Algorithms for the Pagination Problem, a Bin Packing with Overlapping Items

Aristide Grange, Imed Kacem, Sébastien Martin

Algorithms for the Pagination Problem, a Bin Packing with Overlapping Items

Aristide Grange* Imed Kacem* Sébastien Martin *

May 6, 2016

Abstract

We introduce the strongly NP-complete *pagination* problem, an extension of Bin Packing where packing together two items may make them occupy less volume than the sum of their individual sizes. To achieve this property, an item is defined as a finite set of symbols from a given alphabet: while, in Bin Packing, any two such sets would be disjoint, in Pagination, they can share zero, one or more symbols. After formulating the problem as an integer linear program, we try to approximate its solutions with several families of algorithms: from straightforward adaptations of classical Bin Packing heuristics, to dedicated algorithms (greedy and non-greedy), to standard and grouping genetic algorithms. All of them are studied first theoretically, then experimentally on an extensive random test set. Based upon these data, we propose a predictive measure of the difficulty of a given instance, and finally recommend which algorithm should be used in which case, depending on either time constraints or quality requirements.

Keywords Optimization, Bin packing, Virtual-Machine Packing, Integer linear programming, Heuristics, Genetic algorithms

1 Introduction

Over the last decade, the *book* (as a physical object) has given ground to the multiplication of screens of all sizes. However, the *page* arguably remains the fundamental visual unity for presenting data, with degrees of dynamism varying from video to static images, from infinite scrolling (e.g., Windows Mobile interface) to semi-permanent display without energy consumption (e.g., electronic paper). Pagination is to information what Bin Packing is to matter. Both ask how to distribute a given set of items into the fewest number of fixed-size containers. But where Bin Packing generally handles concrete, distinct, one-piece objects, Pagination processes abstract groups of data: as soon as some data is shared by two groups packed in the same container, there is no need to repeat it twice.

1.1 Examples of practical applications

As an introductory example, consider the following problem. A publisher offers a collection of audio CDs for language learning. Say that a typical CD consists of one hundred short texts read by a native speaker; for each of them, a bilingual vocabulary of about twenty terms has to be printed out on the CD booklet. How best to do this? The most expansive option, both financially and environmentally, would require the impression of an 100-page booklet, i.e., with one page per audio text. But now suppose that each page can accommodate up to fifty terms. If all individual vocabularies are collated into one single glossary, no more than $100 \times 20/50 = 40$ pages are needed. This is the cheapest option, but the least convenient, since it forces the consumer to constantly leaf through the booklet while listening to a given text. To minimize cost without sacrificing usability, the publisher will be better off to pack into each page the most individual vocabularies as possible. If there were no common term between any two vocabularies, this problem would be Bin Packing; but obviously, most of the time, the vocabulary of a given text partially overlaps

*name.surname@univ-lorraine.fr, LCOMS EA7306, Université de Lorraine, Metz, FRANCE.

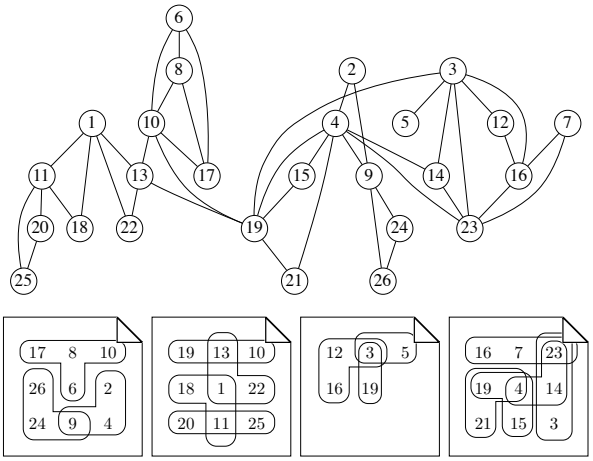

Figure 1: *Application of* Pagination *to the visualization of the cliques of a given graph. The cliques are split over a small number of pages (here, 4) with a given capacity (here, at most 9 vertices). To allow each clique to be entirely contained in a single page, some vertices may be repeated on different pages (here, 19 on 3 pages, and 3, 4, 10, 16 on 2 pages). The individual page layouts are not relevant.*

with several others: this is what we propose to call **the pagination problem**, in short Pagination. In this example, it takes advantage of the fact that the more terms are shared by two vocabularies, the cheaper to pack them together; as an added benefit, a good pagination will tend to associate to a same page the audio texts dealing with a same topic.

Coincidentally, it was in this context of linguistics that we first stumbled across Pagination. At that time, we needed to display selected clusters of morphologically related sinographs, or *kanjis*, on a pocket-sized screen. A full description of our initial purpose would be beyond the scope of this paper, and ultimately unnecessary, since the problem is in fact perfectly general. It only differs from Bin Packing by the nature of the items involved: instead of being atomic, each such item is a combination of elements, which themselves have two fundamental properties: first, they are all the same size (relatively to the bin capacity); and second, their combination is precisely what conveys the information we care about.

For instance, the members of a social network may interest us only to the extent that they are part of one or several friendship circles. Such groups of mutual friends are nothing more than the so-called cliques of a graph (Fig. 1, upper), but the cliques are notoriously difficult to extract visually. Visualizing them as separated sets of vertices is more effective, although quite redundant. A better compromise between compacity and clarity is attained by paginating these sets as in Fig. 1 (lower part). Note that, although no group is scattered across several pages, finding all the friends of a given person may require the consultation of several pages.

1.2 Definition and complexity

Our problem is not just about visualization. It extends to any need of segmentation of partially redundant data (see Section 1.3.1 for an application to virtual machine colocation). Let us define it in the most general way:

Definition 1. Pagination can be expressed as the following decision problem:

- *Instance:* a finite collection $\mathcal{T}$ of nonempty finite sets (the **tiles**[1]) of **symbols**, an integer $C > 0$ (the **capacity**) and an integer $n > 0$ (the **number of pages**).
- *Question:* does there exist an n-way partition (or **pagination**) $\mathcal{P}$ of $\mathcal{T}$ such that, for any tile set (or **page**[2]) p of $\mathcal{P}$, $|\cup_{t\in p} t| \leq C$?

Example 1. Figure 2 shows four valid paginations of the same set of tiles $\mathcal{T} = \{\{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d},\mathsf{e}\}, \{\mathsf{d},\mathsf{e},\mathsf{f}\}, \{\mathsf{e},\mathsf{f},\mathsf{g}\}, \{\mathsf{h},\mathsf{i},\mathsf{j},\mathsf{k}\}\}$. For easier reading, in the remainder of this paper, any set of symbols (especially, a tile) defined by extension (for instance, $\{\mathsf{w},\mathsf{o},\mathsf{r},\mathsf{d}\}$) will be represented as a gray block: word. Likewise, in any collection of symbol sets (especially $\mathcal{T}$, or a page), the separating commas will be omitted: {may june july}.

Proposition 1. Pagination *is NP-complete in the strong sense.*

Proof. Any given pagination $\mathcal{P}$ can be verified in polynomial time. Furthermore, Bin Packing is a

[1]The fact that Pagination generalizes Bin Packing has its counterpart in our terminology: the *items* become the *tiles*, since they can overlap like the tiles on a roof.

[2]Likewise, the move from concrete to abstract is reflected by the choice of the term *page* instead of *bin*.

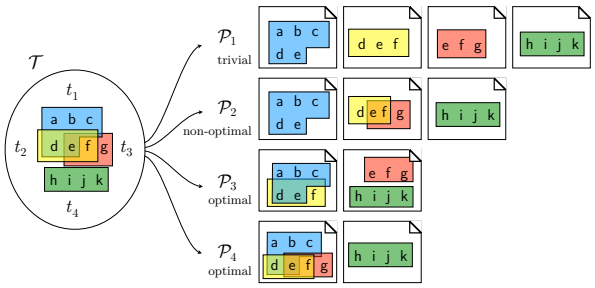

Figure 2: *For the tiles* $\mathcal{T} = \{$abcde def efg hijk$\}$ *and a capacity* $C = 7$, $\mathcal{P}_3 = (\{$abcde def$\}, \{$efg hijk$\})$ *is a possible optimal pagination in* $n = 2$ *pages.*

special case of PAGINATION with no shared symbol. Hence, the latter is at least as difficult as the former, which is strongly NP-complete. □

1.3 Related works

1.3.1 Packing of virtual machines

PAGINATION was first introduced in 2011 by Sindelar et al. [11] under the name of VM PACKING—VM standing for Virtual Machines. A virtual machine can be seen as a set of memory pages: although most of them belong exclusively to a given machine, some pages are identical across several machines. This happens more often as the configurations are close in terms of platform (Linux, Windows, Mac), system version, software libraries, or installed applications. If these virtual machines run on the same physical server, their common memory pages (13 % on average) can actually be pooled in order to spare ressources.

VM PACKING is expressed as follows: being given a collection of VM (our tiles), each consisting in a set of memory pages (our symbols), allocate them to a minimal number of servers (our pages) having a capacity of P memory pages (of C symbols).

In its formulation, this problem appears to be exactly the same as ours; but when it comes to finding solutions, the authors rely on some very domain-specific hierarchical properties. Crucially, they consider each tile (if we are allowed to return to the more general terminology used in the present article) as made up of symbols of various levels. In the first model (*tree*), when two tiles share a symbol of a given level, they must share all their symbols of same or lower level. In the second model (*clustered-tree*), the inequality is strict (lower level sharing only); moreover, the number of symbols in a given level is bounded by an arbitrary constant. This does not mean that the other types of sharing are forbidden; but that these ones, comparatively rare among VM, will be ignored by the packing algorithm. Hence, the involved servers will certainly be under capacity, but on the other hand will more easily absorb the size variations caused by changes in the shared memory pages. These hierarchical restrictions lead to provably-good approximations algorithms, but say little about the general model, which is left open by the authors.

1.3.2 Hypergraph partitioning

Let us recall that a **hypergraph** $G = (V, E)$ consists of a set of vertices V and a set of hyperedges E, where each element of E is a subset of V [1]. Bearing this in mind, it is easy to see the left-hand part of Fig. 2 as a subset-based drawing [10] of a hypergraph mapping the instance, namely with $V = \mathcal{A}$ (the vertices are the symbols) and $E = \mathcal{T}$ (the hyperedges are the tiles). Since PAGINATION is a partitioning problem, it is natural to ask whether we could take advantage of the extensive literature on HYPERGRAPH PARTITIONING. In [9] for instance, the latter problem is defined as "partitioning the vertices of a hypergraph into k roughly equal parts, such that a certain objective function defined over the hyperedges is optimized". Although our capacity constraint on the symbols is reminiscent of this "roughly equal" number of vertices, the main purpose of PAGINATION is to partition the tiles (i.e., the hyperedges), and certainly not the symbols (i.e., the vertices).

So, what happens if we try to exchange the roles of the tiles and the symbols? This gives the following alternative hypergraph representation of the instances of PAGINATION: $V = \mathcal{T}$ (the vertices are the tiles) and $E = \mathcal{A}$ (the hyperedges are the symbols). To be more specific, a symbol shared by several tiles connects them as a hyperedge, while a proper symbol (i.e., a symbol belonging to one tile only) connects this tile to itself as a hyperloop. Now, paginating G indeed amounts to partitioning the vertices, but in the meantime two issues have

arisen. First, we do not care if each part contains roughly the same number of tiles: we want instead that the number of involved hyperedges is at most equal to C. Second, we have to express our objective function (minimizing the number of pages) on the hyperedges (the symbols). To cite [9] again: "a commonly used objective function is to minimize the number of hyperedges that span different partitions". At first sight, it would indeed seem reasonable to minimize the number of replications of a given symbol across the pages. However, this leads to another impasse:

Proposition 2. *Minimizing the number of pages and minimizing the number of symbol replications are not equivalent.*

Counterexample. Let $C = 5$ and $\mathcal{T} = \{$ a1 357 a2 468 $\}$. The optimal pagination $\{\{$ a1 357 $\}, \{$ a2 468 $\}\}$ minimizes the number of pages (2), but not the number of replicas (symbol a is replicated once). Conversely, the non-optimal pagination $\{\{$ a1 a2 $\}, \{$ 357 $\}, \{$ 468 $\}\}$ minimizes the number of symbol replications (0), but not the number of pages (3). □

Therefore, contrary to appearances, PAGINATION has very little in common with HYPERGRAPH PARTITIONING. As a final remark, we should mention that the problem studied in [5], and coincidentally named *pagination* (by reference to the fixed-length contiguous block of virtual memory, or memory-pages), is in fact a special case of HYPERGRAPH PARTITIONING, where the objective is to minimize the total weight of edge-cuts in a weighted graph, with an upper bound on the size of the parts.

The rest of this paper is organized as follows. In Section 2, we formulate PAGINATION as an integer linear program (ILP), and introduce the various metrics and simplifying assumptions needed by our algorithms. Then, in Section 3, we describe several heuristics and meta-heuristics for the problem. Finally, we compare the results produced by all the algorithms (exact or not) in Section 4, and conclude in Section 5.

2 Theoretical tools

To look on our problem from another perspective, let us formulate it as an ILP. In addition, we will be able to solve some (admittedly simple) instances with a generic optimization software. Thereafter, the introduction of several supplementary concepts will permit us to actually generate the instances, and to describe our own algorithms for tackling the largest ones.

2.1 Integer linear programming model

Numberings We use the following sets of indexes:

- $A = \{i : i = 1, \ldots, |\mathcal{A}|\}$ for the symbols;
- $T = \{j : j = 1, \ldots, |\mathcal{T}|\}$ for the tiles;
- $P = \{k : k \in \mathbb{N}\}$ for the pages[3].

Constants

- C is an integer nonnegative capacity;
- a^i_j is an assignment of symbols to tiles: $\forall i \in A, \forall j \in T, a^i_j = 1$ if $i \in t_j$, and 0 otherwise.

Decision variables For all $i \in A$, $j \in T$ and $k \in P$, we define:

- x^i_k as equal to 1 if symbol i is present on page k, and 0 otherwise (pagination of the symbols);
- y^j_k as equal to 1 if tile j is present on page k, and 0 otherwise (pagination of the tiles);
- z_k as equal to 1 if page k is used, and 0 otherwise (unitary usage of the pages).

It is worth noting that $x^i_k = \max_{j \in T}(a^i_j y^j_k)$ and $z_k = \max_{j \in T}(y^j_k)$. The mathematical model of PAGINATION is thus entirely specified with y^j_k: the introduction of these auxiliary variables is only used to achieve the linearity of its formulation.

[3] For the sake of simplicity, we assume that an infinite number of pages are available. In practice, prior to the calculations, this number will be limited to a reasonable value, either given by an heuristics, or $|\mathcal{T}|$ (one tile per page) in the worst case.

Linear program A possible ILP formulation of PAGINATION is:

$$\text{min.} \sum_{k \in P} z_k$$

$$\text{s. t.} \sum_{k \in P} y_k^j = 1, \qquad \forall j \in T \quad (1)$$

$$z_k \geq x_k^i, \qquad \forall i \in A, \forall k \in P \quad (2)$$

$$\sum_{i \in A} x_k^i \leq z_k C, \qquad \forall k \in P \quad (3)$$

$$x_k^i \geq a_j^i y_k^j, \qquad \forall i \in A, \forall j \in T, \forall k \in P \quad (4)$$

$$x_k^i,\ y_k^j, z_k \text{ all binary}, \quad \forall i \in A, \forall j \in T, \forall k \in P \quad (5)$$

Eq. 1 assigns each tile to exactly one page. Eq. 2 ensures that a page is used as soon as it contains one symbol. Conversely, thanks to the objective function, every used page contains at least one symbol. From Eq. 3, a page cannot contain more than C symbols. Eq. 4 guarantees that, when a tile belongs to a page, this page includes all its symbols[4]. The integrality constraints of the auxiliary variables of Eq. 5 could be relaxed.

2.2 Counting of the symbols

Definition 2 (metrics). Let α be a symbol, t a tile and p a set of tiles (most often, a page). Then:

1. the **size** $|t|$ of t is its number of symbols;
2. the **volume** $\mathcal{V}(p)$ of p is its number of distinct symbols: $\mathcal{V}(p) = |\cup_{t \in p} t|$; and its complement $C - \mathcal{V}(p)$, the **loss** on p;
3. by contrast, the **cardinality** $\text{Card}(p)$ is the total number of symbols (distinct or not) in p: $\text{Card}(p) = \sum_{t \in p} |t|$.
4. the **multiplicity** $\mu_p(\alpha)$ counts the occurrences of α in the tiles of p: $\mu_p(\alpha) = |\{t \in p : \alpha \in t\}|$;
5. the **relative size** of t on p is the sum of the reciprocals of the multiplicities of the symbols of t in p: $|t|_p = \sum_{\alpha \in t} \frac{1}{\mu_p(\alpha)}$.

In recognition of the fact that a given symbol may occur several times on the same page, the terms *cardinality* and *multiplicity* are borrowed from the multiset theory [13].

[4] A tighter, but slightly less explicit formulation is still possible: $x_k^i \geq y_k^j, \forall i \in t_j, \forall j \in T, \forall k \in P$.

Example 2. In pagination $\mathcal{P}_3$ of Fig. 2: size $|t_1| = 5$, volume $\mathcal{V}(p_1) = 6$ (loss: $7 - 6$), cardinality $\text{Card}(p_1) = 5 + 3$, multiplicity $\mu_{p_1}(\mathsf{e}) = 2$, relative size $|t_2|_{p_1} = 1/2 + 1/2 + 1$.

These definitions can be extended to more than one page by summing the involved values:

Example 3. In the same figure, volume $\mathcal{V}(\mathcal{P}_3) = 6 + 7$ (loss: $7 - 1 + 7 - 7$), cardinality $\text{Card}(\mathcal{P}_3) = \text{Card}(\mathcal{T}) = 5 + 3 + 3 + 4$, multiplicity $\mu_{\mathcal{P}_3}(\mathsf{e}) = \mu_{\mathcal{T}}(\mathsf{e}) = 2 + 1$, relative size $|t_2|_{\mathcal{P}_3} = |t_2|_{\mathcal{T}} = 1/2 + 1/3 + 1/2$.

We are now able to express a first interesting difference with BIN PACKING:

Proposition 3. *A pagination whose loss is minimal is not necessarily optimal.*

Counterexample. For $C = 4$, the tile set $\mathcal{T} = \{$ 12 13 23 ab ac bc $\}$ has a loss of 1+1 on the optimal pagination ($\{$ 12 13 23 $\}, \{$ ab ac bc $\}$); but a loss of $0 + 0 + 0$ on the non-optimal pagination ($\{$ 12 ab $\}, \{$ 13 ac $\}, \{$ 23 bc $\}$). □

As seen in the previous example, for the complete page set, multiplicity, cardinality and relative size do not depend on the pagination. Then:

Proposition 4. *A pagination is optimal if and only if the average cardinality of its pages is maximal.*

Proof. Since the sum of the page cardinalities is always equal to $\sum_{t \in \mathcal{T}} |t|$, its average depends only of the size n of the pagination. Hence, minimizing this size or that average is equivalent. □

2.3 Simplifying assumptions

Definition 1 encompasses many instances whose pagination is either infeasible (e.g., one tile exceeds the capacity), trivial (e.g., all tiles can fit in one page) or reducible (e.g., one tile is a subset of another one). The purpose of this subsection is to bring us closer to the core of the problem, by ruling out as many such degenerated cases as possible. For each one, we prove that there is nothing lost in ignoring the corresponding instances. In most of the proofs, for convenience, we reason on the associated optimization problem (i.e., where the aim is to minimize the number of pages).

Rule 1. *No tile is included in another one:* $\forall (t, t') \in \mathcal{T}^2, t \subseteq t' \Leftrightarrow t = t'$.

Proof. If $t \subseteq t'$, then t and t' can be put together on the same page of the optimal solution. □

In other words, the tile sets we deal with are Sperner families [12]. It follows that:

Corollary 1 (Sperner's Theorem)**.** *A page of capacity C contains at most $\binom{C}{\lfloor C/2 \rfloor}$ tiles.*

Rule 2. *No tile contains all the symbols:* $\nexists t \in \mathcal{T} : t = \mathcal{A}$.

Proof. Direct consequence of Rule 1. □

Rule 3. *No tile contains less than two symbols:* $\nexists t \in \mathcal{T} : |t| < 2$.

Proof. By definition, no tile is empty. Suppose there exists a tile t of size 1, and let $\mathcal{P}'$ be an optimal pagination of the reduced instance $\mathcal{T} \setminus \{t\}$. If there exists a page $p' \in \mathcal{P}'$ such that $|p'| < C$, then adding t on p' produces an optimal pagination of $\mathcal{T}$. If however all pages are saturated, Rule 1 ensures that $\mathcal{P} = \mathcal{P}' \cup \{\{t\}\}$ is an optimal pagination of $\mathcal{T}$. □

Rule 4. *Each tile has less than C symbols:* $\forall t \in \mathcal{T}, |t| < C$.

Proof. Let t be an arbitrary tile. If $|t| > C$, the problem has no solution. If $|t| = C$, then no other tile t' could appear on the same page as t without violating Rule 1. Let $\mathcal{P}'$ be an optimal pagination of the reduced instance $\mathcal{T} \setminus \{t\}$. Then $\mathcal{P} = \mathcal{P}' \cup \{\{t\}\}$ is an optimal pagination of $\mathcal{T}$. □

Rule 5. *No symbol is shared by all tiles:* $\nexists \alpha \in \mathcal{A} : \forall t \in \mathcal{T}, \alpha \in t$.

Proof. Otherwise, let $\mathcal{P}'$ be an optimal pagination, for a capacity of $C - 1$, of the reduced instance $\mathcal{T}' = \{t \setminus \alpha : t \in \mathcal{T}\}$ (Rule 3 ensures that no tile of $\mathcal{T}'$ is empty). Then adding α to each page of $\mathcal{P}'$ gives an optimal pagination of $\mathcal{T}$ for capacity C. □

In other words, $\mathcal{T}$ has not the Helly property [4]. Contrast this with VM Packing [11], where the mere existence of *root* symbols violates this rule.

Rule 6. *Each symbol belongs to at least one tile:* $\forall \alpha \in \mathcal{A}, \exists t \in \mathcal{T} : \alpha \in t$.

Proof. By Definition 1 of an instance. □

Rule 7. *Each tile is compatible with at least another one:* $\forall t \in \mathcal{T}, \exists t' \in \mathcal{T} \setminus \{t\} : |t \cup t'| \leq C$.

Proof. If there exists a tile t not compatible with any other, any solution should devote a complete page to t. Same conclusion as in the proof of Rule 4. □

Rule 8. *Capacity $C > 2$.*

Proof. From Rules 3 and 4, $\forall t \in \mathcal{T}, 1 < |t| < C$. □

Rule 9. *Capacity $C < |\mathcal{A}|$.*

Proof. Otherwise, all tiles could fit in one page. □

To sum up, an optimal solution of an instance violating Rules 1, 3, 4 (with $|t| = C$), 5, 6 or 7, could be deduced from an optimal solution of this instance deprived of the offending tiles or symbols; an instance violating Rule 4 (with $|t| > C$) would be unfeasible; an instance violating Rules 2, 8 or 9 would be trivial. All these rules can be tested in polynomial time, and are actually required by our instance generator (Section 4.1).

3 Heuristics

In this section, we investigate four families of heuristics for Pagination, from the simplest to the most sophisticated one. The first family consists of direct adaptations of the well-studied Bin Packing's greedy Any Fit algorithms; we show that, in Pagination, their approximation factor cannot be bounded. The second family is similar, but rely on the overlapping property specific to our problem; a general instance is devised, which shows that the approximation factor is at least 3. With the third algorithm, we leave the realm of the greedy decisions for a slightly more complex, but hopefully more efficient strategy, based upon a queue. Finally, we present two genetic algorithms and discuss which encoding and cost function are better suitable to Pagination. All of this is carried out from a theoretical perspective, the next section being devoted to the presentation of our benchmarks.

3.1 Greedy heuristics inspired from Bin Packing

3.1.1 Definitions

The question naturally arises of how the Bin Packing classical approximation algorithms [2] behave in the more general case of Pagination. Let us enumerate a few of them with our terminology:

- Next Fit simply stores each new tile in the last created page or, should it exceed the capacity, in a newly created page.
- First Fit rescans sequentially the pages already created, and puts the new tile in the first page where it fits.
- Best Fit always chooses the fullest possible page. Such a criterion needs to be clarified in our generalization of Bin Packing: fullest *before* or *after* having put the new tile? This alternative should give rise to two variants.
- Worst Fit, contrary to Best Fit, favors the less full page.
- Almost Worst Fit is a variant of Worst Fit opting for the *second* less full page.

These algorithms are known under the collective name of Any Fit (AF). In their offline version, presorting the items by size has a positive impact on their packing; but for Pagination, such a sorting criterion would obviously be defective: due to possible merges, a large tile often occupies less volume than a small one.

3.1.2 A general unfavorable case

Let $\mathcal{A}_\alpha = \{\alpha_1, ..., \alpha_C\}$ and $\mathcal{A}_\beta = \{\mathsf{a}, ..., \beta_C\}$ be two disjoint subsets of size C, assumed to be even ($C = 4$ on Fig. 3). Let $\mathcal{T}_\alpha = \binom{\mathcal{A}_\alpha}{C/2}$ and $\mathcal{T}_\beta = \binom{\mathcal{A}_\beta}{C/2}$ be the set of the $\frac{C}{2}$-combinations of $\mathcal{A}_\alpha$ and $\mathcal{A}_\beta$ (respectively). Then $\mathcal{P}_{\text{opt}} = \{\mathcal{T}_\alpha, \mathcal{T}_\beta\}$ is an optimal pagination of $\mathcal{T} = \mathcal{T}_\alpha \cup \mathcal{T}_\beta$ in 2 pages.

Now, let us feed these tiles to any of our AF algorithms, but only after having sorted them in the worst order: since all of our tiles have the same size $C/2$, we are indeed free to organize them the way we want. The most unfavorable scheduling simply involves alternating the tiles of $\mathcal{T}_\alpha$ and $\mathcal{T}_\beta$. In this way, regardless of the selected AF algorithm, the first two tiles would saturate the first page, the next two, the second page, and so on. In total, an AF algorithm would then create $n = |\mathcal{T}_\alpha| = |\mathcal{T}_\beta| = \binom{C}{C/2} = \frac{C!}{(C/2)!^2}$ pages instead of 2. In this family, the approximation factor is thus unbounded. This stands in stark contrast to the efficiency of the AF algorithms on Bin Packing—where, for instance, First Fit was recently shown [3] to achieve an absolute factor of exactly 1.7.

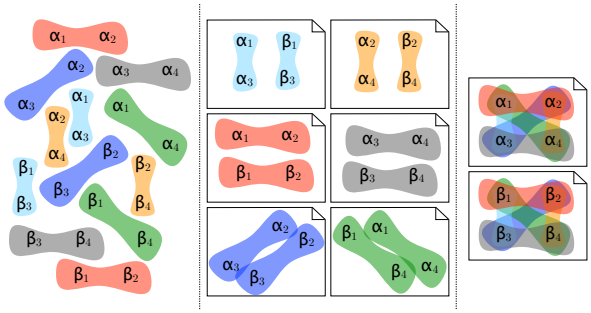

Figure 3: *To the left, a set* $\mathcal{T} = \mathcal{T}_\alpha \cup \mathcal{T}_\beta$ *of* $2\binom{4}{2} = 12$ *tiles; at the center, a pagination of capacity 4 calculated by the* Any Fit *algorithms when the tiles of* $\mathcal{T}_\alpha$ *and* $\mathcal{T}_\beta$ *are fed alternatively; to the right, an optimal pagination.*

3.1.3 Study case: First Fit algorithm

For our tests, we have chosen to focus on First Fit. Its worst case complexity can be analyzed as follows. There are $|\mathcal{T}|$ tiles to paginate. At worst, according to Rule 7, there exists only one tile compatible with any other, resulting in $|\mathcal{T}| - 1$ pages. Lastly, each set intersection costs a linear time in the size of the candidate tile. Hence, the overall complexity is $\mathcal{O}(|\mathcal{T}|^2 \text{Card}(\mathcal{T}))$.

Note that in Bin Packing, for n items, an appropriate data structure can reduce the straightforward quadratic complexity to $\mathcal{O}(n \log(n))$ [8]. This optimization is not applicable here, where the current volume of a given page says little about its ability to accommodate a given tile.

3.2 Specialized greedy heuristics

3.2.1 Definition

We can easily improve on the Bin Packing heuristics by taking into account the merging property of

PAGINATION items.

The corresponding offline greedy algorithms will always select, among the remaining tiles, the one which minimizes or maximizes a certain combination of the metrics introduced in Definition 2 (e.g., volume, multiplicity, relative size, etc.).

Before introducing the particular heuristics we have the most thoroughly tested, BEST FUSION, let us mark out the limitations of the algorithms of this family. We will design an extensible instance whose all tiles are equivalent with respect to these metrics. As such, they may be taken in any order, including the worst.

3.2.2 A general unfavorable case

Take an even capacity C, and $\mathcal{A}_0 = \{\alpha_1, ..., \alpha_C\}$ a first subset of $\mathcal{A}$ Let $\mathcal{T}_0 = \binom{\mathcal{A}_0}{C-1}$ the $(C-1)$-combinations of $\mathcal{A}_0$, which amount to C tiles of size $C-1$. From now on, to better understand the general construction, we will illustrate each step on an example with $C = 4$:

$$\mathcal{A}_0 = \boxed{1234}, \qquad \mathcal{T}_0 = \{\boxed{123}\ \boxed{124}\ \boxed{134}\ \boxed{234}\}$$

Introduce two more symbols, namely a and b, which will be used to lock the pages. Partition $\mathcal{T}$ into $\frac{C}{2}$ couples of tiles: $(t_1, t_2), (t_3, t_4), ..., (t_{C-1}, t_C)$. From each such couple, form $\mathcal{A}_i = (t_{2i-1} \cap t_{2i}) \cup \{\mathsf{a}, \mathsf{b}\}$ with $1 \leq i \leq \frac{C}{2}$, a subset of $\mathcal{A}$. In the same way than on $\mathcal{A}_0$, define $\mathcal{T}_i = \binom{\mathcal{A}_i}{C-1}$ on each $\mathcal{A}_i$:

$$\mathcal{A}_1 = \boxed{12ab}, \qquad \mathcal{T}_1 = \{\boxed{12a}\ \boxed{12b}\ \boxed{1ab}\ \boxed{2ab}\}$$
$$\mathcal{A}_2 = \boxed{34ab}, \qquad \mathcal{T}_2 = \{\boxed{34a}\ \boxed{34b}\ \boxed{3ab}\ \boxed{4ab}\}$$

The construction of this instance was in the same time the construction of an optimal solution to it. Indeed, for the whole tile set, $|\mathcal{P}_{\text{opt}}| = \frac{C}{2} + 1$:

$$p_1 = \{\boxed{123}\ \boxed{124}\ \boxed{134}\ \boxed{234}\} \to \boxed{1234}$$
$$p_2 = \{\boxed{12a}\ \boxed{12b}\ \boxed{1ab}\ \boxed{2ab}\} \to \boxed{12ab}$$
$$p_3 = \{\boxed{34a}\ \boxed{34b}\ \boxed{3ab}\ \boxed{4ab}\} \to \boxed{34ab}$$

Now, let us construct a non-optimal pagination. This is done by pairing each tile of $\mathcal{T}_0$ with a tile including a "locking" symbol. Specifically, on pages $2i-1$ and $2i$, put respectively t_{2i-1} and t_{2i}, and lock these pages with tiles $(t_{2i-1} \cap t_{2i}) \cup \{\mathsf{a}\}$ and $(t_{2i-1} \cap t_{2i}) \cup \{\mathsf{b}\}$ (respectively) from $\mathcal{T}_i$. This process creates C locked pages:

$$p'_1 = \{\boxed{123}\ \boxed{12a}\} \to \boxed{123a}$$
$$p'_2 = \{\boxed{124}\ \boxed{12b}\} \to \boxed{124b}$$
$$p'_3 = \{\boxed{134}\ \boxed{34a}\} \to \boxed{134a}$$
$$p'_4 = \{\boxed{234}\ \boxed{34b}\} \to \boxed{234b}$$

But what enables us to construct such inefficient two-tile pages? All tiles having the same size, it is clear that the first one can be chosen arbitrarily. Now, let t' be an eligible second tile, and p' the resulting page. Then, all t' are equivalent under our various metrics: same size $|t'| = C-1$, same volume $\mathcal{V}(p') = C$, same cardinality $\text{Card}(p') = 2(C-1)$, same relative size $|t'|_{p'} = \frac{C-2}{2} + 1 = \frac{C}{2}$. So, nothing prevents our greedy algorithms to systematically select the worst candidate.

If $C > 2$, the $C-2$ tiles including both a and b still remain in every tile set (but $\mathcal{T}_0$). For each one, gather its tiles on a new page:

$$p'_5 = \{\boxed{1ab}\ \boxed{2ab}\} \to \boxed{12ab}$$
$$p'_6 = \{\boxed{3ab}\ \boxed{4ab}\} \to \boxed{34ab}$$

Finally, we have obtained a non-optimal pagination $\mathcal{P}$ totaling $C + \frac{C}{2} = \frac{3C}{2}$ pages. Hence, for a given even capacity C, any greedy algorithm of this family may yield $\frac{|\mathcal{P}|}{|\mathcal{P}_{\text{opt}}|} = \frac{3C}{C+2}$ times more pages than the optimal. In other words, its approximation factor is at least 3.

3.2.3 Study case: Best Fusion algorithm

In our benchmarks, the following criterion was used: for each tile t, let p be the eligible page on which the relative size $|t|_p$ is minimal. If $|t|_p < |t|$, then put t on p; otherwise, put t on a new page.

The worst-case complexity is the same as in FIRST FIT, i.e., $\mathcal{O}(|\mathcal{T}|^2 \text{Card}(\mathcal{T}))$.

3.3 Overload-and-Remove algorithm

The following non-greedy approach has the ability to reconsider past choices when better opportunities arise. The main idea is to add a given tile t to the page p on which t has the minimal relative size, even if this addition actually overloads

p. In this case, the algorithm immediately tries to unload p by removing the tile(s) t' of strictly lesser size/relative size ratio. The termination is guaranteed by forbidding removed tiles to reenter the same page. At the end, the possible remaining overloaded pages are suppressed, and their tiles redistributed by First Fit.

Algorithm: Overload-and-Remove

$\mathcal{Q} \leftarrow$ queue containing all the tiles of $\mathcal{T}$
$\mathcal{P} \leftarrow$ pagination consisting of one empty page
while $\mathcal{Q}$ is nonempty:
| $t \leftarrow \mathcal{Q}$.dequeue()
| $\mathcal{P}_t \leftarrow$ pages of $\mathcal{P}$ where t has never been put on
| **if** $\mathcal{P}_t$ has no page p such that $|t|_p < |t|$:
| | add to $\mathcal{P}$ a new page consisting solely of $\{t\}$
| | **continue with next iteration**
| $p \leftarrow$ page p of $\mathcal{P}_t$ such that $|t|_p$ is minimal
| put tile t on page p
| **while** $\mathcal{V}(p) > C$ **and** $\exists t_1, t_2 \in p^2 : \frac{|t_1|}{|t_1|_p} \neq \frac{|t_2|}{|t_2|_p}$:
| | remove from p one tile t' minimizing $|t'|/|t'|_p$
| | $\mathcal{Q}$.enqueue(t')
remove all the overloaded pages from $\mathcal{P}$
put their tiles back in $\mathcal{P}$ (by First Fit)

In the worst case, a given tile might successively overload and be removed from all pages, whose total number is at most $|\mathcal{T}|$. Each trial requires one set intersection on each page. Hence, the overall complexity is $\mathcal{O}(|\mathcal{T}|^3 \mathrm{Card}(\mathcal{T}))$.

3.4 Genetic algorithms

3.4.1 Standard model

Encoding Any pagination (valid or not) on n pages is encoded as a tuple $(k_1, ..., k_{|T|})$ where $k_j \in [1, n]$ is the index of the page containing the tile t_j.

Example 4. The four paginations of Fig. 2 would be respectively encoded as $(1, 2, 3, 4)$, $(1, 2, 2, 3)$, $(1, 1, 2, 2)$ and $(1, 1, 1, 2)$.

Due to its overly broad encoding capabilities, Standard GA is *not* guaranteed to produce a valid pagination. Our fitness function is devised with this in mind, in such a way that an invalid chromosome would always cost more than a valid one. Thus, seeding the initial population with at least one valid individual will be enough to ensure success.

Evaluation Our aim is twofold. First and foremost, to penalize the invalid paginations; second, to reduce the volume of the last nonempty page. For this purpose, we will minimize the fitness function f defined as follows:

As soon as one page is overloaded (i.e., $\exists k \in P : \sum_{i \in A} x_k^i > C$), we count $|\mathcal{T}|C$ symbols (as if all possible pages were saturated), to which we also add every extra symbol:

$$f(\mathcal{P}) = |\mathcal{T}|C + \sum_{k \in P} \max\left(0, (\sum_{i \in A} x_k^i) - C\right) \quad \text{(6a)}$$

Otherwise, let us call k the index of the last nonempty page (i.e., such that $\forall k' > k, z_{k'} = 0$). Count $(k-1)C$ symbols (as if all nonempty pages but the last one were saturated), and add the number of symbols on page p_k:

$$f(\mathcal{P}) = (k-1)C + \sum_{i \in A} x_k^i \quad \text{(6b)}$$

Example 5. If all tiles of Fig. 2 were put on a single (invalid) page, by (6a), the fitness value would reach $4 \times 7 + 4 = 32$. By (6b), the paginations $\mathcal{P}_1$ to $\mathcal{P}_4$ have a fitness value of $3 \times 7 + 4 = 25$, $2 \times 7 + 4 = 18$, $1 \times 7 + 7 = 14$, and $1 \times 7 + 4 = 11$ (respectively).

Mutation It consists in transferring one randomly selected tile from one page to another.

Crossover The standard two-point crossover applies here without requiring any repair process.

3.4.2 Grouping model

In [6], Falkenauer shows that classic GAs are not suitable to the **grouping problems**, namely "optimization problems where the aim is to group members of a set into a small number of families, in order to optimize a cost function, while complying with some hard constraints". To take into account the structural properties of such problems, he introduces the so-called **grouping genetic algorithms** (GGA). His main idea is to encode each chromosome on a one gene for one group basis. The length of these chromosomes is thus variable: it depends

on the number of groups. Crucially, the belonging of several items to a same group is protected during crossovers: the good schemata are more likely to be transmitted to the next generations.

PAGINATION is clearly a grouping problem, moreover directly derived from BIN PACKING—one of the very problems Falkenauer chooses to illustrate his meta-heuristics. We thus will adapt, and sometimes directly apply his modelization.

Encoding A *valid* pagination on n pages is encoded as a tuple $(p_1, ..., p_n)$ where p_k is the set of the indexes of the tiles put on the k^{th} page.

Example 6. This time, the paginations of Fig. 2 would be encoded as $(\{1\},\{2\},\{3\},\{4\})$, $(\{1\},\{2,3\},\{4\})$, $(\{1,2\},\{3,4\})$ and $(\{1,2,3\},\{4\})$ (respectively). Or, in Falkenauer's indirect, but sequential notation: 1234:1234, 1223:123, 1122:12 and 1112:12, where the left part denotes, for each j^{th} tile, the index of its page; and the right part, the list of all pages.

Evaluation In our notation, the maximisation function of [6] for BIN PACKING would be expressed as $f_{\text{BP}}(\mathcal{P}) = \frac{1}{n}\sum_{k=1}^{n}(\frac{1}{C}\mathcal{V}(p_k))^d$. In other words, the average of volume rates raised to a certain **disparity** d, which sets the preference given to the bins' imbalance: thus, for a same number of bins and a same total loss, the greater the disparity, the greater the value of an unbalanced packing.

Although this formula still makes sense in the context of PAGINATION, we should not apply it as is. Indeed, whereas minimizing the loss amounts to minimizing the number of bins, Proposition 3 warns us this is actually untrue for the number of pages: in the associated counter-example, with $d = 2$ (the empirical value proposed by Falkenauer), the optimal pagination would be evaluated to $((3/4)^2 + (3/4)^2)/2 = 0.5625$, and the suboptimal one to $((4/4)^2 + (4/4)^2 + (4/4)^2)/3 = 1$.

Instead of privileging the high volume pages, we will privilege the high multiplicity ones (i.e., replace $\mathcal{V}(p_k)$ by $\text{Card}(p_k)$). Proposition 4 guarantees that the higher the average page multiplicity, the better the overall pagination.

One detail remains to be settled: ensure that the quantity raised to the power d never exceeds 1. Here, in the same way that $\mathcal{V}(p_k)$ is bounded by C, $\text{Card}(p_k)$ is bounded by $\text{Card}(\mathcal{T})$. Although a tighter bound can be, and has been implemented (namely, the sum of the multiplicities of the C most common symbols), to make it short we will halt on the following fitness function for PAGINATION:

$$f(\mathcal{P}) = \frac{1}{n}\sum_{k=1}^{n}\Big(\frac{\text{Card}(p_k)}{\text{Card}(\mathcal{T})}\Big)^d \tag{7}$$

Example 7. In Fig. 2, $\text{Card}(\mathcal{T}) = 5+3+3+4 = 15$. With $d = 2$, the four paginations are respectively evaluated as follows:

$$\begin{aligned}\left((5/15)^2 + (3/15)^2 + (3/15)^2 + (4/15)^2\right)/4 &\simeq 0.07\\ \left((5/15)^2 + (6/15)^2 + (4/15)^2\right)/3 &\simeq 0.11\\ \left((8/15)^2 + (7/15)^2\right)/2 &\simeq 0.25\\ \left((11/15)^2 + (4/15)^2\right)/2 &\simeq 0.30\end{aligned}$$

As one can see, $\mathcal{P}_4$ (the most unbalanced pagination) scores better than $\mathcal{P}_3$. The difference would increase with disparity d.

Mutation The mutation operator of [6] consists in emptying at random a few bins, shuffling their items, and then inserting them back by FIRST FIT. We follow the exact same procedure, but without the suggested improvements: at least three reinserted bins, of whom the emptiest one.

Crossover The two parents (possibly of different length) are first sliced into three segments: (a_1, a_2, a_3) et (b_1, b_2, b_3). The tiles of b_2 are withdrawn from a_1 and a_3: $a'_1 = a_1 \setminus b_2$ and $a'_3 = a_3 \setminus b_2$. To construct the first offspring, we concatenate (a'_1, b_2, a'_3), sort the missing tiles in decreasing order, and insert then back by FIRST FIT. The second offspring is calculated in the same manner (just exchange the roles of a and b).

3.5 Post-treatment by decantation

We introduce here a quadratic algorithm which, in an attempt to reduce the number of pages, will be systematically applied to the results produced by all our heuristics but FIRST FIT. Its three steps consist in settling at the beginning of the pagination as much pages, components and tiles as possible. First, we must specify what is a component:

Definition 3. Two tiles are **connected** if and only if they share at least one symbol or are connected to

another tile. The **(connected) components** are the classes of the associated equivalence relation.

Example 8. In Fig. 2, the components of the instance are {abcde def efg} and {hijk}.

Definition 4. A valid pagination is said to be **decanted** on the pages (resp., components, tiles) if and only if no page contents (resp., component, tile) can be moved to a page of lesser index without making the pagination invalid.

Example 9. In Fig. 2, $\mathcal{P}_3$ is decanted on the pages, but not on the components or the tiles. $\mathcal{P}_4$ is a fully decanted pagination (on the pages, the components and the tiles).

Obviously, a pagination decanted on the tiles is decanted on the components; and a pagination decanted on the components is decanted on the pages. To avoid any unnecessary repetition, the corresponding operations must then be carried out in the reverse direction. Moreover, the best decantation of a given pagination $\mathcal{P}$ is attained by decanting $\mathcal{P}$ successively on the pages, the components, and then the tiles.

Example 10. Let ({123}, {14 567}, {189}) be a pagination in 3 pages with $C = 5$. Its decantation on the components, ({123 14}, {567}, {189}), does not decrease the number of pages, as opposed to its decantation on the pages: ({123 189}, {14 567}). For an example on components/tiles, substitute 567 with 1567 in the instance.

Our decantation algorithm is thus implemented as a sequence of three First Fit procedures of decreasing granularity. In our tests, for any heuristics except First Fit itself, it was not unusual to gain one page, and even two, on the final pagination.

4 Experimental results

Supplementary material In order to empower the interested reader to reproduce our analysis and conduct her own investigation, we provide at `https://github.com/pagination-problem/1` a ~60 MB Git repository including: the whole set of our random instances (`gauss`); a companion Jupyter Notebook (`analysis.ipynb`) which generates every plot and numerical result mentioned or alluded in the present section; some instructions for using this notebook interactively (`README.md`).

4.1 Generating a test set

Our instance generator takes as input a capacity C, a number $|\mathcal{A}|$ of symbols and a number $|\mathcal{T}|$ of tiles. First, an integer k is drawn from a normal distribution (whose mean and standard deviation depend on C and a uniform random factor). If $1 < k < |\mathcal{A}|$ (Rules 2 and 3), a candidate tile is made up from a uniform random sample of k distinct symbols. It is added to the accepted set if and only if Rules 1 and 4 are still satisfied. The process is repeated until $|\mathcal{T}|$ tiles are obtained. To be accepted, the resulting instance must finally satisfy Rules 5, 6 and 7.

This algorithm has been called repeatedly with C varying from 15 to 50 in steps of 5, $|\mathcal{A}|$ varying from $C+5$ to 100 in steps of 5 (for lesser values, by Rule 8, all symbols would fit in a single page) and $|\mathcal{T}|$ varying from 20 to 100 in steps of 5. Although in some rare cases, numerous passes were required before halting on a result, it proved to be robust enough to produce, for each distinct combination of its parameters, six valid instances (for a total of 10,986 instances).

4.2 Measuring the difficulty of a given instance

What is a *difficult* instance of Pagination? Our proposed answer takes advantage of the highly experimental nature of the present approach. Indeed, we have not only generated several thousands of instances, but also submitted them to no less than six completely different solvers: one ILP, two genetic algorithms, two greedy algorithms, and one specialized heuristics (not counting its sorted variant), Overload-and-Remove. When these various methods all produce roughly the same number of pages, one can conclude that the instance was an easy one; conversely, when the pagination size differs greatly among methods, it clearly presents some particular challenges.

The dispersion of the pagination sizes can be measured in several ways: range (i.e., difference between minimum and maximum), standard deviation, median absolute deviation... Although, statistically speaking, the latter is the most robust, the former proved to be slightly more suited to our problem, where outliers are arguably no accidents, but rather evidences of some significant structural features. Hence:

Conjecture 1. *The **difficulty** of a given instance can be approximated by the difference between the maximal and the minimal number of pages in the paginations calculated by the various solvers.*

There are some caveats. First, this measure of difficulty is intrinsically correlated (Pearson's $r = 0.777$ [14]) to the size of the best pagination (e.g., if the best algorithm produces 2 pages, it is unlikely that any of its competitors will produce 20 pages). This is by design. Intuitively, a "large" random instance is more difficult to paginate than a "small" one. Therefore, normalizing the difference by dividing it by the best size would be counterproductive. Second caveat, this measure only makes sense for random instances. We certainly could devise a particular instance which would be easy for a special tailored algorithm, but difficult for our general-purpose solvers. Finally, this measure depends on our set of algorithms. Adding another one may paradoxically increase the measured difficulty of some instances. Of course, the more algorithms would be tested, the less the prevalence of this effect.

4.3 Predicting the difficulty of a given instance

PAGINATION can be seen as an almost continuous extension of BIN PACKING: being given a pure BIN PACKING instance (i.e., no tile has shared symbols), we may gradually increase its intricacy by transforming it into a PAGINATION instance as convoluted as desired (i.e., many tiles share many symbols). Therefore, we can expect that:

Conjecture 2. *The difficulty of a given random instance is strongly correlated to the density of its shared symbols, or **average multiplicity**.*

Before we go any further, it is important to be aware that the average multiplicities in our test set are far from being evenly distributed (Fig. 4): for example, there are 1119 instances whose multiplicity lies between 4 and 5, but only 107 between 23 and 24, and 10 between 53 and 54. Overall, more than half of them concentrate between multiplicities 2 and 9. Thus, any observation made on the higher multiplicities (and the smaller ones) must be approached with great caution. To take this into account, we carry out our analysis on a moving window of equally-sized subsets of instances sorted by increasing average multiplicity: the disappearance of some endpoints is greatly outweighed by the gain of a constant confidence level on the remaining data. And indeed, on our extensive test set, the average multiplicity of a given random instance appears to be an excellent predictor ($r = 0.986$) of its difficulty (Fig. 5).

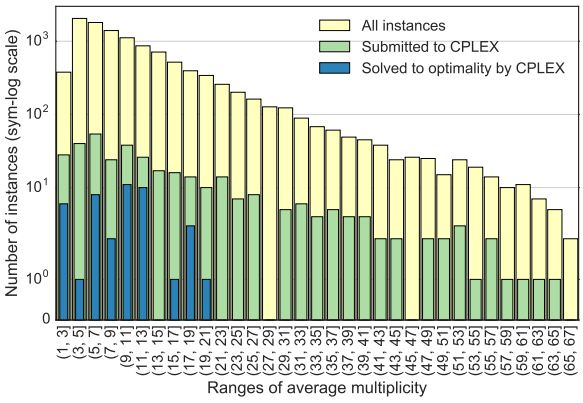

Figure 4: *Number of instances by average multiplicity (sym-log scale).*

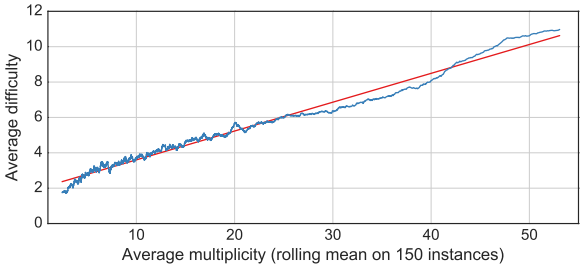

Figure 5: *Average difficulty by average multiplicity. A multiplicity of 10 indicates that, in the corresponding instances, a given symbol is shared by an average of 10 tiles. For these instances, the range of pagination sizes produced by our different solvers (the so-called difficulty) is almost 4.*

4.4 Discussion

4.4.1 Behavior of the integer program

As seen in Fig. 4, only a limited subset of our instances (342 of 10,986) have been submitted to

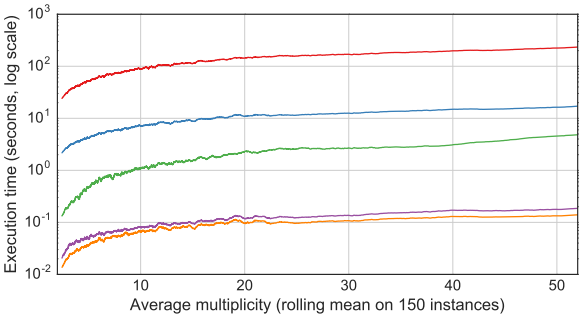

Figure 6: *Performance of the main heuristics (log scale). From top (slowest) to bottom (fastest):* Grouping GA, Standard GA, Overload-and-Remove, Best Fusion, First Fit.

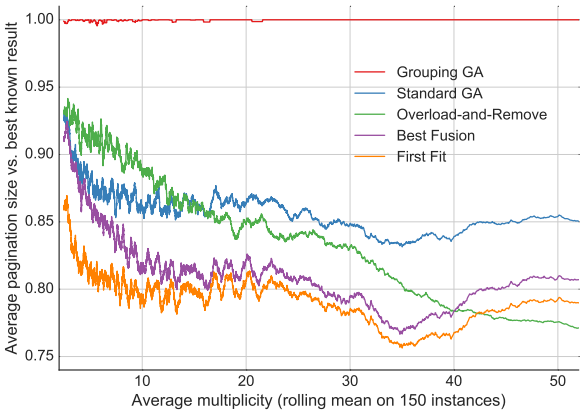

Figure 7: *Relative quality of the five main heuristics. The outcomes are plotted at* $y = \frac{\text{best size}}{\text{size}}$, *with* $y = 1$ *corresponding to the best known solution (which is either the optimal or the best feasible solution found by CPLEX, or the smallest approximation calculated for the given instance).*

CPLEX. This was for practical reasons: despite a powerful testing environment (Linux Ubuntu 12.04 on Intel Core i5-3570K with 4 cores of 3.4 GHz and 4 GB of RAM), CPLEX turned out to need a generous time-limit of one hour to be able to solve to optimality a mere 12.6 % of this subset (i.e., 43 instances). The ratio dropped to 3.8 % when the average multiplicity reached 13; above 20, no more success was recorded. Thus, this ILP quickly becomes irrelevant as the multiplicity increases, i.e., as Pagination starts to distinguish itself from Bin Packing.

Until we could find strong valid inequalities to improve it, our experimentations suggest that the heuristic approach constitutes a better alternative.

4.4.2 Comparison of the heuristic methods

All of our heuristics have been implemented in Python 2.7, and tested under Mac OS X 10.10 on Intel Core i5-3667U with 2 cores[5] of 1.7 GHz and 4 GB RAM. The average execution time ranges from less than 0.1 seconds for the greedy algorithms, 1 second for Overload-and-Remove, 7 seconds for Standard GA, through 90 seconds for Grouping GA. The two GAs were called with the same parameters: 80 individuals, 50 generations, crossover rate of 0.90, mutation rate of 0.01. Their initial population was constituted of valid paginations obtained by applying First Fit to random permutations of the tile set. Figure 6 shows how the various algorithms scale as multiplicity grows: performance-wise at least, all remain practical on our most difficult instances.

Figure 7 compares the results of these various heuristics. The irregularities of the top line also somehow give indirect insight into the rare achievements of our ILP. General observations and recommendations that can be derived from the underlying data are as follows.

Standard GA can definitely be ruled out As expected from Section 3.4.2, Standard GA was consistently surpassed by the more sensible Grouping GA: no more than 4 exceptions (0.036 %) occurred. Rather suspiciously, all of them involved a *minimal* instance, i.e., subject to a 2-page pagination. Further examination revealed that, in each case, this optimal pagination was already present in the initial random population constructed by First Fit; hence, its apparition was by no means indicative of some rare small-scale superiority of Standard GA: due to the fact that our implementation systematically transmits the best individual to the next generation, it was *preserved*, rather than *produced* by the evolution. This may also partially explain as pure chance the com-

[5]Since Python can only execute on a single core, we usually launched two processes in parallel.

paratively good performances in the lowest multiplicities; beyond that, the curve stabilizes quickly around an 85 % efficiency of Grouping GA. Finally, note that, up to an average multiplicity of 15, Standard GA is even outclassed by Overload-and-Remove, a six times faster heuristics.

Grouping GA produces the best overall results When quality is the top priority, Grouping GA is the way to go: it almost always (99.64 % of cases) guarantees equivalent or smaller paginations than any other heuristics. ILP did improved on it in 6 cases (1.75 % of the 342 selected instances): 2 with a better feasible solution, 4 with the optimal solution. But even if such good surprises would multiply on the whole set of instances, we must keep in mind that CPLEX was given one hour on four cores, against about 90 seconds to a pure Python implementation running on a single core (twice as slow): Grouping GA has yet to unleash its full potential.

The fastest non-genetic contender is among Best Fusion and Overload-and-Remove If speed matters, the choice depends on the average multiplicity of the instance: in most cases, Overload-and-Remove records quite honorable results. It is even the only non-genetic algorithm which proved occasionally (0.2 % of cases) able to beat Grouping GA. However, its quality regularly deteriorates (count up to 5 % for a 10 points increase in multiplicity). Around 35, somewhat surprisingly, the greedy algorithms get more and more competitive, with Best Fusion taking over at 40. Regardless of why this happens, a specialized heuristics for such deeply intricate instances would certainly be needed; in the meantime, Best Fusion represents the best tradeoff when average multiplicity meets or exceeds the 40 mark.

5 Conclusion

In this paper, we revisited an extension of Bin Packing devised by Sindelar et al. [11] for the virtual-machine colocation problem, by broadening its scope to an application-agnostic sharing model: in Pagination, two items can share unitary pieces of data, or symbols, in any non-hierarchical fashion. We showed that with such overlapping items, or tiles, the familiar tools and methods of Bin Packing may produce surprising results: for instance, while the family of Any Fit approximations have no more guaranteed performance, genetic algorithms still behave particularly well in the group-oriented encoding of Falkenauer [6]. We tested all these algorithms on a large set of random instances, along with an ILP, and some specialized greedy and non-greddy heuristics. The choice of the best one is not clear-cut, but depends on both time/quality requirements and the average multiplicity of the symbols. The latter measure was proposed as a predictor of the difficulty of a given instance, and correlated experimentally with the actual outcome of our various algorithms.

Obviously, this work did not aim to close the problem, but rather to open it to further research, by providing the required vocabulary, several theoretical tools, and an extensive benchmark. Indeed, numerous directions should be investigated: examples of these are worst-case analysis, proof of lower bounds, elaboration of efficient cuts, etc. To make Pagination more manageable, a promising approach restricts it to one single page: we have already subjected the so-called Fusion Knapsack problem to a preliminary study in [7].

References

[1] C. Berge. *Graphs and Hypergraphs*. Elsevier Science Ltd., Oxford, UK, UK, 1985.

[2] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In Dorit S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, chapter 2, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997.

[3] György Dósa and Jiří Sgall. First Fit bin packing: A tight analysis. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20, pages 538–549, Dagstuhl, Germany, 2013.

[4] Mitre C. Dourado, Fabio Protti, and Jayme L. Szwarcfiter. Complexity aspects of the Helly property: Graphs and hypergraphs. *The Electronic Journal of Combinatorics*, #DS17:1–53, 2009.

[5] John Duncan and Lawrence W. Scott. A branch-and-bound algorithm for pagination. *Operations Research*, 23(2):240–259, 1975.

[6] Emanuel Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 1186–1192, 1992.

[7] Aristide Grange, Imed Kacem, Karine Laurent, and Sébastien Martin. On the Knapsack Problem Under Merging Objects' Constraints. In *45th International Conference on Computers & Industrial Engineering 2015 (CIE45)*, volume 2, pages 1359–1366, Metz, France, 2015.

[8] David S. Johnson. *Near-Optimal Bin Packing Algorithms*. PhD thesis, Massachusetts Institute of Technology, Cambridge, 1973.

[9] George Karypis and Vipin Kumar. Multilevel k-way Hypergraph Partitioning. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, DAC '99, pages 343–348, New Orleans, Louisiana, USA, 1999. ACM.

[10] Michael Kaufmann, Marc Kreveld, and Bettina Speckmann. *Graph Drawing: 16th International Symposium, GD 2008, Heraklion, Crete, Greece, September 21-24, 2008. Revised Papers*, chapter Subdivision Drawings of Hypergraphs, pages 396–407. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[11] Michael Sindelar, Ramesh K. Sitaraman, and Prashant Shenoy. Sharing-aware algorithms for virtual machine colocation. In *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures - SPAA '11*, page 367, New York, USA, 2011. ACM Press.

[12] Emanuel Sperner. Ein satz über untermengen einer endlichen menge. *Mathematische Zeitschrift*, 27(1):544–548, 1928.

[13] Apostolos Syropoulos. Mathematics of Multisets. *Multiset Processing SE - 17*, 2235(August):347–358, 2001.

[14] Wikipedia. Pearson product-moment correlation coefficient — wikipedia, the free encyclopedia, 2016. [Online; accessed 28-April-2016].