynamic load balancing* of the computational effort.
moreover, the parallel algorithm owes to adapt to the dynamic character of P2P platform. Indeed, in P2P platforms, the number of processors taking part in the resolution can dynamically change. This dynamism imposes, (i) mechanisms for saving partials solutions found during the resolution by each processor, (ii) management of new resources (processors) connections and Finally, (iii) processors disconnections.

C. Data structure used in resolution

In the proposed implementation, the algorithm employs a load balancing protocol. For each processor k, $k = 1, \ldots, \eta$, it imposes a threshold limit ξ on the size of a *first internal list* B_k, and uses a *secondary internal list* $\overline{B}_k$ to store unselected local nodes. It broadcasts the first internal list to all processors periodically (every T time units). In addition, the algorithm resets the secondary internal list and sends it to a new processor (when a processor arrives and declares itself in the platform). We note that processor desactivation, involves the drop from these internal lists and the loss of the path developed by this processor.

D. Saving partials solutions

As noted in the preceding section, the processor desactivation in the P2P platform, involves loss of the path under development by this processor, that imposes periodic backups between the processors to avoid the loss of good realizable solutions. These backups are carried through the periodic broadcasts of the first internal list between the processors. These broadcasts keep the best realizable solution of every

processor on all the other processors, in order to prevent processor disconnection.

E. Managing new processor connection

When a new processor arrives and declares itself in the P2P platform, Each *processor* $k,\ k = 1, \ldots, \eta$, sends them, its internal lists. the *new processor* enters the nodes in its *fist internal list* and selects a starting node $\mu_k = \big[\big((L, W - Y),\ (L, Y)\big);\ \mathbf{b}^{res}_{\mu_k}\big]$, whose partial feasible solution's value is $z^{Local}_{\mu_k}$. The new processor refines the complementary upper bound by solving $BK_{L,Y}$ with the demand for piece type i set to its counterpart in $\mathbf{b}^{res}_{\mu_k}$, for all $i \in I$ such that $w_i \leq \overline{w}_r \leq Y < \overline{w}_{r+1}$.

When branching out of μ_k, the new processor creates r nodes. For each node $\nu_j,\ j = 1, \ldots, r$, the new processor k packs the general strip $(L, \overline{w}_j)$, which is the optimal solution to $BK_{L,\overline{w}_j}$, into the current sub-rectangle (L, Y), and computes the residual demand $\mathbf{b}^{res}_{\nu_j}$. It sets $\nu_j = \big[\big((L, W - Y + \overline{w}_j), (L, Y - \overline{w}_j)\big);\ \mathbf{b}^{res}_{\nu_j}\big]$, assesses $z^{Local}_{\nu_j}$, and evaluates $\overline{U}_{\nu_j}$. Next, the new processor k stores the r offspring nodes of μ_k into a secondary internal list $\overline{B}_k$, and save the best $min\{\beta, |\overline{B}_k|\}$ elite nodes of $\overline{B}_k$ to B_k, where the elite nodes are chosen as in SBS.

F. Detailed algorithm

This subsection presents the algorithm in each processor and the communication protocol that ensures their interaction.

1) Algorithm in each processor : In each processor , the algorithm is composed on four main steps: initialization, iterative step, backuping and stopping criterion.

The *initialization step* sets the fist internal list B_k to the root node $\mu := \Big[((L, 0), (L, W));\ \mathbf{b}\Big]$, and constructs the first m general optimal strips by applying dynamic programming to $BK_{L,\overline{w}_m}$. It associates to each general strip an offspring node $\nu_j,\ j = 1, \ldots, m$. It then computes via dynamic programming $\overline{U}_{(L,W)}$; so that, all internal upper bounds $\overline{U}_{(L,Y)},\ Y = 0, \ldots, W$, become known. Next, it applies a first filtering phase on the m created offspring nodes. It sets $z^\star = \max\limits_{j=1,\ldots,m} \big\{z^{Local}_j\big\}$, and searches for a feasible solution by removing the pieces causing the infeasibility of the solutions associated with $\overline{U}_{(L,Y)}$. Last, it chooses the β nodes having the largest $z^{Global}_{\nu_j} = z^{Local}_{\nu_j} + \overline{U}_{(L,W-\overline{w}_j)},\ j = 1, \ldots, m$, and saves them in it internal list.

The *iterative step* selects and removes from B_k the node $\mu = [((L, W - Y), (L, Y)), \mathbf{b}^{res}]$ having the largest global evaluation value. It then solves $BK_{L,\overline{w}_r}$, where $\overline{w}_r \leq Y < \overline{w}_{r+1}$, to generate r general optimal strips, and associates each strip with a branch emanating out of μ. For each created node $\nu_j,\ j = 1, \ldots, r$, slave k computes, in parallel, $z^{Local}_{\nu_j}$ and $\overline{U}_{\nu_j}$, where $z^{Local}_{\nu_j}$ is obtained by applying BFP with the residual demand updated. It then updates the best local solution value if $z^{Local}_{\nu_j} > z^\star$. The offspring node ν_j is retained (i.e., is entered in $\overline{B}_k$) if it could eventually lead to a better solution value; that is, if $z^{Global}_{\nu_j} \geq z^\star$. Next, processor k filters the nodes of $\overline{B}_k$, and only retains the best β nodes realizing better global evaluations (according to Equation (3)). Finally, it saves as many nodes as it can from $\overline{B}_k$ to B_k without exceeding the capacity ξ of B_k.

For the *backuping step*, the reaction of the processor depends on the type of output. There are three types of possible output.

1) A processor $k,\ k = 1, \ldots, \eta' \leq \eta$, has the capacity ξ of B_k exceeded. Thus, it saves residual nodes of $\overline{B}_k$ in its secondry interal list.
2) When a processor k finishes its assigned task without exceeding the capacity ξ of B_k or the runtime limit T, it sends to all processors its best local solution along with its value $z^{Local}_{\mu_k}$.
3) When it stops because it has exceeded its allocated runtime T, a processor $k,\ k = 1, \ldots, \eta' \leq \eta$, transfers to all processors its fist internal list and its best local solution.

The *stopping criterion* stops processor $k,\ k = 1, \ldots, \eta' \leq \eta$,, which returns the best solution value $z^\star$, when its interanl lists and message-queues are empty.

2) Communication Protocol: The implementation of the algorithm requires the management of three independent queues, as The three queues store different information (using a FIFO strategy). The first queue, Q_1, stores the best local solutions transferred by the processor. The second queue, Q_2, receives the residual nodes sent by the others processors. The third queue, Q_3, balances the load among the processors: it stocks the nodes emanating from the processors and redistributes them among the free ones. The three queues are managed independently; i.e., in parallel.

V. Computational results

This section evaluates the effectiveness of the proposed parallel algorithm (noted P2PCGBS) by testing it on set of instances extracted from [15], [16]. The dataset contains six large instances. The algorithm is run under Grid5000 platform and P2Pdc environment (see [29]).

P2PCGBS is run, with $T = 2$, secondes (backuping every 2 secondes) . Each instance is run twice: once with the first cut being horizontal, and once with the first cut being vertical. The optima of these instances are unknown; thus, for each instance, the solution obtained by P2PCGBS is compared, in Table I, to the best lower bound provided by the following algorithms: Cplex solver (v.9), the algorithm PAR of [16], and SBS of [15] with $\beta = 2$ and 4. The time limit is fixed to 3000 seconds for the Cplex solver and

PAR (as in [2], [15]), and to 900 seconds for SBS. Column 2 contains LB, a lower bound corresponding to the best solution value obtained by Cplex, PAR and SBS with $\beta = 2$ and 4. Column 3 displays the gap between LB and the value of the best integer feasible solution of model M1 proposed in [22] when solved using the Cplex. Column 4 tallies the gap between LB and PAR's solution value. Columns 5 and 7 compute respectively the gap between LB and SBS's solution value when $\beta = 2$, and 4, respectively, whereas columns 6 and 8 report the runtime of SBS when β is fixed to 2 and 4, respectively. Column 9 calculates the gap between LB and the solution value of P2PCGBS. Columns 10-12 tally P2PCGBS's runtime when $\beta = 4$, and $\eta = 1to6, 1to10$, and $1to20$ processors, respectively. In this table, the symbol $\circ$ signals an instance where the corresponding algorithm matches LB whereas the symbol $\diamond$ indicates that the algorithm does not terminate before its threshold run time limit.

The analysis of Table I indicates that Cplex produces moderate solutions. It obtains the best solution in only 2 out of 12 instances (WL1V and WL3V), and has an average gap of -3591.33. PAR, on the other hand, performs better than Cplex matching 9 out of 12 best solutions, with an average gap of -464.67. SBS with $\beta = 2$ outperforms both Cplex and PAR: it obtains all the best solutions within an average runtime of 334.24 seconds which is much smaller than the 3000 seconds allocated to Cplex and PAR. However, setting $\beta = 4$ deteriorates the performance of SBS which fails to identify any of the 12 best solutions and has an average gap of -218070.67. In fact, for these difficult instances, SBS with $\beta = 4$ requires more than 900 seconds to explore the best paths.

#Inst.	Best	Cplex Gap	PAR Gap	*SBS Algorithm* $\beta = 2$ Gap	$\beta = 2$ cpu	$\beta = 4$ Gap	$\beta = 4$ cpu	*P2PCGBS Algorithm* $\eta = 1to6$ Gap	$\eta = 1to6$ cpu	$\eta = 1to10$ cpu	$\eta = 1to20$ cpu
						First Cut Horizontal					
UL1	887393	-10711	-294	$\circ$	138.56	-187359	$\diamond$	**1993**	110.09	101.15	120.96
UL2	906237	-5314	$\circ$	$\circ$	310.13	-216139	$\diamond$	**1629**	230.81	190.27	133.66
UL3	1017575	-8544	$\circ$	$\circ$	347.89	-18810	$\diamond$	$\circ$	299.16	322.17	220.39
WL1	688840	-1054	$\circ$	$\circ$	410.09	-168781	$\diamond$	$\circ$	370.82	340.06	299.01
WL2	782566	-4538	$\circ$	$\circ$	454.76	-125136	$\diamond$	**1562**	401.55	499.14	380.18
WL3	899890	-2517	$\circ$	$\circ$	488.76	-486401	$\diamond$	$\circ$	432.12	499.05	460.53
						First Cut Vertical					
UL1	891516	-2754	$\circ$	$\circ$	120.87	-316084	$\diamond$	**8583**	118.81	122.19	99.91
UL2	908982	-2382	$\circ$	$\circ$	210.41	-230231	$\diamond$	**1039**	190.07	180.15	109.77
UL3	1022483	-3595	-3595	$\circ$	320.87	-124832	$\diamond$	$\circ$	254.08	179.08	158.31
WL1	694657	$\circ$	$\circ$	$\circ$	340.89	-177948	$\diamond$	$\circ$	221.95	180.11	160.65
WL2	783836	-1687	-1687	$\circ$	410.78	-293071	$\diamond$	**5985**	310.35	366.88	413.09
WL3	881931	$\circ$	$\circ$	$\circ$	456.81	-272056	$\diamond$	$\circ$	361.44	460.99	312.07
Av_Gap		-3591.33	-464.67	0.00		-218070.67		**1732.58**			
Av_cpu					334.24		900.00		275.10	286.77	322.37

Table I
PERFORMANCE OF THE ALGORITHM ON THE DATASET OF INSTANCES

Finally, P2PCGBS yields better results than Cplex, PAR and SBS. It improves six best solutions of the literature with the average improvement reaching 1732.58. This improvement is accompanied by a sizeable decrease of computational time. Indeed, the average runtime of P2PCGBS with $\eta = 1to6$ is 275.10 seconds; a time that is much smaller than the 334.24 seconds average runtime of SBS with $\beta = 2$; it represents a 17.69% decrease of the average runtime, a speedup factor $S(1-6) = 1.21$,.We notice on the other hand that setting $\eta = 1to10$ or $1to20$ idle P2PCGBS runtime which becomes 286.77 and 322.37 seconds, respectively. These results show that another factor impact P2PCGBS. Our investigations shows that frequency of processor's connection/disconnection impacts P2PCGBS.

VI. CONCLUSION

This paper solves the constrained fixed orientation guillotine two-staged two-dimensional cutting problem using a parallel peer-to-peer cooperative beam search algorithm, which can be viewed as a truncated branch and bound where only a subset of nodes are investigated. The proposed algorithm explores, in parallel a set of elite nodes selected following a best-first search strategy. The computational investigation shows that the parallel algorithm outperforms the serial version and other existing algorithms for set of benchmark problems consisting of large instances. It improves the best known optimum in many instances and matches the optimum or the best known solution for the others. These results show that another factor impact P2PCGBS. Our investigations shows that frequency of processor's connection/disconnection on P2PDC platform impacts P2PCGBS. The computational investigation showes, also, that another factor impact performance of the parallel algorithm. This factor is the stability of the peer-to-peer platform.

REFERENCES

[1] R. Alvarez-Valdes, A. Parajón, J.M. Tamarit, A tabu search algorithm for large-scale guillotine (un)constrained two-

dimensional cutting problems, *Computers and Operations Research, Vol.29, pp.925–947 (2002).*

[2] R. Alvarez-Valdes, R. Martì, A. Parajón, J.M. Tamarit, GRASP and path relinking for the two-dimensional two-staged cutting stock problem, INFORMS Journal on Computing, Vol.19, pp.1–12 (2007).

[3] M. Arenales, R. Morabito, An overview of and−or-graph approaches to cutting and packing problems, *Decision Making under Conditions of Uncertainly, ISBN 5-86911-161-7 pp.207–224 (1997).*

[4] BEA, MessageQ, programming guide, BEA Corporation (2007).

[5] BEA, MessageQ, Introduction to Message Queuing, *BEA Corporation (2007).*

[6] J.E. Beasley, Algorithms for unconstrained two-dimensional guillotine cutting, Journal of the Operational Research Society, Vol. 36, pp. 297–306 (1985).

[7] G. Belov, G. Scheithauer, A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting, *European Journal of Operational Research, (1985) Vol.171(1), pp.85–106 (2006)* .

[8] T. G. Crainic, B. Le Cun, C. Roucairol, Parallel branch-and-bound algorithms, Parallel combinatorial optimization, E-G. Talbi (Ed.), Wiley, ISBN: 978-0-471-72101-7 (2006).

[9] H. Dyckhoff, A typology of cutting and packing problems, *European Journal of Operational Research, Vol.44, pp.145–159 (1990).*

[10] B. Gendron, T.G. Crainic, Parallel branch-and-bound algorithms: survey and synthesis. Operations Research, Vol.42(6) pp.1042–1066 (1994).

[11] P.C. Gilmore, R.E. Gomory, A linear programming approach to the cutting stock problem, *Operations Research, Vol.9, pp.849–859 (1961).*

[12] P.C. Gilmore, R.E. Gomory, Multistage cutting problems of two and more dimensions, Operations Research, Vol.13, pp.94–119 (1965).

[13] A. Grama, V. Kumar, P. Pardalos, Parallel processing of discrete optimization problems, *Encyclpedia of Microcomputers, John Wiley & Sons (1993).*

[14] M. Hifi, Exact algorithms for large-scale unconstrained two and three staged cutting problems, Computational Optimization and Applications, Vol.18, pp.63–88 (2001).

[15] M. Hifi, R. M'Hallah, T. Saadi, Algorithms for the constrained two-staged two-dimensional cutting problem, *INFORMS Journal on Computing, Vol.20, pp.212– 221 (2008).*

[16] M. Hifi, R. M'Hallah, Strip generation algorithms for two-staged two-dimensional cutting stock problems, European Journal of Operational Research, Vol.172, pp.515–527 (2006).

[17] M. Hifi, R. M'Hallah, An exact algorithm for constrained two-dimensional two-staged cutting problems, *Operations Research, Vol.53, pp.140–150 (2005).*

[18] M. Hifi, T. Saadi, Un algorithme par génération de couches pour le problème de découpe à deux niveaux, 7ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision -ROADEF-, February 6-8, Lille, France (2006).

[19] M. Hifi, C. Roucairol, Approximate and exact algorithms for constrained (un)weighted two-dimensional two-staged cutting stock problems, *Journal of Combinatorial Optimization, Vol.5, pp.465–494 (2001).*

[20] M. Hifi, T. Saadi, Using strip generation procedures for solving constrained two-staged cutting problems, the Fifth ALIO/EURO conference on combinatorial optimization, ENST, Paris, France (2005).

[21] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack problems, *Springer (2003).*

[22] A. Lodi, M. Monaci, Integer linear programming models for 2-staged two-dimensional knapsack problems, Mathematical Programming, Vol.94, pp.257–278 (2003).

[23] R. Morabito, M. Arenales, Staged and constrained two-dimensional guillotine cutting problems: An and−or-graph approach, *European Journal of Operational Research, Vol.94(3) pp.548–560 (1996).*

[24] R. Morabito, V. Garcia, The cutting stock problem in hardboard industry: a case study, Computers and Operations Research, Vol.25 pp.469–485 (1998).

[25] P.S. Ow, T.E. Morton, Filtered beam search in scheduling, *International Journal of Production Research, Vol.26, pp.297–307 (1988).*

[26] G. Vairaktarakis. Analysis of algorithms for master-slave system, IEEE Transactions, Vol.29(11), pp.39–949 (1997).

[27] D. W. Walker, J. J. Dongarra, MPI: a standard Message Passing Interface, S*upercomputer, Vol.12, pp.56–68 (1996)*

[28] G. Wäscher, G., Haussner, H. Schumann, An improved typology of cutting and packing problems, European Journal of Operational Research, Vol.183, pp.1106-1108 (2007).

[29] N. The Tung, D. El Baz, P. Spiter, G. Jourjon, High Performance Peer-to-Peer Distributed Computing with Application to Obstacle Problem, *Parallel and Distributed processing workshops and phd forums (IPDPSW),2010 IEEE International symposium, Issue Date:19-23 April 2010 ,Atlanta GA.*