UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2577

An *n*-Tet Graph Approach for Non-guillotine Packings of *n*-Dimensional Boxes into an *n*-Container

Lauro Lins
Sóstenes Lins
Reinaldo Morabito

Nº 48

NOTAS

Série Computação

São Carlos – SP
Jul./2000

Uma Abordagem em Grafo n-Tet para Empacotamentos Não-Guilhotinados de Caixas n-Dimensionais em um n-Contêiner*

Lauro Lins, Sóstenes Lins e Reinaldo Morabito†

Departamento de Informática, Universidade Federal de Pernambuco

Cidade Universitária, 50740-540 Recife, Brazil, e-mail: ldl@di.ufpe.br

Departamento de Matemática, Universidade Federal de Pernambuco

Cidade Universitária, 50740-540 Recife, PE, Brazil, e-mail: sostenes@dmat.ufpe.br

Departamento de Engenharia de Produção, Universidade Federal de São Carlos

Caixa Postal 676, 13565-905 São Carlos, SP, Brazil, e-mail: morabito@power.ufscar.br

July 13, 2000

Abstract

Neste artigo propomos um simples algoritmo uniforme recursivo para o problema de empacotar caixas n-dimensionais dentro de um n-contêiner. Estamos particularmente interessados no caso especial $n = 3$ em que as caixas podem ser empacotadas num dado subconjunto de seus seis possíveis posicionamentos. Nosso método estuda simetrias nos empacotamentos pelo uso de um conjunto ordenado de três grafos direcionados com as mesmas arestas (3-tet ou tríade) e estruturas menores induzidas do mesmo tipo denominadas *minors*. Com o método, aspectos de degeneração e simetria, que reduzem a enumeração implícita para tempos computacionais praticamente aceitáveis, tornam-se transparentes. Para ilustrar o desempenho do algoritmo, os resultados computacionais de exemplos 3-dimensionais gerados aleatoriamente são apresentados e comparados com resultados de uma abordagem camadas&mochila. O presente estudo tem aplicações reais para os problemas de carregamento de paletes e contêineres.

Palavras-chaves: *empacotamentos 3-dimensionais, carregamento de paletes e contêineres, otimização combinatória, teoria de grafos.*

---

*Artigo submetido em junho de 2000 para o European Journal of Operations Research (EJOR) — na edição especial em problemas de corte e empacotamento

†Orientador do Programa de Pós-Graduação em Ciências da Computação e Matemática Computacional. ICMC/USP

Uma Abordagem em Grafo n-Tet para Empacotamentos Não-Guilhotinados de Caixas n-Dimensionais em um n-Contêiner*

Lauro Lins, Sóstenes Lins e Reinaldo Morabito†

Departamento de Informática, Universidade Federal de Pernambuco

Cidade Universitária, 50740-540 Recife, Brazil, e-mail: ldl@di.ufpe.br

Departamento de Matemática, Universidade Federal de Pernambuco

Cidade Universitária, 50740-540 Recife, PE, Brazil, e-mail: sostenes@dmat.ufpe.br

Departamento de Engenharia de Produção, Universidade Federal de São Carlos

Caixa Postal 676, 13565-905 São Carlos, SP, Brazil, e-mail: morabito@power.ufscar.br

July 13, 2000

Abstract

Neste artigo propomos um simples algoritmo uniforme recursivo para o problema de empacotar caixas n-dimensionais dentro de um n-contêiner. Estamos particularmente interessados no caso especial $n = 3$ em que as caixas podem ser empacotadas num dado subconjunto de seus seis possíveis posicionamentos. Nosso método estuda simetrias nos empacotamentos pelo uso de um conjunto ordenado de três grafos direcionados com as mesmas arestas (3-tet ou tríade) e estruturas menores induzidas do mesmo tipo denominadas *minors*. Com o método, aspectos de degeneração e simetria, que reduzem a enumeração implícita para tempos computacionais praticamente aceitáveis, tornam-se transparentes. Para ilustrar o desempenho do algoritmo, os resultados computacionais de exemplos 3-dimensionais gerados aleatoriamente são apresentados e comparados com resultados de uma abordagem camadas&mochila. O presente estudo tem aplicações reais para os problemas de carregamento de paletes e contêineres.

Palavras-chaves: *empacotamentos 3-dimensionais, carregamento de paletes e contêineres, otimização combinatória, teoria de grafos.*

*Artigo submetido em junho de 2000 para o European Journal of Operations Research (EJOR) — na edição especial em problemas de corte e empacotamento

†Orientador do Programa de Pós-Graduação em Ciências da Computação e Matemática Computacional, ICMC/USP

An n-Tet Graph Approach for Non-guillotine Packings of n-Dimensional Boxes into an n-Container*

Lauro Lins, Sóstenes Lins and Reinaldo Morabito†

Departamento de Informática, Universidade Federal de Pernambuco

Cidade Universitária, 50740-540 Recife, Brazil, e-mail: ldl@di.ufpe.br

Departamento de Matemática, Universidade Federal de Pernambuco

Cidade Universitária, 50740-540 Recife, PE, Brazil, e-mail: sostenes@dmat.ufpe.br

Departamento de Engenharia de Produção, Universidade Federal de São Carlos

Caixa Postal 676, 13565-905 São Carlos, SP, Brazil, e-mail: morabito@power.ufscar.br

July 13, 2000

Abstract

In this paper we propose a simple recursive uniform algorithm for the problem of packing n-dimensional boxes into an n-container. We are particularly concerned about the special case $n = 3$ where the boxes can be packed in a given subset of their six possible positionings. Our method studies symmetries in the packings by the use of an ordered set of three directed graphs with the same edges (a 3-*tet or triad*) and induced smaller structures of the same kind named *minors*. With the method, degeneracy and symmetry issues, which curtail the implicit enumeration to practically acceptable running times, become transparent. In order to illustrate the performance of the algorithm, computational results from solving randomly generated 3-D examples are presented and compared with the ones of a layers&knapsack approach. The present study has real world applications for the problems of pallet and container loading.

Keywords: *3-dimensional packing, pallet/container loading, combinatorial optimization, graph theory.*

1 Introduction

In this study we present a simple recursive algorithm for the problem of packing n-dimensional boxes (or simply boxes) into a larger n-box, henceforth referred to as a *container*. The algorithm

*Paper submitted on june 27, 2000 to the European Journal of Operations Research (EJOR) — in the special issue on cutting and packing problems

†Orientador do Programa de Pós-Graduação em Ciências da Computação e Matemática Computacional, ICMC/USP

is based on the representation of a feasible packing as a depth assignment in an object called n-tet, which is a sequence of n directed graphs. To each such an assignment in the n-tet there corresponds a partition of the container into specific subcontainers. Each of these subcontainers can be recursively partitioned into further ones, according to the specifications in the n-tet. A subcontainer which is not partitioned is homogeneously loaded with the boxes. The algorithm performs a search of all feasible depth assignments of the n-tets involved to find the most valuable one, that is, the one which induces a maximum occupancy of the volume of the container by packed boxes. To organize and simplify the search, the procedure explores recursion techniques, which extends to higher dimensions the algorithm of Morabito and Morales [29] proposed for the $n = 2$ case.

Triads are 3-tets and were introduced in Lins et al. [23], where it was shown how a method based on this representation can produce better 3-dimensional non-guillotine packings than methods using layers&knapsack approaches. In the present paper we formalize a general n-packing algorithm considering degenerative, reflexive and rotational symmetry issues which are essential for the performance of the method. The algorithm can be easily adapted to generate n-dimensional (n-D) guillotine packings. The *3-D packing problem* (3PP) has various practical applications, as in the loading of products (packed into boxes) on pallets, or inside containers or safe-trucks. Such problems appear in the logistic activities of transporting and storing goods or supplies and, depending on the scale of the supply/distribution chain, a small increase in the volume of products loaded on a pallet, or inside a container, can result in substantial savings.

Several studies treating 2-D and 3-D packing problems have been reported in the literature in this last decade, such as Bischoff and Marriott [6], Haessler and Talbot [20], George [19], K. Dowsland [14], Abdou and Yang [1], Morabito and Arenales [28], Chen et al. [11], Bischoff and Ratcliff [7], W. Dowsland [16], Nelissen [31], Scheithauer and Terno [33], Herbert and Dowsland [21], Miyazawa and Wakabayashi [27], Liu and Hsiao [24], Bhattacharya et al. [9], Bortfeldt and Gehring [10], Chien and Wu [12], Scheithauer and Sommerweiss [32], and Morabito et al. [30]. Other references can be found in the surveys and special issues in Dowsland and Dowsland [15], Dyckhoff and Finke [17], Sweeney and Paternoster [35], Balasubramanian [4], Martello [25] [26], Bischoff and Waescher [8], Dyckhoff et al. [18], Arenales et al. [3], and the eletronic databases of the Special Interest Group on Cutting and Packing (SICUP [34]).

In the present study we report some computational experience in the 3-D case in which the boxes are of equal size. However, the algorithm is given in full generality and our implementation can be applied to the more general case where the boxes are different and of higher dimensions. Moreover, the algorithm can be extended to the case where there is a limited number of boxes of each type, or the case where there is a great number of boxes to be packed in different containers, in which case it plays the role of a column generator to a simplex approach. These matters will be treated elsewhere. Here, we also assumed that the boxes, available in large quantity, are to be orthogonally arranged inside the container, that is, with their faces parallel to those of the container. This assumption provides, in general, six nonequivalent ways to position a box. The subset of the permitted positionings is an input to the algorithm.

3PP is an NP-hard problem and can be formulated as an integer linear programming, for example, extending the (0,1)-model in Beasley [5], originally proposed for the 2-D case, or applying the (0,1)-model in Tsai et al. [36] based on disjunctive restrictions. Exact methods of branch-and-bound type exploring bounds coming from the surrogate and Lagrangean relaxation can be defined

following [5] and [36]. Alternatively, Dowsland [13] presented an interesting approach for the 2-D packing, which can also be extended to treat 3PP. Basically, the approach consists of finding the maximum stable set of a particular finite graph where the nodes correspond to the possible positioning of the boxes inside the container. Two nodes are the ends of an edge if the box positions which they represent overlap. However, because of the size of practical 3PPs, both the (0,1)-models mentioned above and Dowsland's graph approach are generally too large to be computationally treated. This way, most methods found in the literature are heuristics, such as those listed in the previous paragraphs. The recursive procedure treated here based on the n-tet representation is also heuristic. However, it deterministically finds the optimum over a well-defined class of patterns, which will contain an optimum in the vast majority of the cases.

The paper is organized as follows: In the next section we introduce the concept of n-tets and characterize their close connections with container partitions. In section 3 we use the language of n-tets to present a recursive algorithm to generate n-D non-guillotine packing patterns. In section 4 we present an illustrative 3-D example to justify the approach. In section 5 we treat extensively degeneracy and reflexive/rotational symmetry issues which substantially curtail the implicit enumeration. In section 6, in order to illustrate the performance of the algorithm, we solve randomly generated 3-D examples and compare the solutions with the 2-D, the guillotine and the layers&knapsack approaches. We finish in section 7 with concluding remarks and some perspectives for future research.

2 n-Tets, δ-tets and partitions of n-containers

n-Tets. Let $G = (G_1, G_2, \ldots, G_n)$ $(n \geq 2)$ be a sequence of n directed graphs (*digraphs*) with $v_1, v_2, \ldots, v_n$ vertices, respectively. Every G_j has the same number e of edges and is called a *profile* of G. The vertices of G_j are labeled with the elements of $\{0, 1, 2, \ldots, v_j - 1\}$ in such a way, as permitted by their acyclic conditions, that all their edges go from lower end to higher end labeled vertices. Let $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ be given where β_j is a $1-1$ correspondence from the edge set of G_j onto the set of $E = \{1, 2, \ldots, e\}$. Set E is the *edge-label* set of G. These bijections permit us to identify i_j $(i \in E, j \in N = \{1, 2, \ldots, n\})$ as the i-th labeled edge in the j-th profile G_j. Let $t(i_j)$ denotes the label of the tail of i_j and $h(i_j)$ denotes the label of its head. The *length* of i_j is the positive integer $\lambda(i_j) = h(i_j) - t(i_j)$. G has the *disjointness property* if for each distinct pair $i, k \in E$ there is $j \in N$ such that the real open intervals $(t(i_j), h(i_j))$ and $(t(k_j), h(k_j))$ are disjoint. Given $i \in E$, the *volume* of i is $V(i) = \prod(\lambda(i_j) | j \in N)$. The *volume of* G itself is $V(G) = \prod((v_j - 1) \mid j \in N)$, that is, the product of one less the number of vertices of the profiles. G has the *volume additivity property* if $V(G) = \sum(V(i) | i \in E)$. G is an *n-tet* if it satisfies both the disjointness and volume additivity properties.

δ-Tets. A *depth assignment* δ for an n-tet G is a sequence $\delta = (\delta_1, \delta_2, \ldots, \delta_n)$ where each δ_j is a function of the vertex set of G_j into the non-negative integers such that $\delta_j(0) = 0$ and $\delta_j(k) \leq \delta_j(k+1)$ for $k \in \{0, 1, \ldots, v_j - 2\}$. Thus we may think of δ_j as a non-decreasing 0-starting sequence $(0 = \delta_{j,0}, \delta_{j,1}, \ldots, \delta_{j,v_j-1})$, where $\delta_j(k) = \delta_{j,k}$. A depth assignment δ is *non-degenerated* if all the sequences δ_j are strictly increasing. Otherwise it is called *degenerated*. A *δ-tet* is an n-tet G together with a depth assignment δ for it, and is represented by G^δ. The δ-length of edge i_j in G_j is $\lambda_\delta(i_j) = \delta_j(h(i_j)) - \delta_j(t(i_j))$. For $i \in E$, the *δ-volume* of i is $V_\delta(i) = \prod(\lambda_\delta(i_j) \mid j \in N)$. The *$\delta$-volume of* G is $V(G) = \prod(\delta_j(v_j - 1) \mid j \in N)$. Clearly, given any G^δ, it satisfies the *δ-volume*

additivity property: $V_\delta(G) = \sum(V_\delta(i) \mid i \in E)$. The container c *induced by a* G^δ is the container whose dimensions $(c_1, c_2, \ldots, c_n)$ are obtained as the last entries of the sequences δ_j's, namely, $(\delta_1(v_1 - 1), \delta_2(v_2 - 1), \ldots, \delta_n(v_n - 1))$. Given a pair (G, c) the set of δ's such that G^δ induces c is named the (G, c)-*admissible set of depth assignments* and is denoted by $Adm(G, c)$.

Partitions of containers. Consider a partition of an n-container c into e n-subcontainers $s_1, s_2, \ldots, s_e$. We position a coordinate system $\mathcal{X}$ in such a way that each side of the container is parallel to some axis of $\mathcal{X}$. We also assume that all the vertices of the subcontainers have integer coordinates in $\mathcal{X}$. The *starting* vertex of s_i is its vertice with lexicographically smaller coordinates. All the information to specify s_i inside c is given by the lengths of its sides $(s_{i1}, s_{i2}, \ldots, s_{in})$ and coordinates of its starting vertex $(x_1, x_2, \ldots, x_n)$. These observations permit us to think of a δ-tet G^δ as a partition of its induced container c. Indeed, each $i \in E$ corresponds to a subcontainer $s_i = s_i(G^\delta)$ of c whose length of sides are $s_{i1} = \lambda_\delta(i_1), s_{i2} = \lambda_\delta(i_2), \ldots, s_{in} = \lambda_\delta(i_n)$ and whose coordinates of starting vertex are $x_1 = \delta_1(t(i_1)), x_2 = \delta_2(t(i_2)), \ldots, x_n = \delta_n(t(i_n))$. It follows from the disjointeness and δ-volume additivity properties of G that the subcontainers $s_1, s_2, \ldots, s_e$ form a partition of c, the container induced by G^δ. We denote the correspondence by ψ so that $\psi(G^\delta)$ is c with a well defined partition.

A *duet* is a 2-tet. As an example of the correspondence ψ consider Fig. 1. The δ-duet $T^2((0 = x_0, x_1, x_2, x_3 = p), (0 = y_0, y_1, y_2, y_3 = q))$ is mapped by an inversive map ψ, which we defined above, into a 5-fold partition of its induced (p, q)-rectangle. The specific partition of the (p, q)-rectangle at the right of Fig. 1 into those five smaller rectangles is the image, under ψ, of the δ-duet at the left of this figure.

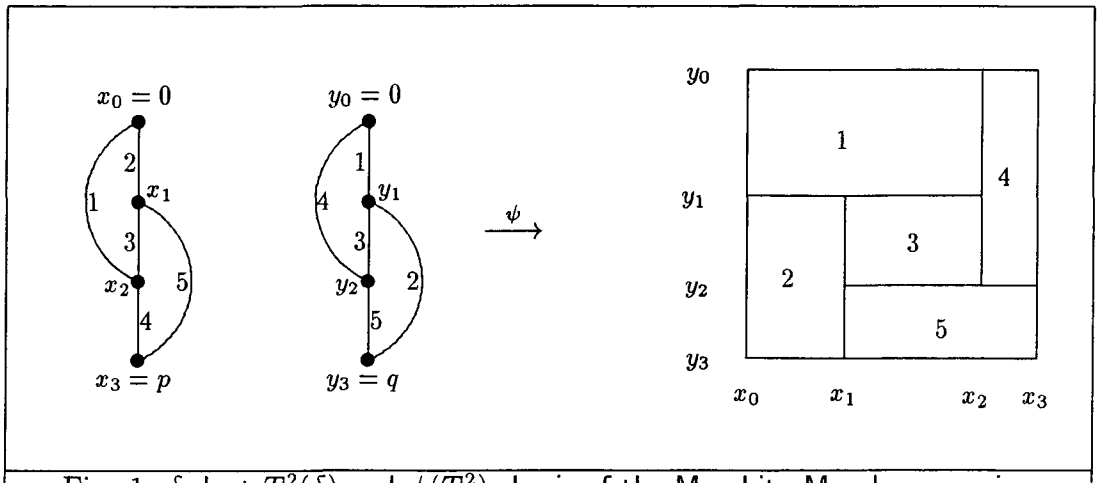


Fig. 1: δ-duet $T^2(\delta)$ and $\psi(T^2)$: basis of the Morabito-Morales recursion

Note that the dimensions of the i-th subrectangle equals to the lenghts of the occurrences of the i edges in the profiles. For instance, subrectangle 3 has dimensions $s_{31} = \delta_1(h(3_1)) - \delta_1(t(3_1)) = x_2 - x_1$, $s_{32} = \delta_2(h(3_2)) - \delta_2(t(3_2)) = y_2 - y_1$. Thus, the δ-duet $T^2(\delta)$ in Fig. 1 corresponds, under ψ, to the canonical partition of the Morabito and Morales [29] recursion.

The sequence T^n of the minimal trissecting packing patterns. What is important about T^2 is that is the simplest 2-D non-guillotine packing pattern. In dimension 2 the simplest non-guillotine pattern corresponds to the *minimal trissecting pattern*, that is, it has a minimum number of edge-labels and two internal vertices at each profile. By means of recursion in the subrectangles,

it can produce complicated first-order non-guillotine patterns (see Fig. 6), attaining the optimum in the vast majority of cases (Morabito and Morales [29]).

One key idea in this work is to define the n-D's counterparts of this partition. In fact, we do it explicitly only for $n \in \{3,4\}$. We work mainly with the ψ^{-1}-associated 3- and 4-tets which are denoted T^3 and T^4. A *triad* is a 3-tet. To define the triad T^3 we form the *9-fold canonical partition* displayed in Fig. 2. Starting with subcontainers $1,4,5,7$ and 8 in the left-hand picture, we add subcontainers 3 and 9 as shown in the middle and, finally, add subcontainers 2 and 6 to obtain the complete pattern on the right. Note that T^3, the *minimal trissecting 3-D packing pattern* extends T^2 to dimension 3. T^3 corresponds, under ψ^{-1}, to the 9-fold *canonical partition of a* $(3 \times 3 \times 3)$*-cube* displayed in Fig. 2. The depth assignment is only implicitly present in the figure. Let it be $\delta = (0 = x_0, x_1, x_2, x_3 = p), (0 = y_0, y_1, y_2, y_3 = q), (0 = z_0, z_1, z_2, z_3 = r)$. The corresponding δ-triad is denoted $T^3(\delta)$ and is shown in Fig. 3.

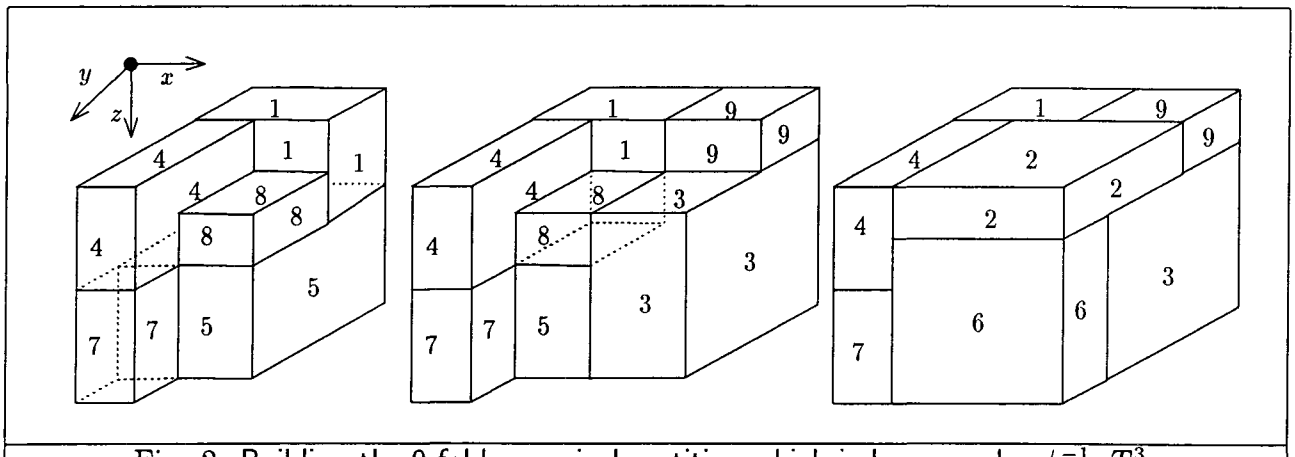


Fig. 2: Building the 9-fold canonical partition which induces, under ψ^{-1}, T^3

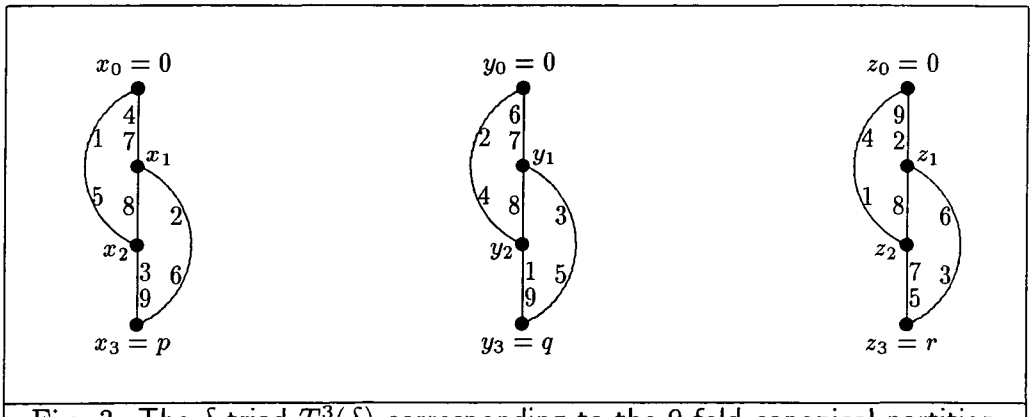


Fig. 3: The δ-triad $T^3(\delta)$ corresponding to the 9-fold canonical partition

We observe that $\psi(T^3(\delta))$ is not a minimal non-guillotine pattern: consider triad T_{38} of appendix A. For any container c and a non-degenerated $\delta \in Adm(T_{38}, c)$, $\psi(T_{38}^{\delta})$ is a non-guillotine packing pattern having only five subcontainers. Concerning packing properties in dimensions 3 and higher, the concept of non-guillotine does not coincide and is less useful than the concept of trissecting. Therefore, T^3 plays a basic role in producing better packing patterns. T_{38}, the simplest non-guillotine packing pattern, is weaker in this respect. So is T_{57} (appendix A), the *minimal bissecting* packing pattern, which generates the 3-D guillotine packing patterns.

A *quartet* is a 4-tet. Fig. 4 depicts the simplest trissecting (a 17-fold) partition of a 4-container by means of a specific quartet, T^4. In the figure, the labels of the vertices of the profiles are 0,1,2,3 from top to bottom. The reader can check that T^4 satisfies the disjointeness property. For instance, subcontainers s_1 and s_7 are disjoint because $(t(1_3), h(1_3)) \bigcap (t(7_3), h(7_3)) = (2,3) \bigcap (0,2) = \emptyset$ (Fig. 4). So, the third profile is a *witness* of the fact that the intersection of the interiors of s_1 and s_7 are disjoint. There is a similar witness for each one of the other 135 pairs of elements in $\{1, 2, \ldots, 17\}$. The volume additivity property is easier to verify. The 17 4-subcontainers labelled $s_1, s_2, \ldots, s_{17}$ have 4-volumes: $2, 8, 2, 2, 8, 8, 8, 2, 1, 2, 8, 8, 8, 2, 2, 8, 2$, respectively. The sum of these volumes is equal to the volume of T^4 which is $3^4 = 81$.

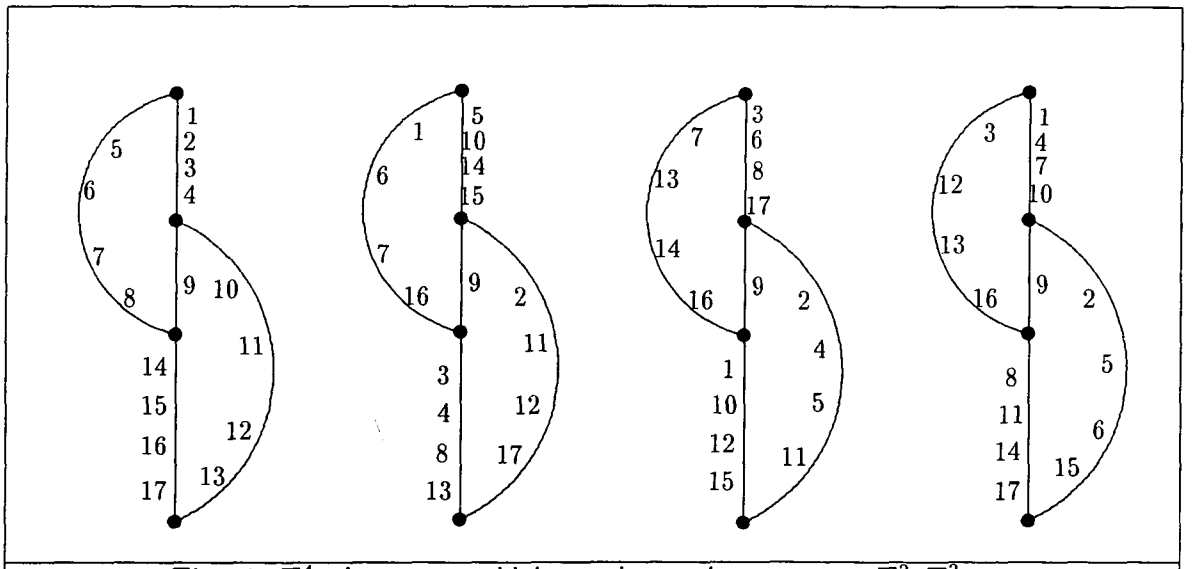


Fig. 4: T^4: A quartet which continues the sequence $T^2, T^3, \ldots$

In general, it is possible to define $T^n, n > 4$, as a trissecting n-tet having an edge label E with $|E| = 2^n + 1$ elements. We note, however, that the exact description of these n-tets becomes rather cumbersome, and is not given explicitly in this work. What follows, we no longer make a distinction between a δ-tet and its associated container partition, that is, we think of map ψ as the identity.

3 Tet based recursive algorithm

The class $G_k(c)$ of container partitions. Throughout this section c is a fixed container with non-negative integers side lengths $c_1, c_2, \ldots, c_n$ and G is an n-tet. Let $G(c) = \{G^\delta \mid \delta \in Adm(G, c))\}$. Therefore, by identifying the sets of δ-tets and partitions of containers, the set $G(c)$ can be thought of as a finite set of partitions of c. Let $G_k(c)$ be a set of partitions induced by (G, c) and defined inductively as follows: $G_0(c)$ is the set of just one, the trivial partition of c, namely, c itself not partitioned. $G_1(c)$ is $G(c)$. For $k \geq 2$, a member of $G_k(c)$ is characterized as a $\delta \in Adm(G, c)$, which defines a sequence of subcontainers $s_i(G^\delta), i = 1, 2, \ldots, e$, forming a partition of c; each s_i in its turn is further partitioned as specified by member of $G_{k-1}(s_i)$, which is already defined by induction. Note that a member of $G_k(c)$ defines a partition of c which potentially has e^k subcontainers. In practice, because of degeneracy, most of them have null volume.

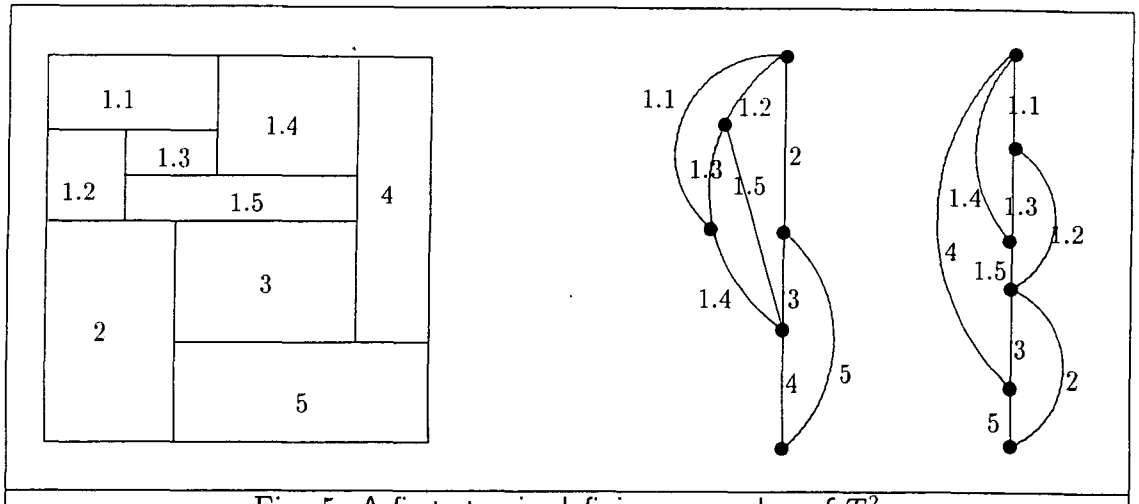


Fig. 5: A first step in defining a member of T_2^2

We show in Fig. 5 the first step in defining a member of $T_2^2(c = (p, q))$: edge 1_1 is replaced by a copy of the first profile of T^2 and edge 1_2 is replaced by a copy of the second profile of T^2. These replacements correspond to subdividing the subcontainer s_1 according to a T^2 pattern. The above notation is convenient for making uniform various classes of packing patterns well known in the literature. The set of *first order level k non-guillotine packing patterns* of a rectangle r (see Arenales and Morabito [2]) is simply the set $T_k^2(r)$. We extend this definition to the general dimension n by replacing $T_k^2(c)$ with $T_k^n(c)$. It is worth noting that the straightforward generalization to 3-D of the class of guillotine packing patterns of c at level k can be based on the triad T_{57} presented in the appendix A and is precisely the set $(T_{57})_k(c)$.

The functions ν_B and δ_B^*. Let B be a sequence of h box types $b_1, b_2, \ldots, b_h$. The dimensions of n-box type b_i are $(b_{i1}, b_{i2}, \ldots, b_{in})$. The order of these dimensions is important: if a 3-box b can be positioned in four of the six possibilities, b contributes four triples to B. We only work with loadings which are *orthogonal*, namely, each face of a loaded box is parallel to some face of the container. A *homogeneous loading of c* is an orthogonal loading of c by copies of boxes of only one type, say b_i. The maximum number of boxes of type b_i which can be homogeneously loaded into c is $NH(c, b_i) = \lfloor c_1/b_{i1} \rfloor \lfloor c_2/b_{i2} \rfloor \ldots \lfloor c_n/b_{in} \rfloor$. The corresponding *volume occupancy* is $VO(c, b_i) = b_{i1} b_{i2} \ldots b_{in} NH(c, b_i)$. The *best homogeneous occupancy* of c is $BHO(c, B) = \max\{VO(c, b_i) \mid i = 1, 2, \ldots, h\}$. Given a G^δ corresponding to a partition of an n-container c by means of subcontainers $s_1(G^\delta), s_2(G^\delta), \ldots, s_p(G^\delta)$, and a set B of h box types $b_1, b_2, \ldots, b_h$, the *B-value of* G^δ is $\nu_B(G^\delta) = \sum_{i=1}^{p} BHO(s_i, B)$. Let $\Omega(c)$ be a set of δ-tets with each of its member inducing c. The *B-value* of $\Omega(c)$ is $\nu_B(\Omega(c)) = \max\{\nu_B(G^\delta) \mid G^\delta \in \Omega(c)\}$. In the case that $\Omega(c) = G_k(c)$ for some n-tet G with e edge-labels and non-negative integer k, we present an algorithm ρ, recursive on k, which finds both $\nu_B(\Omega)$ and a $J^\delta \in \Omega$ such that $\nu_B(\Omega) = \nu_B(J^\delta)$. Note that J may have up to e^k edge-labels. Let c' be a container whose dimensions do not exceed those of c, $c' \subseteq c$. Let k' be integer satisfying $0 \leq k' \leq k$. The algorithm is based in the following structural fact on the class of c'-partitions $G_{k'}(c')$: $\nu_B(G_k(c)) = \max\{\sum_{i=1}^{e} \nu_B(G_{k-1}(s_i(G^\delta))) \mid \delta \in Adm(G, c)\}$. Define $\delta_B^*(G_k(c))$ to be an arbitrary but fixed $\delta \in Adm(G, c)$, which attains the maximum in the above expression.

Tet-based recursive algorithm ρ. Let c' and k' be as above. Define the *discrete ordered sets* $X_j(B, c'_j), j \in N$ as follows: set $X_j(B, c'_j)$ is the ordered sequence of non-negative linear combinations of $b_{1j}, b_{2j}, \ldots, b_{hj}$ which do not exceed c'_j. The importance of these sets is that all entries of the admissible depth functions are in $X_j(B, c'_j)$. This is observed in Herz [22] and

substantially speeds up the procedure. Throughout algorithm ρ, ν' is a current lower bound for $\nu_B(G'_k(c'))$ attained by a current $\delta' \in Adm(G, c')$. When the partition of the container c' is the trivial partition, namely, the partition consisting of just one part c' itself, the corresponding δ' is the *empty depth assignment*, and we write $\delta' = \emptyset$. In the input of algorithm ρ (and its improvements ρ_1, ρ_2, ρ_3 of section 5), $B = \{b_1, b_2, \ldots, b_h\}$ is a set of h n-box types where $b_i = (b_{i1}, b_{i2}, \ldots, b_{in})$, G is an n-tet, $c = (c_1, c_2, \ldots, c_n)$ is an n-container, and k is a non-negative integer, the level of the recursion.

Recursive Algorithm ρ: Input (B, G, c, k). Output $(\delta^*_B(G_k(c)), \nu_B(G_k(c)))$.
Initialization: Form the sets $X_j(B, c_j)$, $j \in N$. Allocate adequate memory space for storing $\delta^*_B(G_{k'}(c'))$ and $\nu_B(G_{k'}(c'))$, where $0 \leq k' \leq k$ and $c' \subseteq c$. Main call: Call routine ρ with parameters (B, G, c, k). Exit.
Routine $\rho(B, G, c', k')$: returns the pair $(\delta^*_B(G_{k'}(c'), \nu_B(G_{k'}(c'))))$.
Step 0: If $k' = 0$ then $(\delta', \nu') = (\emptyset, BHO(c', B))$, and go to Step 3. Step 1: Make $\delta' = \emptyset$ and $\nu' = \nu_B(G_{k'-1}(c'))$. Step 2: For each $\delta \in Adm(G, c')$ whose entries of δ_j are in $X_j(B, c'_j)$ do Step 2.1: Determine the $e = \lvert E(G)\rvert$ subcontainers, $s_i(G^\delta), i \in E(G)$. Step 2.2: If $\nu'' = \sum_{i=1}^{e} \nu_B(G_{k'-1}(s_i(G^\delta))) > \nu'$, then $\delta' = \delta$, $\nu' = \nu''$. Step 3: Return the best found solution $\rho(B, G, c', k') = (\delta', \nu')$.

Note that in Steps 1 and 2.2 each first computation of the summands $\nu_B(G_{k'-1}(c'))$ and $\nu_B(G_{k'-1}(s_i(G^\delta)))$ are realized by recursive calls to $\rho(B, G, c', k')$ (ν_B is the second parameter of the output of ρ). The need to have the same specific summand appears at many distinct points of the algorithm, and so, we keep them in memory, as we first encounter them. Note that the δ-tet J^δ which realizes the partition of c in $G_k(c)$ with best B-value can be recovered from the output of the calls to $\rho(B, G, c', k')$ because each such an output yields the external partition of c', defined by $\delta^*_B(G_{k'}(c'))$. Therefore, J^δ is recovered via a tree like structure obtained from the recursive calls to ρ. In the next section we present an example of the use of algorithm ρ in the case $G = T^3$. In section 5, we provide a complete detailed analysis of the crucial role of degeneracy and reflexive/rotational symmetry in decreasing the number of duplicated choices in Step 2. The careful analysis justifies because the algorithm has straightforward practical applications and so, curtailing its running time, becomes important. Algorithm ρ, as it stands, was not implemented, because choosing degenerated depth assignments yields a lot of duplicated effort making the running time prohibitively high. The ones which were implemented are ρ_1, ρ_2, and ρ_3 explained in section 5. These algorithms are successive improvements of ρ.

4 An illustrative example

In this section we exemplify the ρ-algorithm in the case $n = 3$ where G is T^3. Let c be the 3-container $(61, 44, 50)$ to be loaded with identical boxes of size $(5, 6, 9)$. Observe that a volume based upper bound for the number of boxes is: $\lfloor 61 \times 44 \times 50/9 \times 6 \times 5 \rfloor = 497$. We show that the approach based on T^3 obtains better solutions than the ones based on good 2-D approaches like the layers&knapsack method. Since the boxes (and rectangles) are identical, we use the number of packed boxes (and rectangles), instead of the volume, and thus, the ν of this section is the ν of the ρ-algorithm divided by the common volume of the boxes.

Fixed orientation. If the vertical orientation of the boxes is fixed, then the box types are $(5,6,9)$ and $(6,5,9)$. In this case, the problem is essentially 2-D and an optimal solution can be found by applying the ρ-algorithm for the duet T^2, the 2-container (rectangle) $r = (61,44)$ and box types $B = \{b_1, b_2\} = \{(5,6),(6,5)\}$. The corresponding solution $J_3^2 \in T_3^2(r)$ is shown below. Algorithm ρ for the case $n = 2$, $G = T^2$ is the recursive 2-D Morabito-Morales algorithm. J_3^2 packs a total of 89 rectangles of types $(5,6)$ and $(6,5)$ into the $(61,44)$-rectangle. This is optimal and is achieved at recursion level $k = 3$. To find a 3-D solution we just have to stack $\lfloor 50/9 \rfloor$ layers of the J_3^2 packing pattern shown below. There is a total of $5 \times 89 = 445$ boxes. This is the optimum with fixed vertical orientation.

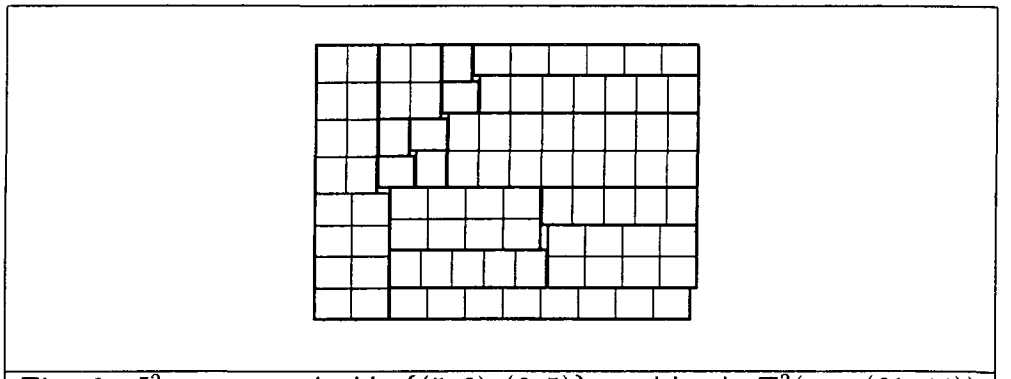


Fig. 6: J_3^2: a most valuable $\{(5,6),(6,5)\}$-partition in $T_3^2(r = (61,44))$

Non fixed orientation - layers&knapsack approach. If the vertical orientation of the boxes is no longer fixed, then the loading can be substantially improved. One possibility to solve the 3-D problem is to have the packing pattern divided into layers, where each layer is a 2-D pattern made of box types which have constant third dimension. We have three possible directions to stack the layers: left-to-right, front-to-back and top-to-bottom. Each choice guide us to a set of three 2-D problems and one corresponding knapsack problem to find the number of layers of each type. So the layers&knapsack method solves nine 2-D packing problems (by the ρ-algorithm with $G = T^2$) and three corresponding knapsack problems.

Left-to-right. Optimal values of the first three 2-D problems, corresponding to the 2-container (rectangle) $r_1 = (44,50)$ are

$$(1)\ \nu_{\{(5,6),(6,5)\}}T_3^2(r_1) = 73, \quad (2)\ \nu_{\{(5,9),(9,5)\}}T_1^2(r_1) = 48, \quad (3)\ \nu_{\{(6,9),(9,6)\}}T_1^2(r_1) = 37.$$

Thus, the associated knapsack problem is

$$\max 73x_1 + 48x_2 + 37x_3, \quad \text{subject to } 9x_1 + 6x_2 + 5x_3 \leq 61, \quad x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_i \text{ integer},$$

where x_1, x_2, x_3 correspond respectively to the number of layers using the patterns of problems 1, 2, 3, as above. An optimal solution for this knapsack problem is $x_1 = 5, x_2 = 1, x_3 = 2$. Therefore, we obtain a solution with $5 \times 73 + 1 \times 48 + 2 \times 37 = 487$ boxes. Recall that the solution with fixed vertical orientation loaded only 445 boxes.

Back-to-front. Optimal values of the next three 2-D problems, corresponding to the rectangle $r_2 = (61,50)$ are

$$(4)\ \nu_{\{(5,6),(6,5)\}}T_1^2(r_2) = 101, \quad (5)\ \nu_{\{(5,9),(9,5)\}}T_1^2(r_2) = 67, \quad (6)\ \nu_{\{(6,9),(9,6)\}}T_1^2(r_2) = 53.$$

The associated knapsack problem is

$$\max 101x_1 + 67x_2 + 53x_3, \text{ subject to } 9x_1 + 6x_2 + 5x_3 \leq 44, \quad x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_i \text{ integer.}$$

An optimal solution for this knapsack problem is $x_1 = 3, x_2 = 2, x_3 = 1$. Therefore, we obtain a solution with $3 \times 101 + 2 \times 67 + 1 \times 53 = 490$ boxes.

Top-to-bottom. Optimal values of the last three 2-D problems, corresponding to the rectangle $r_3 = (61, 44)$ are

$$(7)\ \nu_{\{(5,6),(6,5)\}}T_3^2(r_3) = 89, \quad (8)\ \nu_{\{(5,9),(9,5)\}}T_3^2(r_3) = 58, \quad (9)\ \nu_{\{(6,9),(9,6)\}}T_1^2(r_3) = 46.$$

The associated knapsack problem this time is

$$\max 89x_1 + 58x_2 + 46x_3, \text{ subject to } 9x_1 + 6x_2 + 5x_3 \leq 50 \quad x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_i \text{integer.}$$

An optimal solution for this knapsack problem is $x_1 = 5, x_2 = 0, x_3 = 1$. Therefore, we obtain a solution with $5 \times 89 + 0 \times 58 + 1 \times 46 = 491$ boxes.

Non fixed orientation - T^3 based approach. Now we show the output of the T^3-based ρ-algorithm for $B = \{b_1, b_2, \ldots, b_6\}$=$\{(5,6,9),(5,9,6),(6,5,9),(6,9,5),(9,5,6),(9,6,5)\}$, $c = (61, 44, 50)$, and $k = 1, 2, 3$.

Level 1. The 3-D solution obtained at level $k = 1$ of the recursion already loads 490 boxes in a pattern involving 8 subcontainers, $\{A, B, C, D, E, F, G, H\}$, as shown below (Fig. 7). The δ-triad at the right of the figure completely specifies the container partition. The homogeneous loading inside the subcontainers is given in a *load instruction plan* with two layers at the left of the figure. They are self explanatory: for instance, subcontainer G is a 5-stack of (11×4)-arrangement of $(5\times6\times9)$-boxes, containing 220 boxes. The first four blocks start at depth 0 and the last four at depth 5. The number of boxes packed into the subcontainers are $A:3, B:1, C:24, D:18, E:35, F:9, G:220, H:180$, for a total of 490 boxes.

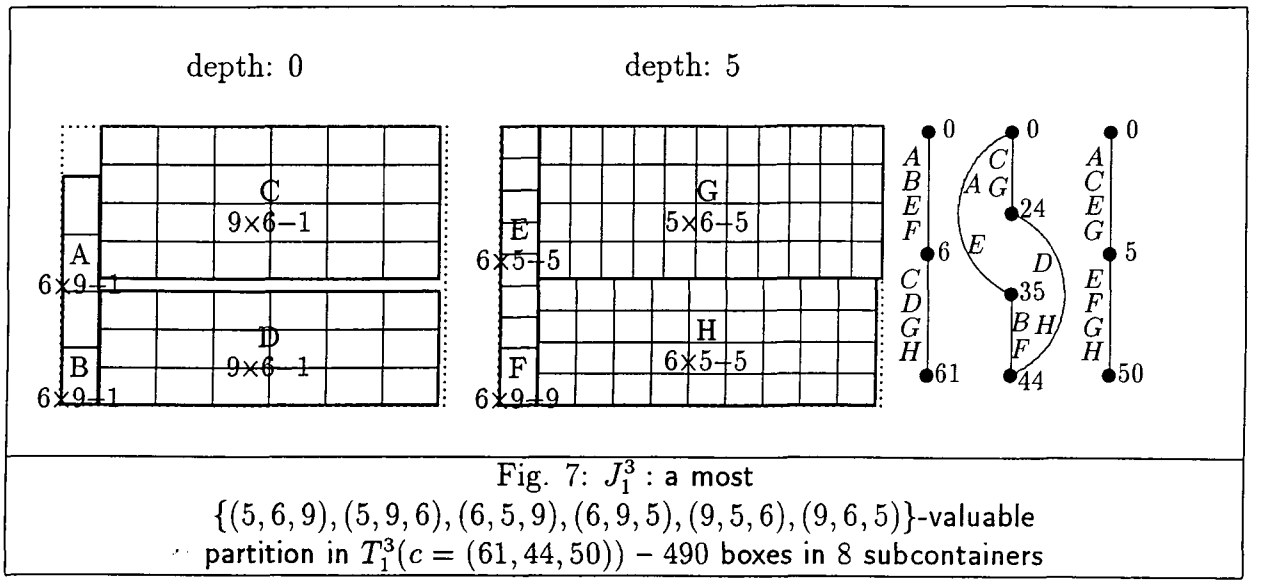


Fig. 7: J_1^3 : a most $\{(5,6,9),(5,9,6),(6,5,9),(6,9,5),(9,5,6),(9,6,5)\}$-valuable partition in $T_1^3(c = (61, 44, 50))$ – 490 boxes in 8 subcontainers

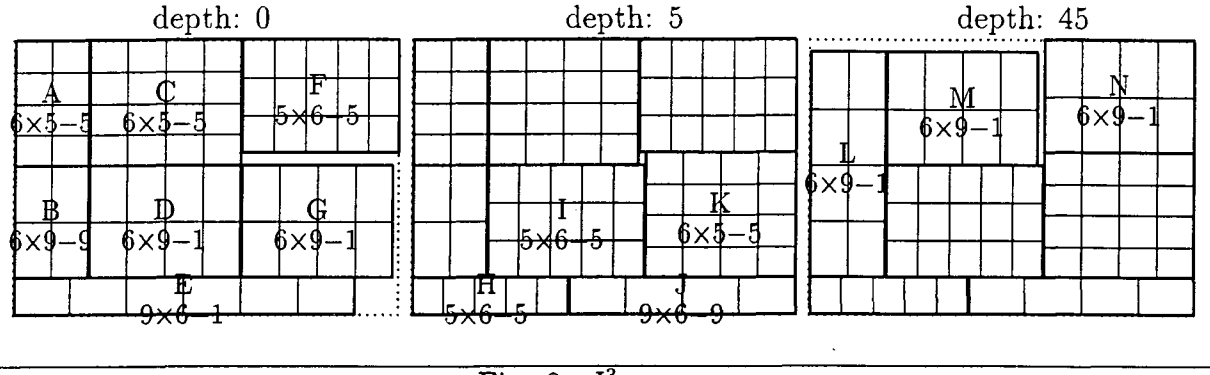


Fig. 8: J_2^3 : a most $\{(5,6,9),(5,9,6),(6,5,9),(6,9,5),(9,5,6),(9,6,5)\}$-valuable partition in $T_2^3(c=(61,44,50))$ – 493 boxes in 14 subcontainers

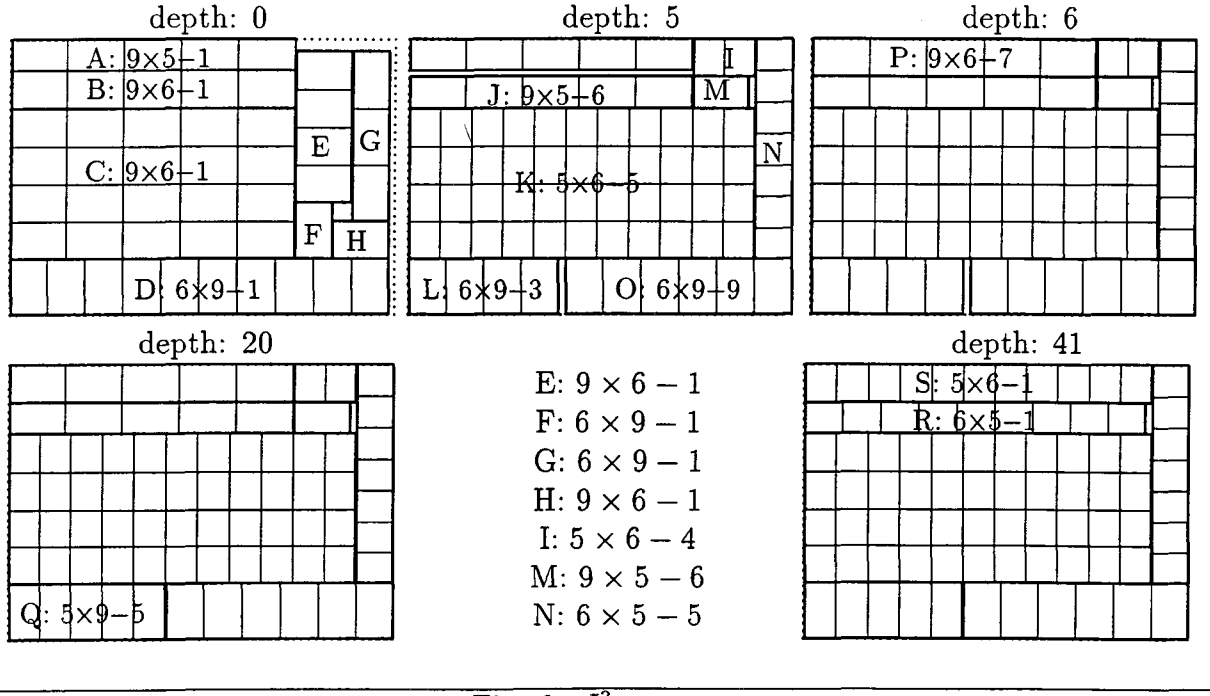


Fig. 9: J_3^3 : a most $\{(5,6,9),(5,9,6),(6,5,9),(6,9,5),(9,5,6),(9,6,5)\}$-valuable partition in $T_3^3(c=(61,44,50))$ – 494 boxes in 19 subcontainers

Levels 2 and 3. Fig. 8 shows that the packing pattern obtained at level $k = 2$ (493 boxes) is better than the best pattern found by the layers&knapsack approach (491 boxes). In Fig. 9 we show a solution at level $k = 3$ (494 boxes), which leaves a gap of 3 boxes relative to the volume upper bound. Note that the best 3-D pattern obtained at level 3 (Fig. 9) loaded 4 boxes more than the pattern found at level 1 (Fig. 7), and 1 box more than the pattern at level 2 (Fig. 8). It loaded 3 boxes more than the best solution found by the 2-D approaches. Other examples are given in section 6.

5 Degeneracy and symmetry issues

Degeneracy and minors. If a depth assignment δ is degenerated, then $\lambda_\delta(i_j) = 0$ for some edge i_j. It follows that $V_\delta(i) = 0$. That is, subcontainer s_i has zero volume, and may be discarded. In terms of the δ-tet, this means to contract the edge i_j in G_j (identifying its ends) and to delete the edges i_k in the other profiles G_k, $k \neq j$. This operation is called *a permissible contraction in* G_j. The result is an δ-tet G' having $|E'| = |E| - 1$. A *δ-minor* of G^δ is a non-degenerated δ-tet H obtained from G^δ by a sequence of permissible contractions in the profiles. A *minor* is the subjacent tet H of a δ-minor H^δ. The set of minors of an n-tet G is denoted by $minors(G)$. We note that different degenerated δ's may yield the same minor. Indeed, as we shall see, for T^3 there is a set of $7^3 - 1 = 342$ patterns of degenerated δ's and only 63 proper minors. In this section we show how to take advantage of minors and their symmetries to quotient the number of choices in step 2 of the ρ-algorithm. After the introduction of general issues, we exemplify the theory in the case of $G = T^3$, by presenting in appendix A the proper minors of T^3 (excluding itself). Various time saving crucial modifications in the ρ-algorithm are effected to take into account these ideas.

The precode of an n-tet. Let G be an n-tet. Given $i \in E$, let σ_i be the $2n$ sequence: $\sigma_i := (t(i_1), h(i_1), t(i_2), h(i_2), \ldots, t(i_n), h(i_n))$. If $i \neq k$, then it follows from the disjointness property that σ_i is different from σ_k. We say that i is *G-smaller* than k, $i <_G k$, if in the first entry where σ_i differs from σ_k, the smallest of the two is in σ_i (lexicographical ordering). In this way, an n-tet G induces a total ordering of E, its *intrinsic ordering*. In general the intrinsic ordering is different from the canonical (inherited from the integers) ordering. It is useful to apply the inverse of the permutation given by the intrinsic ordering to relabel the elements of E in such a way that the G-ordering becomes $(1, 2, \ldots, e)$. This relabeling is the *intrinsic labeling of* E and depends only on the disjointness property of G. Let G be an n-tet with E being intrinsically labeled. The *precode of* G is the sequence of length $2 \times e \times n$, $\kappa'(G)$, given by

$$\begin{aligned}\kappa'(G) = (&t(1_1)h(1_1)t(2_1)h(2_1)\ldots t(e_1)h(e_1),\\ &t(1_2)h(1_2)t(2_2)h(2_2)\ldots t(e_2)h(e_2),\\ &\ldots\\ &t(1_n)h(1_n)t(2_n)h(2_n)\ldots t(e_n)h(e_n)).\end{aligned}$$

An n-tet can be easily reconstructed from its precode and in the reconstruction we get the intrinsic labelling. The n-tets G and H, having the same number of edge-labels are isomorphic if there is a permutation γ of E such that $t(i_j^G) = t(\gamma(i)_j^H)$, $h(i_j^G) = h(\gamma(i)_j^H)$, or, in other words, if there is an E-relabeling γ of the edge-label set of G transforming it into H. Clearly, it is rather simple to compute the precode of an n-tet. This fact is important because it solves the isomorphism problem for n-tets: it is easy to show that the n-tets G and H are isomorphic if and only if $\kappa'(G) = \kappa'(H)$.

The reflexive group and the code of an n-tet. Let G be an n-tet. The reversal of ordering of the vertices in a profile G_j of an n-tet, *putting G_j upside-down*, corresponds to a reflexion of the partition in the hyperplane perpendicular to the j-th axis. The set of such transformations, 2^n number, depends only on the dimension n and is denoted by R_n, the *reflexive group of* G. Each element in R_n is denoted by an n-bit sequence with a τ in the exponent: for instance, $R_3 = \{(000)^\tau, (100)^\tau, (010)^\tau, (110)^\tau, (001)^\tau, (101)^\tau, (011)^\tau, (111)^\tau\}$. R_n acts on the set of n-tets as

follows: $(k_1 k_2 \dots k_n)^\tau(G) = H$ where G and H are n-tets, and H_j is either G_j or G_j upside-down, depending on whether k_j is 0 or 1. From the point of view of their packing properties, G and $r(G)$ (where $r \in R_n$) are equivalent. However, they might be non-isomorphic n-tets. Define $R_n(G)$ as $\{r(G) \mid r \in R_n\}$. It is convenient to distinguish one member of $R_n(G)$ to work with. The *code of* G, $\kappa(G)$, is the lexicographically minimum among the precodes of $G' \in R_n(G)$. Therefore, among other useful facts to be displayed, the code permits us to work with only one member of $R_n(G)$.

We show in Fig. 10 the initial and final steps in computing the code of T^3. Initially the E-labeling is arbitrary. The labels of the vertices of the profiles are not shown explicitly, but they are $0, 1, 2, 3$ from top to bottom. We get the eight precodes of the members of $R_3(T^3)$ by relabeling them intrinsically and choosing the smallest precode. This is attained when we keep the orientation of the first and second profiles and put G^3 upside-down. The new edge 1 is old edge 7,..., etc. The resulting n-tet is the distinguished member of $R_3(T^3)$, which we use in the computer implementation. In general, it is convenient not make any distinction between an n-tet and its code.

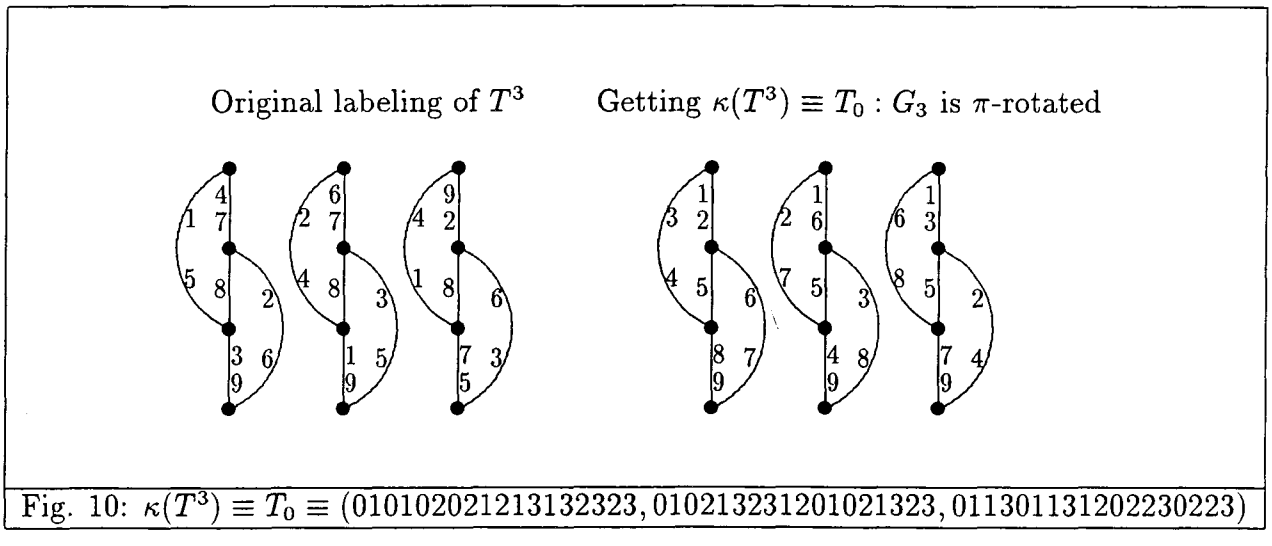


Fig. 10: $\kappa(T^3) \equiv T_0 \equiv (010102021213132323, 010213231201021323, 011301131202230223)$

Types of degeneracy of depth assignments. Given a profile G_j of an n-tet G, there are $2^{v_j - 1} - 1$ types of degeneracy. Indeed, these types are in a $1-1$ correspondence with the set of $(v_j - 1)$-bits vectors except for the all null vector $(00\dots0)$. To the vector $(k_1 k_2 \dots k_{v_j - 1})$ with u $1's$ corresponds a partition of $\{0, 1, 2, \dots, v_j\}$ into $u + 1$ classes as follows: start by writing down the sequence $|0\ 1\ 2\ \dots v_j - 1|$ and whenever $k_j = 1$, put a vertical bar before j. Each adjacent pair of the $u + 2$ vertical bars define a class of the partition formed by consecutive integers, a total of $u + 1$ classes. δ_j is of *degeneracy type* $(k_1 k_2 \dots k_{v_j - 1})$ if for any class C of the partition just defined, $v_i, v_k \in C \Rightarrow \delta_j(v_i) = \delta_j(v_k)$. The type $(00\dots0)$ is ruled out because it corresponds to the partition into a single class $\{0, 1, 2, \dots, v_j - 1\}$, and this partition would imply that the j-th dimension of the container, c_j, was zero, which is of no interest. Thus, since degeneracy is independent in the profiles, by considering all the profiles of an n-tet G, there are $\prod\{2^{v_j - 1} - 1 \mid j \in N\}$ types of degeneracy (including no degeneracy) of depth assignments. Filtering these by the code we are able to obtain the set of minors of G, $\{H_0, H_1, H_2, \dots, H_q\}$, where H_0 is G itself and H_1 is the most degenerated minor, having $|E(H_1)| = 1$.

Algorithm ρ_1. Algorithm ρ of Section 2 can be substantially speeded up if, instead of working with all depth assignments possibly degenerated, we make an outer loop which first chooses a

minor H and then an inner loop which chooses all non-degenerated depths assignments for H. To implement this work's saving idea, we need to store a function $\mu^*_B(G_{k'}(c'))$ which informs the minor H that was used in the first subdivision of c' according to the pattern given by H itself and by $\delta^*_B(G_{k'}(c'))$. Variable μ' denote the current best minor and $|E(H_1)|$=1. Let $Adm^*(H,c')$ be the subset of $Adm(H,c')$ consisting of the non-degenerated members.

Recursive Algorithm ρ_1: Input (B,G,c,k). Output $(\mu^*_B(G_k(c)), \delta^*_B(G_k(c)), \nu_B(G_k(c)))$.
Initialization: Form the sets $X_j(B,c_j)$, $j \in N$. Allocate adequate memory space for storing $\mu^*_B(G_{k'}(c'))$, $\delta^*_B(G_{k'}(c'))$ and $\nu_B(G_{k'}(c'))$, where $0 \leq k' \leq k$ and $c' \subseteq c$. Main call: Call routine ρ_1 with parameters (B,G,c,k). Exit.
Routine $\rho_1(B,G,c',k')$: returns the triple $(\mu^*_B(G_{k'}(c')), \delta^*_B(G_{k'}(c'), \nu_B(G_{k'}(c'))))$.
Step 0: If $k'=0$ then $(\mu',\delta',\nu') = (H_1, \emptyset, BHO(c',B))$, and go to Step 3. Step 1: Make $\mu' = H_1$, $\delta' = \emptyset$ and $\nu' = \nu_B(G_{k'-1}(c'))$. Step 2: For each $H \in minors(G)$ do Step 2.1: For each $\delta \in Adm^*(H,c')$ whose entries of δ_j are in $X_j(B,c'_j)$ do Step 2.1.1: Determine the $e = \|E(H)\|$ subcontainers, $s_i(H^\delta), i \in E(H)$. Step 2.1.2: If $\nu'' = \sum_{i=1}^{e} \nu_B(G_{k'-1}(s_i(H^\delta))) > \nu'$, then $\mu' = H$, $\delta' = \delta$, $\nu' = \nu''$. Step 3: Return the best found solution $\rho_1(B,G,c',k') = (\mu',\delta',\nu')$.

The case $G = T_m$. Suppose now that G is one of the minors of T^3. In this case, since $v_j \leq 4$, there are at most 7 possible types of degeneracies at a profile G_j, and a maximum of $7^3 = 343$ degeneracy types for G, including no degeneracy. The 7 types of degeneracy of a profile can be represented by a letter in A,B,C,D,E,F,G, according to the partition of the set of vertex labels $\{0,1,2,3\}$ into subsets with the same image under δ_j: $A = \{0\},\{1\},\{2\},\{3\}$ (no degeneracy), $B = \{0\},\{1,2\},\{3\}$, $C = \{0,1\},\{2\},\{3\}$, $D = \{0\},\{1\},\{2,3\}$, $E = \{0,1,2\},\{3\}$, $F = \{0\},\{1,2,3\}$, and $G = \{0,1\},\{2,3\}$. A degeneracy type of G can then be represented by a sequence of 3 letters, *a 3-sequence,* in $\{A,B,C,D,E,F,G\}$. Let t_{XYZ} denote all the 3-sequences which induce the same minor as degeneracy of type XYZ. The reader can verify that $t_{EBA} = \{EBA, EBB, EBC, FBA, FBB, FBD, GBB\}$. All of these degeneracies induce the same minor denoted by $\mathsf{minor}(t_{EBA})$.

By starting with the 343 degeneracy types and filtering them using the code, we are able to identify the 64 minors of T^3. The minors of T^3 are denoted by $T_0, T_1, \ldots, T_{63}$. They are thoroughly presented in appendix A. T_0 is T^3 itself and is shown at the right of Fig. 10. Each T_i is specified as $\mathsf{minor}(t_{XYZ})$ for some degeneracy type XYZ. For instance, T_{15} is equal to $\mathsf{minor}(t_{EBA})$. The subset of degeneracy types inducing T_i is denoted $D(T_i)$. In appendix A we show the cardinalities, $|D(T_i)|$, for $i = 1,2,\ldots,63$ ($|D(T_i)|$ is the number in parenthesis). In particular, $|D(T_0)| = 1$, $|D(T_1)| = 57$ and $|D(T_{15})| = 7$. These cardinalities give an indication of the saving in going from ρ to ρ_1. In appendix A, XYZ before the parenthesis means that $T_i = \mathsf{minor}(t_{XYZ})$.

Precode fixing subgroups and improvements in ρ_1. Algorithm ρ_1 can be significantly speeded up if we consider the subsets, $F(H)$, of transformations in R_n which applied to H do not change its code. Given an n-tet H with $\kappa(H) = \kappa'(H)$, we define the *H-precode fixing subgroup of R_n* to be $F(H) = \{r \in R_n \mid \kappa'(r(H)) = \kappa'(H) = \kappa(H)\}$.

Consider, for example, $n = 3$. There are 12 subgroups of R_3. Their set, denoted by $\mathcal{R}_3$, is {0,01,02,04,07,0123,0145,0167,0246,0257,0347, 01234567}. Each index corresponds to an element

of R_3: $0 = (000)^\tau$, $1 = (100)^\tau$, $2 = (010)^\tau$, $3 = (110)^\tau$,$4 = (001)^\tau$, $5 = (101)^\tau$, $6 = (011)^\tau$, $7 = (111)^\tau$. The justaposed notation means subset of transformations, for instance, $0145 = \{(000)^\tau, (100)^\tau, (001)^\tau, (101)^\tau\}$. In appendix A we give $F(T_m), m = 1, 2, \ldots, 63$. In particular, $F(T_0) = 07$ and $F(T_1) = F(T_{15}) = 01234567$. In general, let $\mathcal{R}_n$ denote the set of subgroups of R_n.

A partition of $Adm^*(H, c')$ to take advantage of the reflections. Let P_{3^n} be the 3^n-element set $\{\alpha = \alpha_1\alpha_2\ldots\alpha_n \mid \alpha_j \in \{<, =, >\}\}$. For example,

$$P_{3^3} = \{<<<, <<=, <<>, <=<, <==, <=>, <><, <>=, <>>,$$
$$=<<, =<=, =<>, ==<, ===, ==>, =><, =>=, =>>,$$
$$><<, ><=, ><>, >=<, >==, >=>, >><, >>=, >>>\}.$$

A $\delta \in Adm^*(H, c')$ is of *type* $\alpha = \alpha_1\alpha_2\ldots\alpha_n$ if $\delta_j(1) - \delta_j(0)$ α_j $\delta_j(v_j - 1) - \delta_j(v_j - 2)$, $j \in N$. Given $\alpha \in P_{3^n}$, let $Adm^*_\alpha(H, c')$ be the subset of $Adm^*(H, c')$ whose δ's are of type α. If H and c' are already fixed, as in the loops of the algorithms ρ_2 and ρ_3, we define $[\alpha] = Adm^*_\alpha(H, c')$, and $[P_{3^n}] = \{[\alpha] \mid \alpha \in P_{3^n}\}$. Note that, with this definition, $[P_{3^n}]$ is a well defined partition of $Adm^*(H, c')$.

An element $(k_1k_2\ldots k_n)^\tau \in R_n$ acts on $[P_{3^n}]$ in the obvious way: $(k_1k_2\ldots k_n)^\tau([\alpha])$ is defined as $[\alpha']$, where α' is obtained from α by interchanging the j-th symbols '<' and '>' whenever $k_j = 1$, and leaving symbol '=' unchanged, independently of k_j. Recall that if H is endowed with a depth assignment $\delta \in Adm^*(H, c')$, we denote the associated δ-tet by H^δ. Observe that $H^\delta \in G(c')$, where this set is defined in the beginning of section 3. An element $(k_1k_2\ldots k_n)^\tau \in R_n$ acts on $H(c')$ in a natural way: define $(k_1k_2\ldots k_n)^\tau(H^\delta)$ as $J^{\delta'}$, where J and δ' are as follows: tet J is obtained from tet H by putting its profile j upside-down, whenever $k_j = 1$ and leaving the others unchanged; for each $k_j = 1$, define new δ'_j, where $\delta'_j(k) = \delta_j(v_j - 1) - \delta_j(k)$, for all vertices k of the j-th profile of H; define $\delta'_j = \delta_j$, whenever $k_j = 0$. With these definitions it is easy to verify that $J^{\delta'} \in G(c')$.

An element $R \in \mathcal{R}_n$ induces a partition in $[P_{3^n}]$ as follows. Note that $R \subseteq R_n$. Two elements $[\alpha]$ and $[\alpha'] \in [P_{3^n}]$ are *R-related* if there is an element $r \in R$ such that $r([\alpha]) = [\alpha']$. By taking r to be the identity, we see that R is a reflexive relation. If $[\alpha]$ is R-related to $[\alpha']$, then $r([\alpha]) = [\alpha']$ and so, $r(r([\alpha])) = r([\alpha'])$; since $r^2 = r \circ r$ is the identity transformation, it follows that $r([\alpha']) = [\alpha]$, $[\alpha']$ is R-related to $[\alpha]$ and the R-relation is symmetric. If there are $r, r' \in R$ such that $r([\alpha]) = [\alpha']$ and $r'([\alpha']) = [\alpha'']$, it follows that $r'(r([\alpha])) = [\alpha'']$. Since R is a subgroup, $r' \circ r \in R$. This proves that the R-relation is transitive implying that given $\alpha, \alpha' \in P_{3^n}$, either $[\alpha] = [\alpha']$ or else $[\alpha] \bigcap [\alpha'] = \emptyset$. Hence, R-relation is an equivalence relation and it induces a partition on $[P_{3^n}]$, and, by dropping the square brackets [ ], a partition on P_{3^n}. As an example, the partition induced by $0145 = \{(000)^\tau, (100)^\tau, (001)^\tau, (101)^\tau\} \in \mathcal{R}_3$ on P_{3^3} has 12 classes and is

$$| <<<, <<>, ><<, >< > | <<=, ><= | <=<, <=>, >=<, >=> |$$
$$| <==, >== | <><, <>>, >><, >>> | <>=, >>= | =<<, =<> |$$
$$| =<= | ==<, ==> | === | =><, =>> | =>= |.$$

For each element $R \in \mathcal{R}_n$ and n-tet G, let $M_G(R)$ denote the set of minors H of G which have $F(H) = R$. For instance, if $G = T^3$, then $M_{T^3}(0145) = \{T_6, T_{12}, T_{47}\}$. In appendix B, restricting

to $G = T^3$, we give the partitions on P_{3^3} induced by each element $R \in \mathcal{R}_3$ as well as the values of M_{T^3}. Let $Q_{3^n}(H)$ denote the subset of P_{3^n} formed by choosing one element (say, the first) of each part of the partition induced by $F(H)$. In appendix C, restricting to $G = T^3$, we give $Q_{3^3}(T_m), m = 1, 2, \ldots, 63$. For instance, $F(H) = 0145 \Rightarrow$

$$Q_{3^3}(H) = \{<<<, <<=, <=<, <==, <><, <>=, =<<, =<=, ==<, ===, =><, =>=\}.$$

Reflexive savings and algorithm ρ_2. Here is the main point of all this analysis: it is enough to choose $\delta \in \bigcup\{ Adm^*_\alpha(H, c') \mid \alpha \in Q_{3^n}(H)\}$. And in doing so, we are, implicitly, choosing δ's in the whole set $Adm^*(H, c')$. This substantial saving is made possible by the reflexive symmetries. Indeed, if $\delta' \in [\alpha']$ and $\alpha' \notin Q_{3^n}(H)$ with $\kappa(H) = \kappa'(H)$, then there is a unique α in $Q_{3^n}(H)$ such that α, α' are in the same part of the partition induced by $F(H)$. This implies that there is an $r \in F(H)$ such that $r([\alpha']) = [\alpha]$. Instead of using $H^{\delta'}$ directly we are free to use any of its reflected forms. Note that, since $\kappa'(r(H)) = \kappa'(H)$, $r(H) = H$ and therefore, $r(H^{\delta'}) = (r(H))^{r(\delta')} = H^{\delta}$, where $\delta = r(\delta') \in [\alpha]$. Thus, $H^{\delta'}$ is taken care of by H^{δ}.

Follows the refined algorithm ρ_2, which chooses $\delta's$ in a minimally sufficient subset of $Adm(H, c')$ $\delta \in \bigcup\{Adm^*_\alpha(H, c') \mid \alpha \in Q_{3^n}(H)\}$. It has an extra loop in α which assumes all the values in $Q_{3^n}(H)$. The ρ_2-algorithm takes full advantage of the reflexive symmetries. The reader can notice the effectiveness in taking into account these symmetries by comparing $t(\rho_1)$ and $t(\rho_2)$ in the table of section 6. Recall that minor H_1 is the one which has $|E(H_1)| = 1$.

Recursive Algorithm ρ_2: Input (B, G, c, k). Output $(\mu^*_B(G_k(c)), \delta^*_B(G_k(c)), \nu_B(G_k(c)))$.
Initialization: Form the sets $X_j(B, c_j)$, $j \in N$. Allocate adequate memory space for storing $\mu^*_B(G_{k'}(c'))$, $\delta^*_B(G_{k'}(c'))$ and $\nu_B(G_{k'}(c'))$, where $0 \leq k' \leq k$ and $c' \subseteq c$.
Main call: Call routine ρ_2 with parameters (B, G, c, k). Exit.
Routine $\rho_2(B, G, c', k')$: returns the triple $(\mu^*_B(G_{k'}(c')), \delta^*_B(G_{k'}(c'), \nu_B(G_{k'}(c'))))$.
Step 0: If $k' = 0$ then $(\mu', \delta', \nu') = (H_1, \emptyset, BHO(c', B))$, and go to Step 3.
Step 1: Make $\mu' = H_1$, $\delta' = \emptyset$ and $\nu' = \nu_B(G_{k'-1}(c'))$.
Step 2: For each $H \in minors(G)$ do
Step 2.1: For each $\alpha \in Q_{3^n}(H)$ do
Step 2.1.1: For each $\delta \in Adm^*_\alpha(H, c') = [\alpha]$ whose entries of δ_j are in $X_j(B, c'_j)$ do
Step 2.1.1.1: Determine the $e = \|E(H)\|$ subcontainers, $s_i(H^\delta), i \in E(H)$.
Step 2.1.1.2: If $\nu'' = \sum_{i=1}^{e} \nu_B(G_{k'-1}(s_i(H^\delta))) > \nu'$, then $\mu' = H$, $\delta' = \delta$, $\nu' = \nu''$.
Step 3: Return the best found solution $\rho_2(B, G, c', k') = (\mu', \delta', \nu')$.

Rotational symmetries and algorithm ρ_3. Let S_n denote the group of permutations of $1, 2, \ldots, n$. If $p \in S_n$, let $p(c') = p(c'_1, c'_2, \ldots, c'_n)$ be $(c'_{p(1)}, c'_{p(2)}, \ldots, c'_{p(n)})$. Also, if $b_i \in B$, $b_i = (b_{i1}, b_{i2}, \ldots, b_{in})$, let $p(b_i) = (b_{ip(1)}, b_{ip(2)}, \ldots, b_{ip(n)})$. Define also $p(B)$ as $\{p(b_1), p(b_2), \ldots, p(b_n)\}$ and for an n-tet $G = (G_1, G_2, \ldots, G_n)$, let $p(G)$ be $(G_{p(1)}, G_{p(2)}, \ldots, G_{p(n)})$. Due to the recursive nature of ρ_2, many times the container c' has coincident dimensions. Also it may happen in practice that $p(B) = B$ for various $p \in S_n$. These facts are now used to improve ρ_2 into ρ_3. We say that the pair (B, c') *is fixed under* p if $p(B) = B$ and $p(c') = c'$. In this case, the inputs to ρ_2, $(B, G, c'. k')$ and $(p(B), p(G), p(c'), k')$, are clearly equivalent: the action of p on G is generated by a finite number of interchanges of two profiles and each such an interchange corresponds to a $\pm\pi/2$-rotation of the container which switches two axis while leaving the remaining axis fixed; since

$p(c') = c'$, these rotations produces the same container partition and from $p(B) = B$, the resulting packing is a valid one. Note, however, that G and $p(G)$ may not be the same n-tet. In this case, we need only to consider one of the two n-tets G and $p(G)$, and we choose the one with smallest code. In general, define the (B, c')-fixing subgroup of S_n to be $F(B, c') = \{p \in S_n \mid (B, c') \text{ is fixed under } p\}$. The fact that $F(B, c')$ is a subgroup follows from $p_1, p_2 \in F(B, c') \Rightarrow p_1 \circ p_2 \in F(B, c')$, where, for $i \in N$, $p_1 \circ p_2(i) = p_1(p_2(i))$. If F is a subgroup of S_n, let $\kappa_F(G) = \min\{\kappa(G') \mid G' = p(G) \text{ for some } p \in F\}$. Finally, define $M_{rot}(G, B, c') = \{H \text{ minor of } G \mid \kappa(H) = \kappa_{F(B,c')}(H)\}$. Here is the refined algorithm ρ_3, which is ρ_2 with its Step 2 modified:

Recursive Algorithm ρ_3: Input (B, G, c, k). Output $(\mu^*_B(G_k(c)), \delta^*_B(G_k(c)), \nu_B(G_k(c)))$.
Initialization: Form the sets $X_j(B, c_j)$, $j \in N$. Allocate adequate memory space for storing $\mu^*_B(G_{k'}(c'))$, $\delta^*_B(G_{k'}(c'))$ and $\nu_B(G_{k'}(c'))$, where $0 \le k' \le k$ and $c' \subseteq c$.
Main call: Call routine ρ_3 with parameters (B, G, c, k). Exit.
Routine $\rho_3(B, G, c', k')$: returns the triple $(\mu^*_B(G_{k'}(c')), \delta^*_B(G_{k'}(c'), \nu_B(G_{k'}(c'))))$.
Step 0: If $k' = 0$ then $(\mu', \delta', \nu') = (H_1, \emptyset, BHO(c', B))$, and go to Step 3.
Step 1: Make $\mu' = H_1$, $\delta' = \emptyset$ and $\nu' = \nu_B(G_{k'-1}(c'))$.
Step 2: For each $H \in M_{rot}(G, B, c') \subseteq minors(G)$ do
Step 2.1: For each $\alpha \in Q_{3^n}(H)$ do
Step 2.1.1: For each $\delta \in Adm^*_\alpha(H, c') = [\alpha]$ whose entries of δ_j are in $X_j(B, c'_j)$ do
Step 2.1.1.1: Determine the $e = \lvert E(H)\rvert$ subcontainers, $s_i(H^\delta), i \in E(H)$.
Step 2.1.1.2: If $\nu'' = \sum_{i=1}^{e} \nu_B(G_{k'-1}(s_i(H^\delta))) > \nu'$, then $\mu' = H$, $\delta' = \delta$, $\nu' = \nu''$.
Step 3: Return the best found solution $\rho_3(B, G, c', k') = (\mu', \delta', \nu')$.

Algorithm ρ_3 takes full advantage of $\pm\pi/2$-rotations permitted by coincidences in the container dimensions and by the rotational invariance on the set of box types. The reader can notice its effectiveness by comparing $t(\rho_2)$ and $t(\rho_3)$ displayed in the table of next section.

The case $G = T^3$. There are just five subgroups of S_3. They are S_3 itself, $\{(123), (213)\}$, $\{(123), (321)\}$, $\{(123), (132)\}$ and $\{(123)\}$. If $F(B, c') = S_3$, to define $M_{rot}(T^3, B, c')$, consider the 21-element index subset, exch_{123} = { 0,1,2,5,6,14,15,16,19,32,33,34,38,42,43,44,46,54,57,58,60}. As we have computed, $M_{rot}(T^3, B, c') = \{ H_i \mid i \in \mathsf{exch}_{123}\}$. For the second, third and fourth subgroups, we have correpondingly the three 51-element subsets of indices, from which we define, similarly to the case of the full group, $M_{rot}(T^3, B, c')$:

$\mathsf{exch}_{12} := \{0, 1, 2, 3, 5, 6, 7, 8, 10, 14, 15, 16, 17, 18, 19, 21, 22, 25, 26, 27, 32, 33, 34, 35, 36, 37,$
$38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63\},$
$\mathsf{exch}_{13} := \{0, 1, 2, 3, 5, 6, 7, 8, 9, 14, 15, 16, 17, 18, 19, 20, 21, 23, 24, 27, 32, 33, 34, 35, 36, 37,$
$38, 39, 40, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63\},$
$\mathsf{exch}_{23}: = \{0, 1, 2, 4, 5, 6, 9, 11, 13, 14, 15, 16, 19, 20, 21, 23, 24, 27, 28, 30, 32, 33, 34, 35, 36, 37,$
$38, 39, 40, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63\}.$

Finally, if $F(B, c') = \{(123)\}$, then $M_{rot}(T^3, B, c')$ is the whole set of minors of G.

The subcase $G = T_m$. For the important subcase of a minor of T^3 we may take $M_{rot}(T_m, B, c')$ as $M_{rot}(T^3, B, c') \bigcap minors(T_m)$.

The subsubcase $G = T_{57}$. In the case of the minimal bissecting packing pattern T_{57}, which generates the guillotine packing patterns, we have $minors(T_{57}) = \{T_1, T_2, T_3, T_4, T_{15}, T_{23}, T_{25}, T_{57}\}$.

It follows that $M_{rot}(T_{57}, B, c')$ is one of the five subsets, $\{T_1, T_2, T_{15}T_{57}\}$, $\{T_1, T_2, T_3, T_{15}, T_{25}, T_{57}\}$, $\{T_1, T_2, T_3, T_{15}, T_{23}, T_{57}\}$, $\{T_1, T_2, T_4, T_{15}, T_{23}, T_{57}\}$, $\{T_1, T_2, T_3, T_4, T_{15}, T_{23}, T_{25}, T_{57}\}$, according to $F(B,$ be S^3, $(12), (13), (23), Id$. Here (ij) is the 2-element subgroup of S^3 generated by the exchange of i and j.

The subsubcase $G = T_{38}$. In the case of the minimal non-guillotine packing pattern T_{38}, we have $minors(T_{38}) = \{T_1, T_2, T_3, T_4, T_{15}, T_{23}, T_{25}, T_{38}\}$. All the proper minors agree with the previous subsubcase. Thus, it is enough to replace T_{57} by T_{38} throughout.

6 Computational results for a $(50, 50, 50)$-container

In order to illustrate the computational performance of the algorithms ρ_1, ρ_2, ρ_3 for the case $G = T^3$, we present the results obtained for 14 randomly generated examples with a $(c_1, c_2, c_3) = (50, 50, 50)$ container. For each example, the box dimensions (d_1, d_2, d_3) were simply sampled from a uniform distribution in the interval $[5, 25]$. To form the set of boxes types B, in each one of the 14 examples we considered, except for column 2-D, all the six permutations of the corresponding (d_1, d_2, d_3), that it, $B = \{(d_1, d_2, d_3), (d_1, d_3, d_2), (d_2, d_1, d_3), (d_2, d_3, d_1), (d_3, d_1, d_2), (d_3, d_2, d_1)\}$. $B = \{(d_1, d_2, d_3), (d_2, d_1, d_3)\}$, in the case of column 2-D. The lines of the table are ordered by decreasing volume of the boxes.

The columns of table 1 are as follows: (1) the index of the example (Ex); (2) the dimensions and volume of the boxes ($c_1 \times c_2 \times c_3 = vol$); (3) the volume upper bound (v_{ub}); (4) the number of boxes packed in the best 2-D solution (2-D); (5) the number of boxes packed in the best lk solution (lk); (6) the recursion level (k); (7) the number of boxes packed in the best guillotine solution at various recursion levels (gui); (8) the number of boxes packed in a 3-D solution at various recursion levels (3-D); (9) the number of subcontainers corresponding to column (8) ($\#s$); (10) the ρ_1-time corresponding to column (8) ($t(\rho_1)$); (11) the ρ_2-time corresponding to column (8) ($t(\rho_2)$); and (12) the ρ_3-time corresponding to column (8) ($t(\rho_3)$). Column headed by 2-D was obtained by fixing the vertical directions (third entries c_3) and running the 2-D algorithm. For this column and column lk we used the maximum level of the recursion ($k = 3$) in the implicit 2-D problems. Column headed by gui was obtained by simply using $G = T_{57}$ in algorithms ρ_1, ρ_2, ρ_3 (recall from section 3 that minor T_{57} is the generator of all guillotine packings). Times are in seconds and were obtained in a Pentium II at 400MHz.

Note in the table that for all examples the 3-D approach was able to produce solutions better than or equal to the lk approach which, in turn, produces better solutions than the 2-D and gui approaches. Indeed, except for examples 8 and 13, the 3-D solutions were better than the lk solutions. In some instances the difference between the solutions was substantial, such as in example 4, where the 3-D approach loaded 54 boxes at levels 2 and 3, while the lk method loaded only 47 boxes. The behaviour of the lk versus the 3-D approaches is expected because if the lk approach produces a packing pattern P for c with ℓ layers, then $P \in T_k^3(c)$, where k is of the order of $\log_3(\ell)$.

Ex	$d_1 \times d_2 \times d_3 = vol$	v_{ub}	2-D	lk	k	gui	3-D	#s	$t(\rho_1)$	$t(\rho_2)$	$t(\rho_3)$
1	$13 \times 14 \times 23 = 4186$	29	18	24	1	21	24	4	0.8	0.7	0.3
					2	22	26	26	1.8	1.4	1.0
					3	22	26	26	2.5	2.1	1.5
2	$17 \times 20 \times 12 = 4080$	28	16	22	1	20	22	4	1.3	0.9	0.4
					2	22	26	25	2.7	1.9	1.3
3	$11 \times 22 \times 15 = 3630$	30	24	27	1	26	27	4	0.6	0.5	0.2
					2	27	28	7	0.9	0.8	0.4
					3	27	29	19	1.1	0.9	0.6
					4	27	29	21	1.0	1.0	0.5
4	$17 \times 21 \times 6 = 2142$	58	32	47	1	44	46	5	66	31	21
					2	46	54	26	268	115	108
					3	46	54	26	348	163	148
5	$22 \times 8 \times 11 = 1936$	60	48	54	1	52	54	4	670	417	242
					2	54	57	15	26	15	14
					3	54	58	19	31	31	17
6	$18 \times 21 \times 5 = 1890$	65	40	54	1	52	54	5	125	60	43
					2	54	57	16	1087	549	523
7	$9 \times 11 \times 19 = 1881$	66	48	60	1	56	58	7	25	16	11
					2	57	61	17	113	84	53
					3	57	62	28	168	89	81
8	$13 \times 8 \times 18 = 1872$	66	48	64	1	58	60	5	12	7.2	4.7
					2	59	64	8	20	12	9.4
					3	60	64	13	37	33	19
9	$9 \times 16 \times 11 = 1584$	78	64	74	1	69	72	4	20	11	7.4
					2	72	75	19	57	33	27
					3	72	75	13	84	84	40
10	$13 \times 15 \times 7 = 1365$	91	63	79	1	75	79	5	100	53	38
					2	79	84	36	913	371	347
11	$7 \times 9 \times 18 = 1134$	110	78	102	1	100	102	8	36	23	15
					2	100	105	18	237	143	133
					3	100	106	24	285	174	163
12	$7 \times 6 \times 18 = 756$	165	118	158	1	156	158	7	271	132	98
					2	156	160	15	2522	1186	1051
					3	156	161	26	6118	4152	1977
13	$7 \times 9 \times 11 = 693$	180	156	176	1	168	170	8	184	124	91
					2	170	176	10	1371	666	624
					3	170	176	12	2525	1442	1419
14	$9 \times 7 \times 5 = 315$	396	390	394	1	391	395	6	90	53	27
					2	394	396	9	684	505	377

Table 1: Results for 14 examples in a $(c_1, c_2, c_3) = (50, 50, 50)$ container

7 Conclusions and Perspectives

In this study we presented a simple recursive algorithm for the problem of packing n-dimensional boxes into an n-container. The algorithm is based on the representation of a feasible non-guillotine packing as a depth assignment in an n-tet, which is a sequence of n directed graphs. To each of such an assignment of the n-tet there corresponds a partition of the container into specific subcontainers. Each subcontainer can be recursively partitioned into further subcontainers, according to the specifications in the n-tet. The approach utilizes reflexive and rotational symmetry issues that curtail the implicit enumeration substantially.

In order to illustrate the performance of the algorithm, we presented computational results from solving randomly generated examples for the $n = 3$ case, where all boxes are identical and can be packed in any orientation. Nevertheless, restrictions on the allowed orientations and extensions to boxes of different sizes are also treated by the algorithm. Moreover, by changing the input from the standard triad T_0 to its minor T_{57}, the algorithm adapts itself to generate only guillotine packings. 3PP is an NP-hard problem and has various practical applications as in the loading of products on pallets, and inside containers or safe-trucks. The present method was able to produce much better 3-D packings than the usual layer-knapsack approach or the 3-D guillotine approach, within acceptable running time increasing. Much remains to be done in improving the running time of algorithm ρ_3. Another interesting research topic is to find practical applications for the elegant quartet T^4. These topics are currently under investigation.

Acknowledgements: This research was partially supported by FINEP, CNPq (grants #107/97, #30.1103/80, #680082/95-6, #522973/95-7) and FAPESP (grants #9522-0, #97/13930-1).

References

[1] G. Abdou and M. Yang (1994). A systematic approach for the three-dimensional palletization problem. *International Journal of Production Research* 32(10), 2381-2394.

[2] M. Arenales and R. Morabito (1995). An AND/OR-graph approach to the solution of two-dimensionsl non-guillotine cutting problems. *European Journal of Operational Research* 84, 599-617.

[3] M. Arenales, R. Morabito and H. Yanasse (eds.) (1999). Special issue: Cutting and packing problems. *Pesquisa Operacional* 19, 2, 109-284.

[4] R. Balasubramanian (1992). The pallet loading problem: A survey. *International Journal of Production Economics* 28, 217-225.

[5] J. Beasley (1985). An exact two-dimensional non-guillotine tree search procedure. *Operations Research* 33, 49-64.

[6] E. Bischoff and M. Marriott (1990). A comparative evaluation of heuristics for container loading. *European Journal of Operational Research* 44, 2, 267-276.

[7] E. Bischoff and M. Ratcliff (1995). Loading multiple pallets. *Journal of the Operational Research Society* 46, 1322-1336.

[8] E. Bischoff and G. Waescher (eds.) (1995). Special issue on cutting and packing. *European Journal of Operational Research* 84(3), 503-712.

[9] S. Bhattacharya, R. Roy and S. Bhattacharya (1998). An exact depth-first algorithm for the pallet loading problem. *European Journal of Operational Research* 110, 610-625.

[10] A. Bortfeldt and H. Gehring (1998). A tabu search algorithm for weakly heterogeneous container loading problems. *OR Spektrum* 20, 4, 237-250.

[11] C. Chen, S. Lee and Q. Shen (1995). An analytical model for the container loading problem. *European Journal of Operational Research* 80, 1, 68-76.

[12] C. Chien and W. Wu (1998). A recursive computational procedure for container loading. *Computers and Industrial Engineering* 35, 319-322.

[13] K. Dowsland (1987). An exact algorithm for the pallet loading problem. *European Journal of Operational Research* 31, 78-84.

[14] K. Dowsland (1993). Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research* 68, 389-399.

[15] K. Dowsland and W. Dowsland (1992). Packing problems. *European Journal of Operational Research* 56, 2-14.

[16] W. Dowsland (1995). Improving palletization efficiency - the theoretical basis and practical applications. *International Journal of Production Research* 33, 8, 213-222.

[17] H. Dyckhoff and U. Finke (1992). *Cutting and packing in production and distribution: Typology and bibliography*, Springler-Verlag Co, Heildelberg.

[18] H. Dyckhoff, G. Scheithauer, G. and J. Terno (1997). Cutting and packing. In M. Amico, F. Maffioli, F., S. Martello (ed), *Annotated bibliographies in combinatorial optimisation*, John Wiley & Sons, New York, NY, 393-414.

[19] J. George (1992). A method for solving container packing for a single size of box. *Journal of the Operational Research Society* 43, 307-312.

[20] R. Haessler and F. Talbot (1990). Load planning for shipments of low density products. *European Journal of Operational Research* 44, 289-299.

[21] A. Herbert and K. Dowsland (1996). A family of genetic algorithms for the pallet loading problem. *Annals of Operations Research* 63, 415-436.

[22] J. Herz (1972). Recursive computational procedure for two dimensional stock cutting. *IBM Journal of Research Development* 16, 462-469.

[23] L. Lins, S. Lins and R. Morabito (1999). A 9-fold partition heuristic for packing boxes into a container. *Investigacion Operativa* 7(3), 69-82.

[24] T. Liu and C. Hsiao (1997). A three-dimensional pallet loading method for single-size boxes. *Journal of the Operational Research Society* 48, 7, 726-735.

[25] S. Martello (ed.) (1994a), Special issue: Knapsack, packing and cutting, Part I: One dimensional knapsack problems. *INFOR*, 32(3).

[26] S. Martello (ed.) (1994b), Special issue: Knapsack, packing and cutting, Part II: Multidimensional knapsack and cutting stock problems. *INFOR*, 32(4).

[27] F.K. Miyazawa and Y. Wakabayashi (1997). An algorithm for the three-dimensional packing problem with asymptotic performance analysis. *Algorithmica* 18(1), 122-144.

[28] R. Morabito and M. Arenales (1994). An AND/OR-graph approach to the container loading problem. *International Transactions in Operations Research* 1(1), 59-73.

[29] R. Morabito and S. Morales (1998). A simple and effective recursive procedure for the manufacturer's pallet loading problem. *Journal of the Operational Research Society*, 49, 819-828.

[30] R. Morabito, S. Morales and J. Widmer (2000). Loading optimization of palletized products on trucks. To appear in *Transportation Research.*

[31] J. Nelissen (1995). How to use structural constraints to compute an upper bound for the pallet loading problem. *European Journal of Operational Research* 84, 662-680.

[32] G. Scheithauer and G. Sommerweiss (1998). 4-block heuristic for the rectangle packing problem. *European Journal of Operational Research* 108, 3, 509-526.

[33] G. Scheithauer and J. Terno (1996). The G4-heuristic for the pallet loading problem. *Journal of the Operational Research Society* 47, 511-522.

[34] SICUP (2000). Special Interest Group on Cutting and Packing, http://www.prodlog.wiwi.uni-halle.de/sicup/.

[35] P. Sweeney and E. Paternoster (1992). Cutting and packing problems: A categorized, application-oriented research bibliography. *Journal of the Operational Research Society* 43, 691-706.

[36] R. Tsai, E. Malstrom and W. Kuo (1993). Three dimensional palletization of mixed box sizes. *IEE Transactions* 25(4), 64-75.

8 Appendix A: the proper minors of T^3

We present 63 non-degenerated minors of T^3, denoted by T_1, T_2, ..., T_{63}. The T_m-coding fixing subgroup of R_3 is given in its coded form (see the part of the text which follows algorithm ρ_1) between the symbols ":" and "-". The 3-sequence of letters that follows "-" is the type of degeneracy inducing T_m. Finally, the number between parenthesis is the cardinality of the subset $D(T_m)$ of degeneracies of T_m.

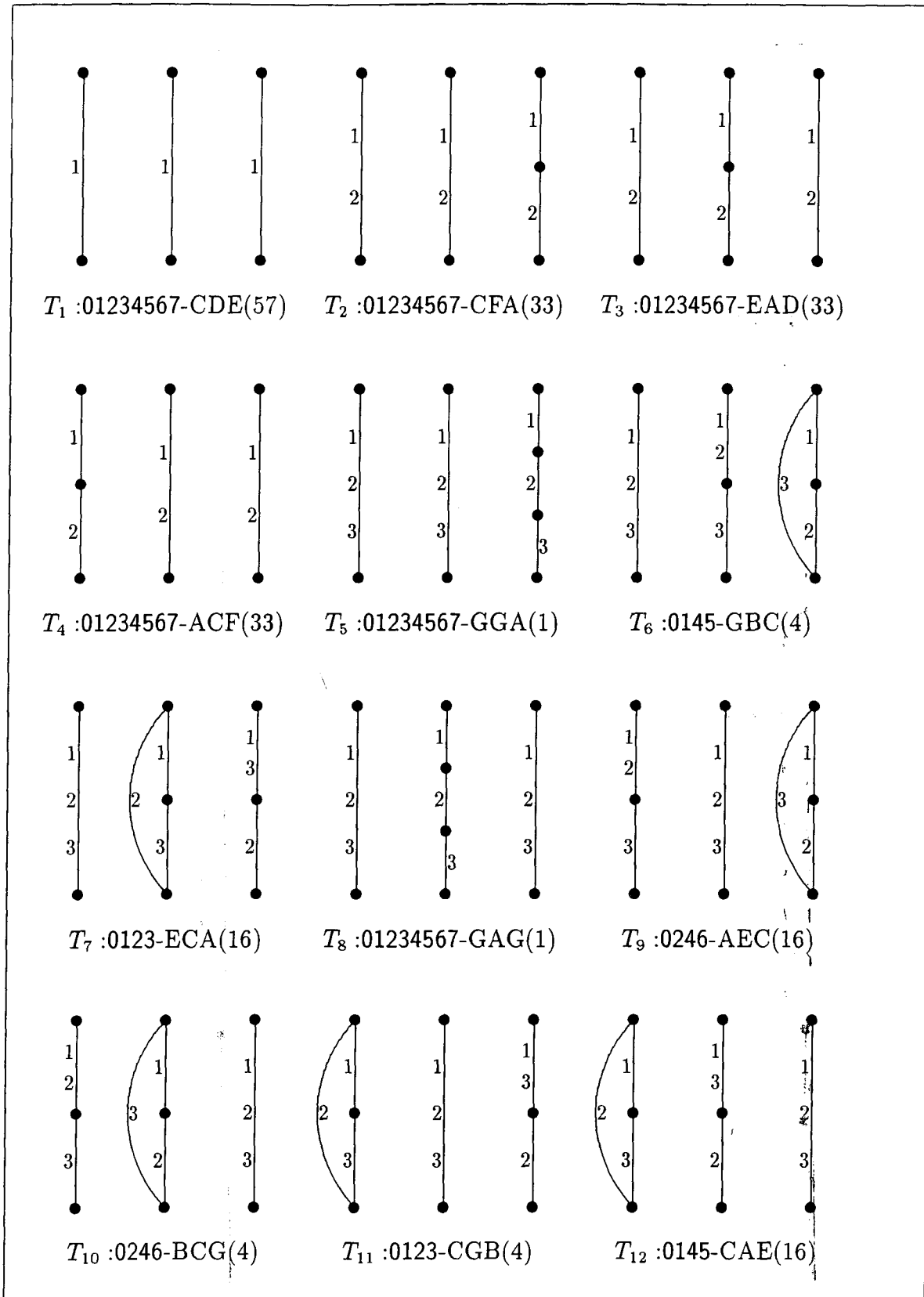


Fig. 11: Minors $T_1 \ldots T_{12}$ of T^3

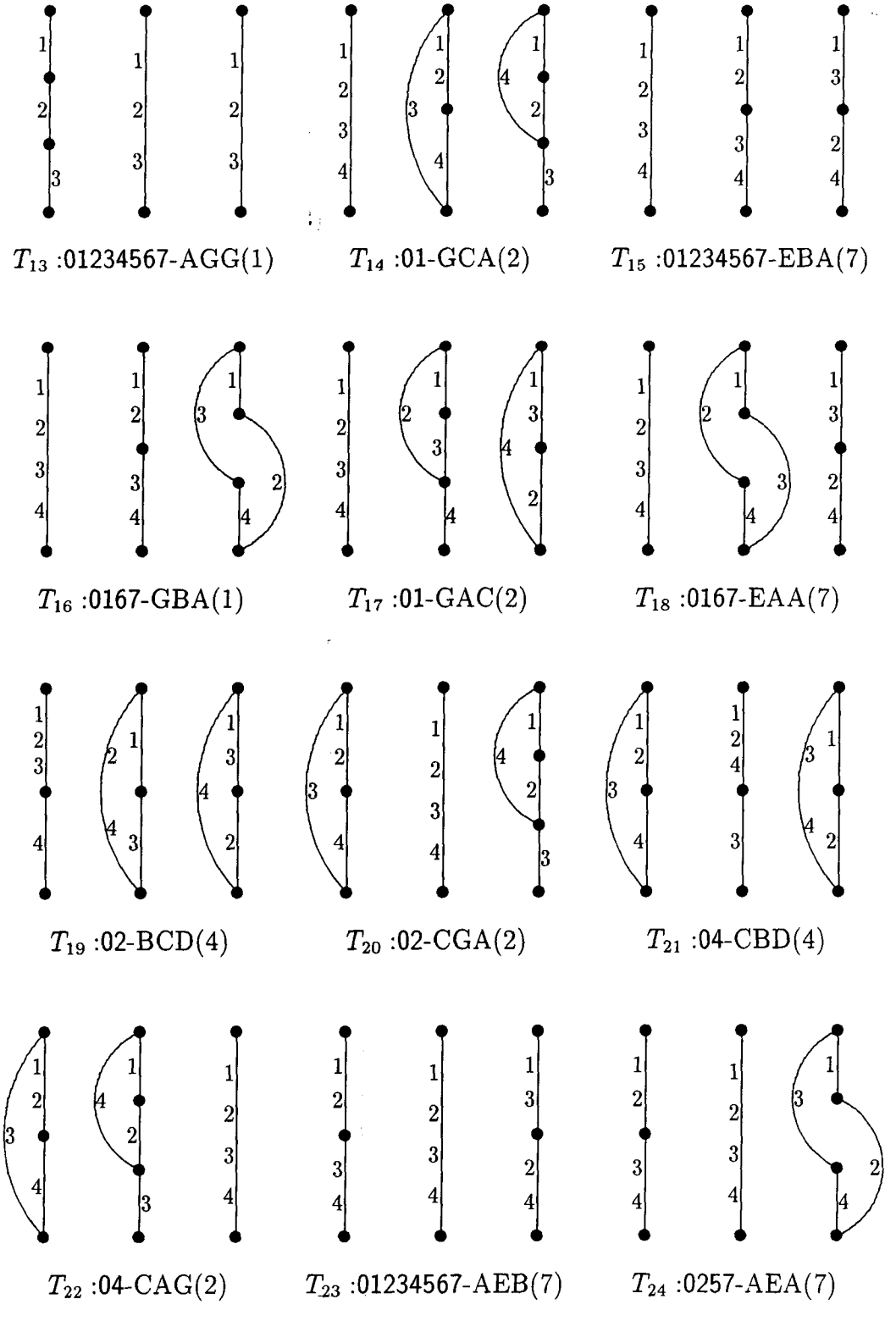
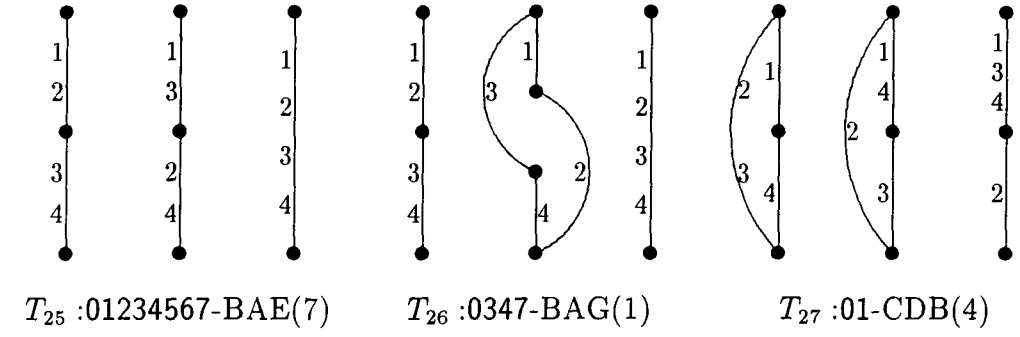


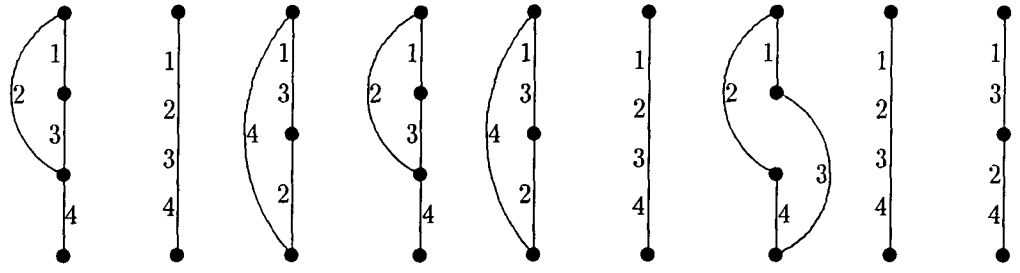
Fig. 12: Minors $T_{13} \ldots T_{24}$ of T^3



T_{25} :01234567-BAE(7)

T_{26} :0347-BAG(1)

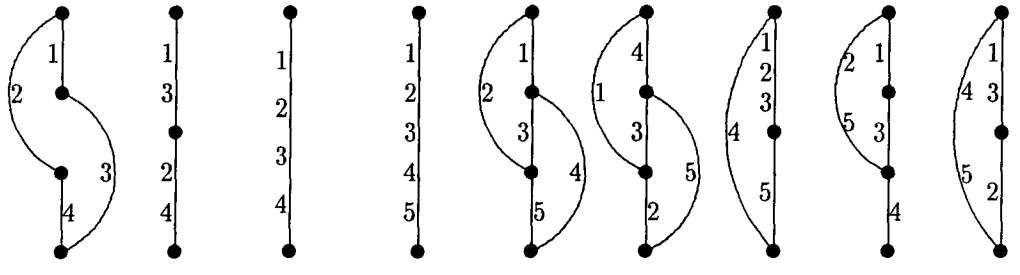
T_{27} :01-CDB(4)



T_{28} :02-AGC(2)

T_{29} :04-ACG(2)

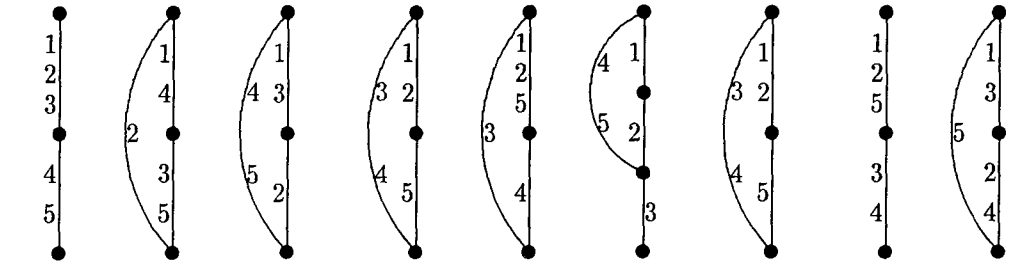
T_{30} :0257-AGB(2)



T_{31} :0347-AAE(7)

T_{32} :0167-GAA(1)

T_{33} :0-CAD(1)

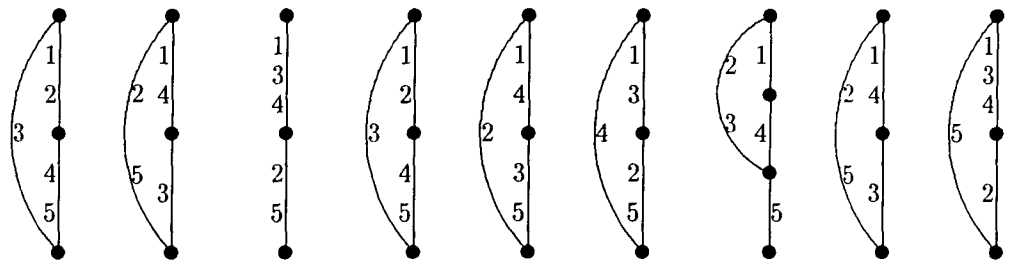


T_{34} :02-BCC(2)

T_{35} :0-CDA(2)

T_{36} :04-CBC(2)

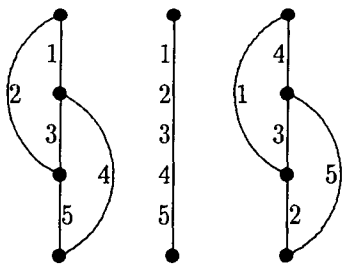
Fig. 13: Minors $T_{25} \ldots T_{36}$ of T^3



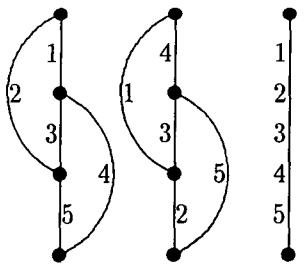
T_{37} :01-CCB(2)

T_{38} :0-CCC(2)

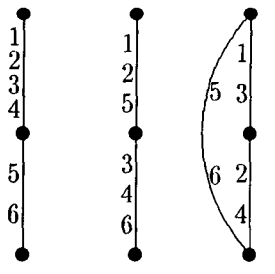
T_{39} :0-ACD(2)



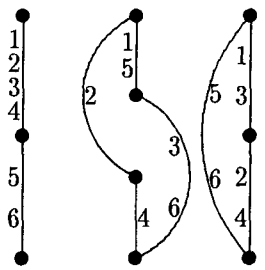
T_{40} :0257-AGA(1)



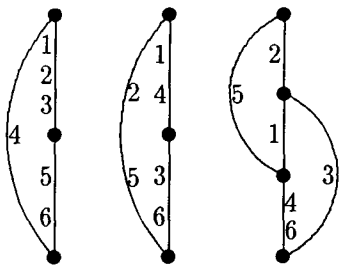
T_{41} :0347-AAG(1)



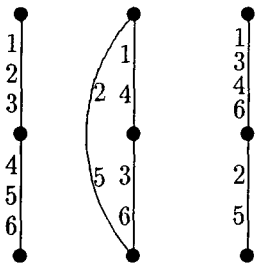
T_{42} :0246-BBC(2)



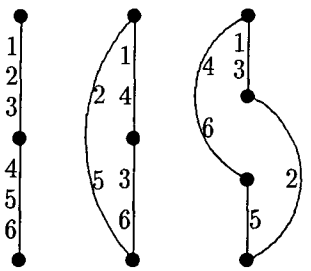
T_{43} :0-BAC(2)



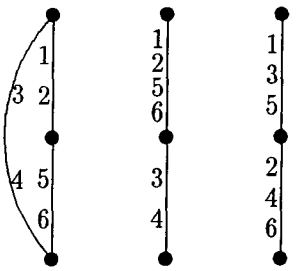
T_{44} :0-CCA(2)



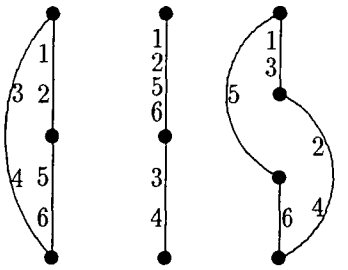
T_{45} :0123-BCB(2)



T_{46} :02-BCA(2)

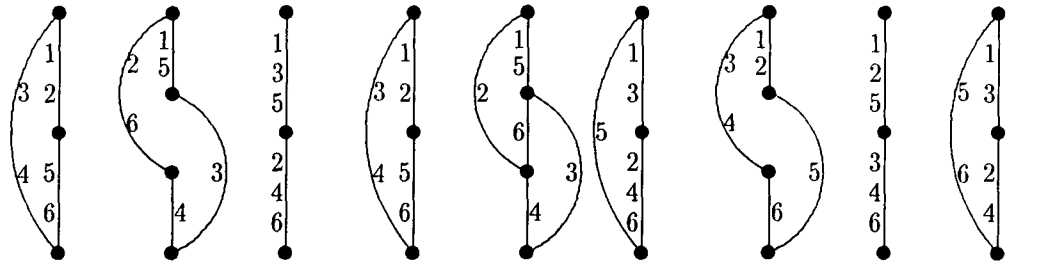


T_{47} :0145-CBB(2)

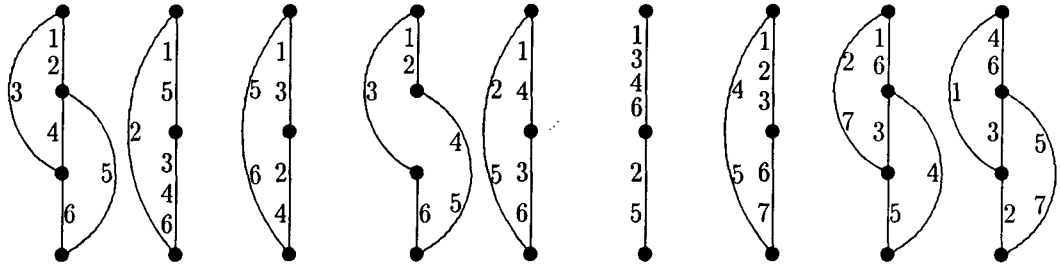


T_{48} :0-CBA(2)

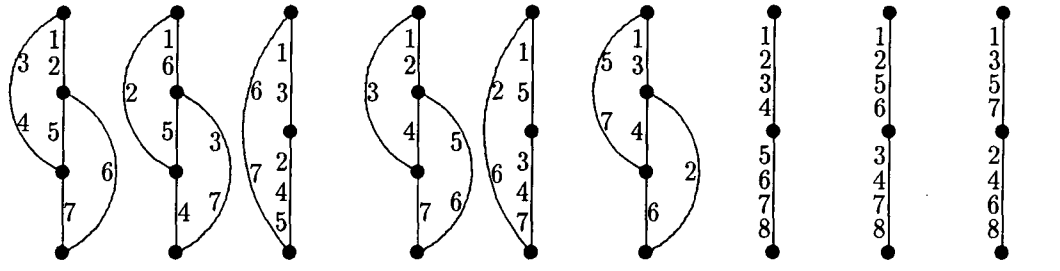
Fig. 14: Minors $T_{37} \ldots T_{48}$ of T^3



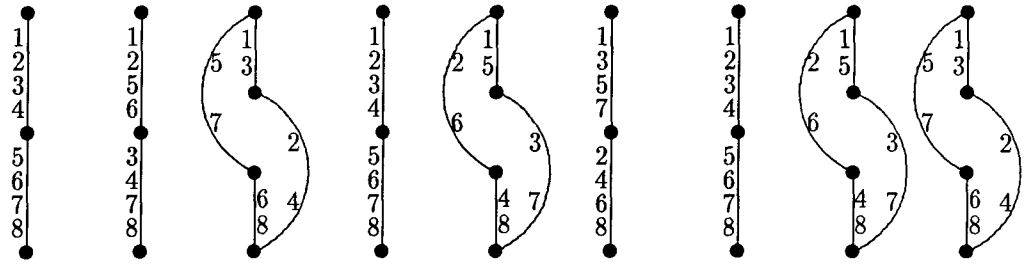
T_{49} :01-CAB(2) T_{50} :0-CAC(2) T_{51} :04-ABC(2)



T_{52} :0-ACC(2) T_{53} :0-ACB(2) T_{54} :0-CAA(2)

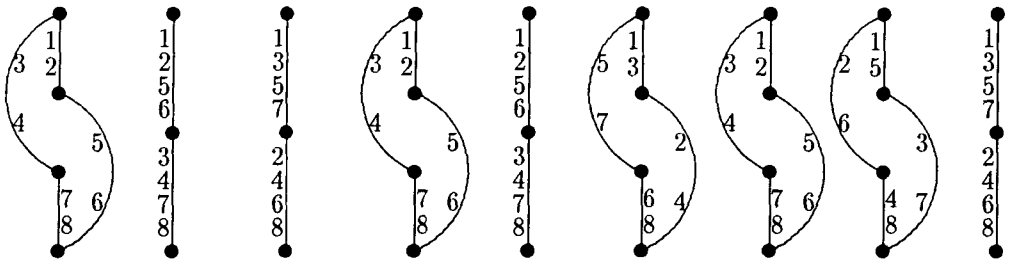


T_{55} :0-AAC(2) T_{56} :0-ACA(2) T_{57} :01234567-BBB(1)



T_{58} :0257-BBA(1) T_{59} :0167-BAB(1) T_{60} :07-BAA(1)

Fig. 15: Minors $T_{49} \ldots T_{60}$ of T^3



T_{61} :0347-ABB(1) T_{62} :07-ABA(1) T_{63} :07-AAB(1)

Fig. 16: Minors $T_{61} \ldots T_{63}$ of T^3

9 Appendix B: The action of $\mathcal{R}_3$ on P_{3^3}

$$M_{T^3}(0) = \{T_{33}, T_{35}, T_{38}, T_{39}, T_{43}, T_{44}, T_{48}, T_{50}, T_{52}, T_{53}, T_{54}, T_{55}, T_{56}\}$$

<<<	<<=	<<>	<=<	<==	<=>	<><	<>=	<>>
=<<	=<=	=<>	==<	===	==>	=><	=>=	=>>
>><	>>=	>>>

$$M_{T^3}(01) = \{T_{14}, T_{17}, T_{27}, T_{37}, T_{49}\}$$

<<<, >><	<<=, >>=	<<>, >>>	<=<, >=<	<==, >==
<=>, >=>	<><, >><	<>=, >>=	<>>, >>>	=<<
=<=	=<>	==<	===	==>	=><	=>=	=>>

$$M_{T^3}(02) = \{T_{19}, T_{20}, T_{28}, T_{34}, T_{46}\}$$

<<<, <><	<<=, <>=	<<>, <>>	<=<	<==	<=>
=<<, =><	=<=, =>=	=<>, =>>	==<	===	==>
><<, >><	><=, >>=	><>, >>>	>=<	>==	>=>

$$M_{T^3}(04) = \{T_{21}, T_{22}, T_{29}, T_{36}, T_{51}\}$$

<<<, <<>	<<=	<=<, <=>	<==	<><, <>>	<>=
=<<, =<>	=<=	==<, ==>	===	=><, =>>	=>=
><<, ><>	><=	>=<, >=>	>==	>><, >>>	>>=

$$M_{T^3}(07) = \{T_0, T_{60}, T_{62}, T_{63}\}$$

<<<, >>>	<<=, >>=	<<>, >><	<=<, >=>	<==, >==
<=>, >=<	<><, ><>	<>=, ><=	<>>, >><	=<<, =>>
=<=, =>=	=<>, =><	=<<, =>>	===

$$M_{T^3}(0123) = \{T_7, T_{11}, T_{45}\}$$

$$| <<<, <><, ><<, >>< | <<=, <>=, ><=, >>= |$$
$$| <<>, ><>, <>>, >>>, ><>, >>> | <=<, >=< | <==, >== |$$
$$| <=>, >=> | =<<, =>< | =<=, =>= | =<>, =>> | ==< | === | ==> |$$

$$M_{T^3}(0145) = \{T_6, T_{12}, T_{47}\}$$
$$| <<<, <<>, ><<, ><> | <<=, ><= | <=<, <=>, >=<, >=> |$$
$$| <==, >== | <><, <>>, >><, >>> | <>=, >>= | =<<, =<> |$$
$$| =<= | ==<, ==> | === | =><, =>> | =>= |$$

$$M_{T^3}(0246) = \{T_9, T_{10}, T_{42}\}$$
$$| <<<, <<>, <><, <>> | <<=, <>= | <=<, <=> | <== |$$
$$| =<<, =<>, =><, =>> | =<=, =>= | ==<, ==> | === |$$
$$| ><<, ><>, >><, >>> | ><=, >>= | >=<, >=> | >== |$$

$$M_{T^3}(0167) = \{T_{16}, T_{18}, T_{32}, T_{59}\}$$
$$| <<<, <>>, ><<, >>> | <<=, <>=, ><=, >>= |$$
$$| <<>, <><, ><>, >>< | === | <=<, <=>, >=<, >=> |$$
$$| <==, >== | =<<, =>> | =<=, =>= | =<>, =>< | ==<, ==> |$$

$$M_{T^3}(0257) = \{T_{24}, T_{30}, T_{40}, T_{58}\}$$
$$| <<<, <><, ><>, >>> | <<=, <>=, ><=, >>= |$$
$$| <<>, <>>, ><<, >>< | <=<, >=< | <==, >== |$$
$$| =<<, =<>, =><, =>> | =<=, =>= | ==<, ==> | <=>, >=> | === |$$

$$M_{T^3}(0347) = \{T_{26}, T_{31}, T_{41}, T_{61}\}$$
$$| <<<, <<>, >><, >>> | <<=, >>= | <=<, <=>, >=<, >=> |$$
$$| <==, >== | <><, <>>, ><>, ><> | <>=, ><= |$$
$$| =<<, =<>, =><, =>> | =<=, =>= | ==<, ==> | === |$$

$$M_{T^3}(01234567) = \{T_1, T_2, T_3, T_4, T_5, T_8, T_{13}, T_{15}, T_{23}, T_{25}, T_{57}\}$$
$$| <<<, <<>, <><, <>>, ><<, ><>, >><, >>> |$$
$$| <<=, <>=, ><=, >>= | <=<, <=>, >=<, >=> |$$
$$| <==, >== | =<<, =<>, =><, =>> | ==<, ==> | =<=, =>= | === |$$

10 Appendix C: $Q_{3^3}(T_m), m = 0, 1, \ldots, 63$

$$M_{T^3}(0) = \{T_{33}, T_{35}, T_{38}, T_{39}, T_{43}, T_{44}, T_{48}, T_{50}, T_{52}, T_{53}, T_{54}, T_{55}, T_{56}\}$$
$$T \in M_{T^3}(0) \Rightarrow Q_{3^3}(T) =$$
$$\{| <<<, <<=, <<>, <=<, <==, <=>, <><, <>=, <>>,$$
$$, =<<, =<=, =<>, ==<, ===, ==>, =><, =>=, =>>,$$
$$, ><<, ><=, ><>, >=<, >==, >=>, >><, >>=, >>>\}$$

$$M_{T^3}(01) = \{T_{14}, T_{17}, T_{27}, T_{37}, T_{49}\}.\ T \in M_{T^3}(01) \Rightarrow Q_{3^3}(T) = \{<<<, <<=, <<>, <=<, <==, <=>, <><, <>=, <>>, =<<, =<=, =<>, ==<, ===, ==>, =><, =>=, =>>\}$$

$$M_{T^3}(02) = \{T_{19}, T_{20}, T_{28}, T_{34}, T_{46}\}.\ T \in M_{T^3}(02) \Rightarrow Q_{3^3}(T) = \{<<<, <<=, <<>, <=<, <==, <=>, =<<, =<=, =<>, ==<, ===, ==>, ><<, ><=, ><>, >=<, >==, >=>\}$$

$$M_{T^3}(04) = \{T_{21}, T_{22}, T_{29}, T_{36}, T_{51}\}.\ T \in M_{T^3}(04) \Rightarrow Q_{3^3}(T) = \{<<<, <<=, <=<, <==, <><, <>=, =<<, =<=, ==<, ===, =><, =>=, ><<, ><=, >=<, >==, >><, >>=\}$$

$$M_{T^3}(07) = \{T_0, T_{60}, T_{62}, T_{63}\}.\ T \in M_{T^3}(07) \Rightarrow Q_{3^3}(T) = \{<<<, <<=, <<>, <=<, <==, <=>, <><, <>=, <>>, =<<, =<=, =<>, =<<, ===\}$$

$$M_{T^3}(0123) = \{T_7, T_{11}, T_{45}\}.\ T \in M_{T^3}(0123) \Rightarrow Q_{3^3}(T) = \{<<<, <<=, <=<, <=<, <==, <=>, =<<, =<=, =<>, ==<, ===, ==>\}$$

$$M_{T^3}(0145) = \{T_6, T_{12}, T_{47}\}.\ T \in M_{T^3}(0145) \Rightarrow Q_{3^3}(T) = \{<<<, <<=, <=<, <==, <><, <>=, =<<, =<=, ==<, ===, =><, =>=\}$$

$$M_{T^3}(0246) = \{T_9, T_{10}, T_{42}\}.\ T \in M_{T^3}(0246) \Rightarrow Q_{3^3}(T) = \{<<<, <<=, <=<, <==, =<<, =<=, ==<, ===, ><<, ><=, >=<, >==\}$$

$$M_{T^3}(0167) = \{T_{16}, T_{18}, T_{32}, T_{59}\}.\ T \in M_{T^3}(0167) \Rightarrow Q_{3^3}(T) = \{<<<, <<=, <<>, ===, <=<, <==, =<<, =<=, =<>, ==<\}$$

$$M_{T^3}(0257) = \{T_{24}, T_{30}, T_{40}, T_{58}\}.\ T \in M_{T^3}(0257) \Rightarrow Q_{3^3}(T) = \{<<<, <<=, <<>, <=<, <==, =<<, =<=, ==<, <=>, ===\}$$

$$M_{T^3}(0347) = \{T_{26}, T_{31}, T_{41}, T_{61}\}.\ T \in M_{T^3}(0347) \Rightarrow Q_{3^3}(T) = \{<<<, <<=, <=<, <==, <><, <>=, =<<, =<=, ==<, ===\}$$

$$M_{T^3}(01234567) = \{T_1, T_2, T_3, T_4, T_5, T_8, T_{13}, T_{15}, T_{23}, T_{25}, T_{57}\}$$
$$T \in M_{T^3}(01234567) \Rightarrow Q_{3^3}(T) = \{<<<, <<=, <=<, <==, =<<, ==<, =<=, ===\}$$

11 Appendix D: selected loading plans with T^3-based ρ_3

Problem 1
Container: 118 × 99 × 99
Box: 36 × 33 × 26

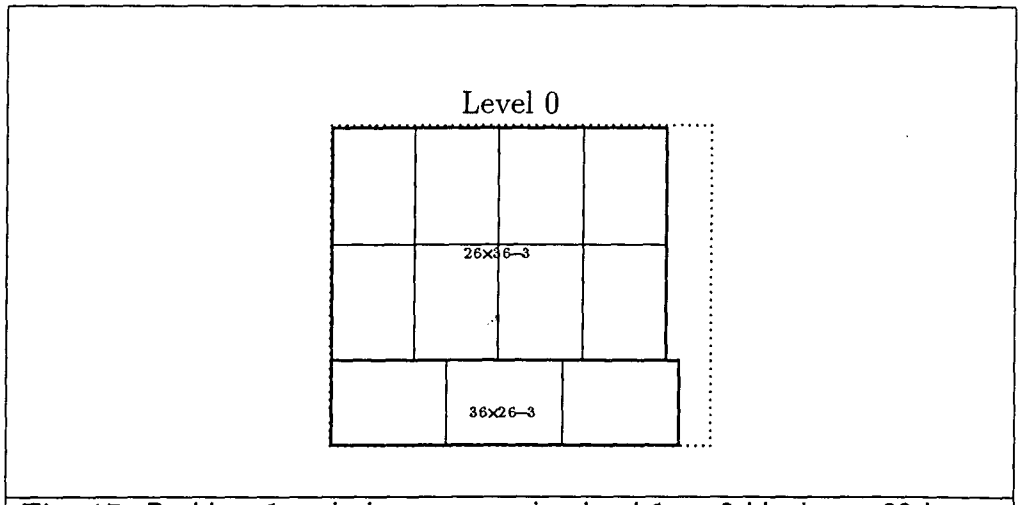


Fig. 17: Problem 1: solution at recursion level 1 — 2 blocks — 33 boxes

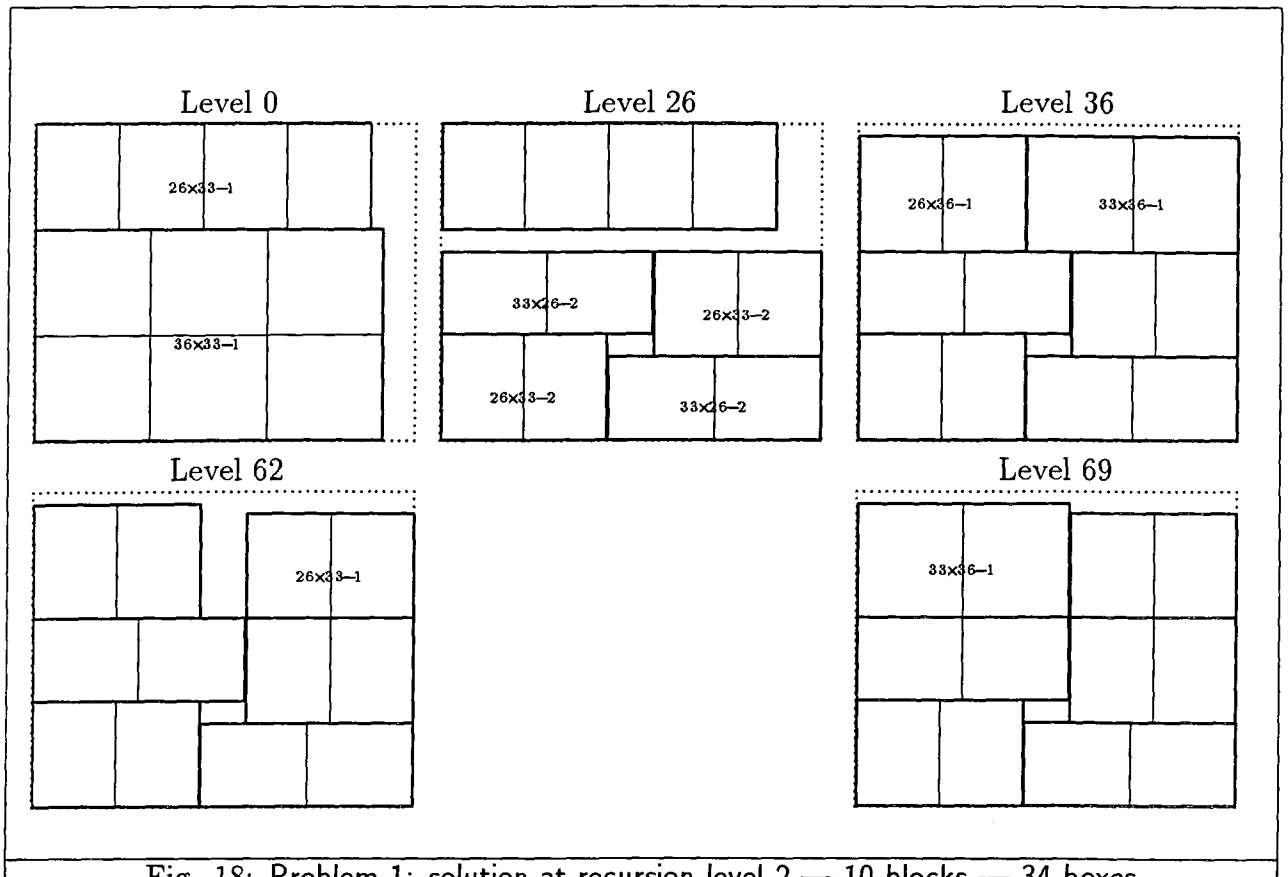


Fig. 18: Problem 1: solution at recursion level 2 — 10 blocks — 34 boxes

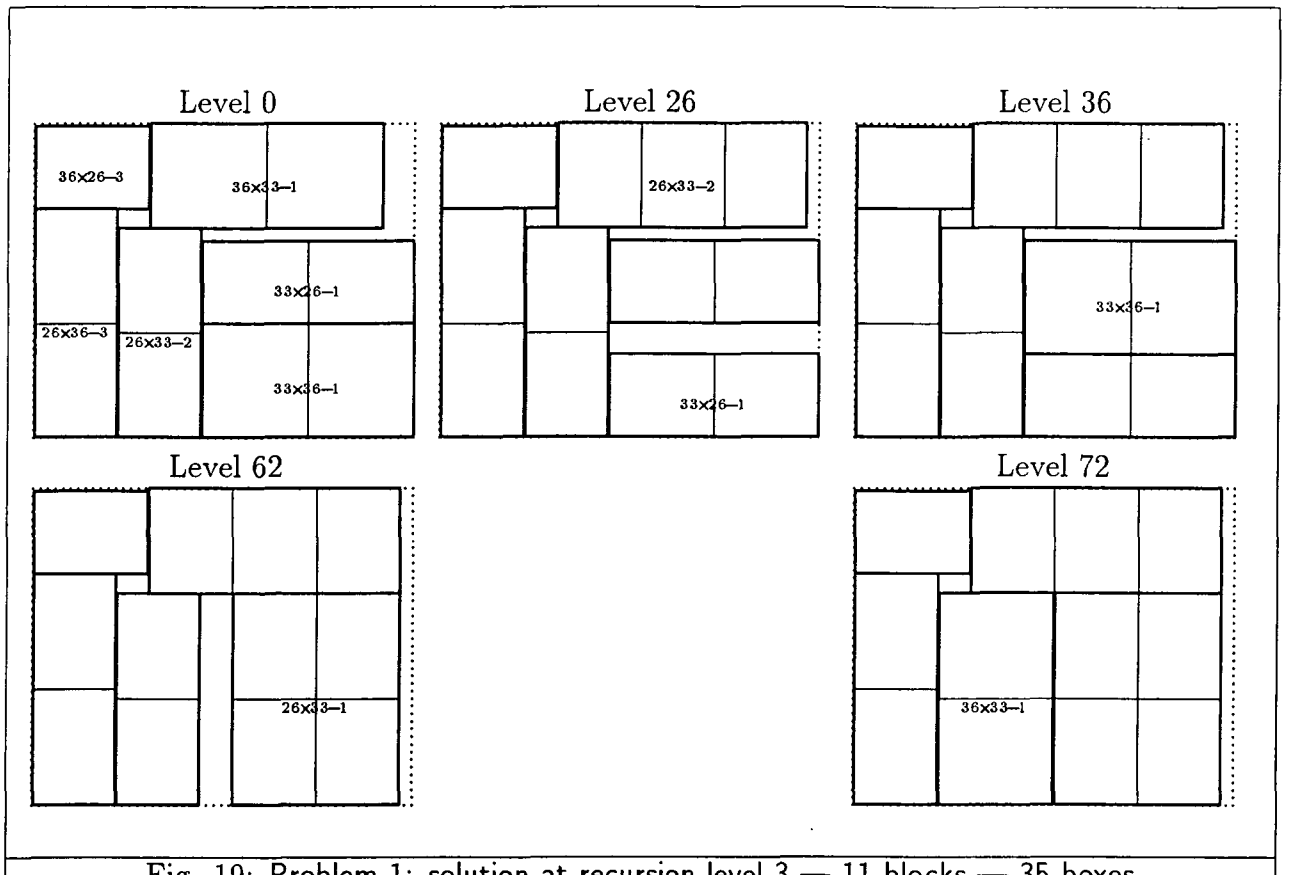


Fig. 19: Problem 1: solution at recursion level 3 — 11 blocks — 35 boxes

Problem 2

Container: 120 × 100 × 100

Box: 36 × 31 × 28

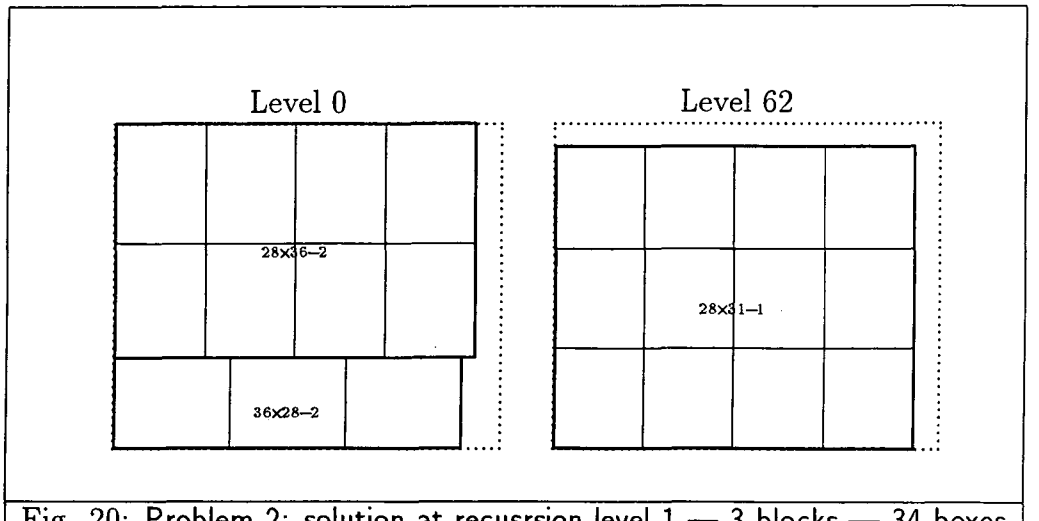


Fig. 20: Problem 2: solution at recusrsion level 1 — 3 blocks — 34 boxes

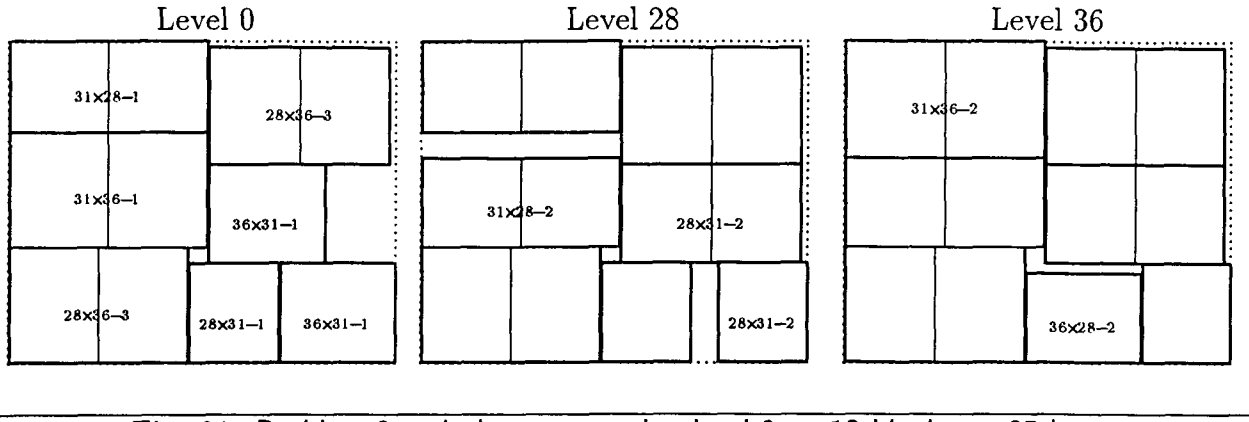


Fig. 21: Problem 2: solution at recursion level 2 — 12 blocks — 35 boxes

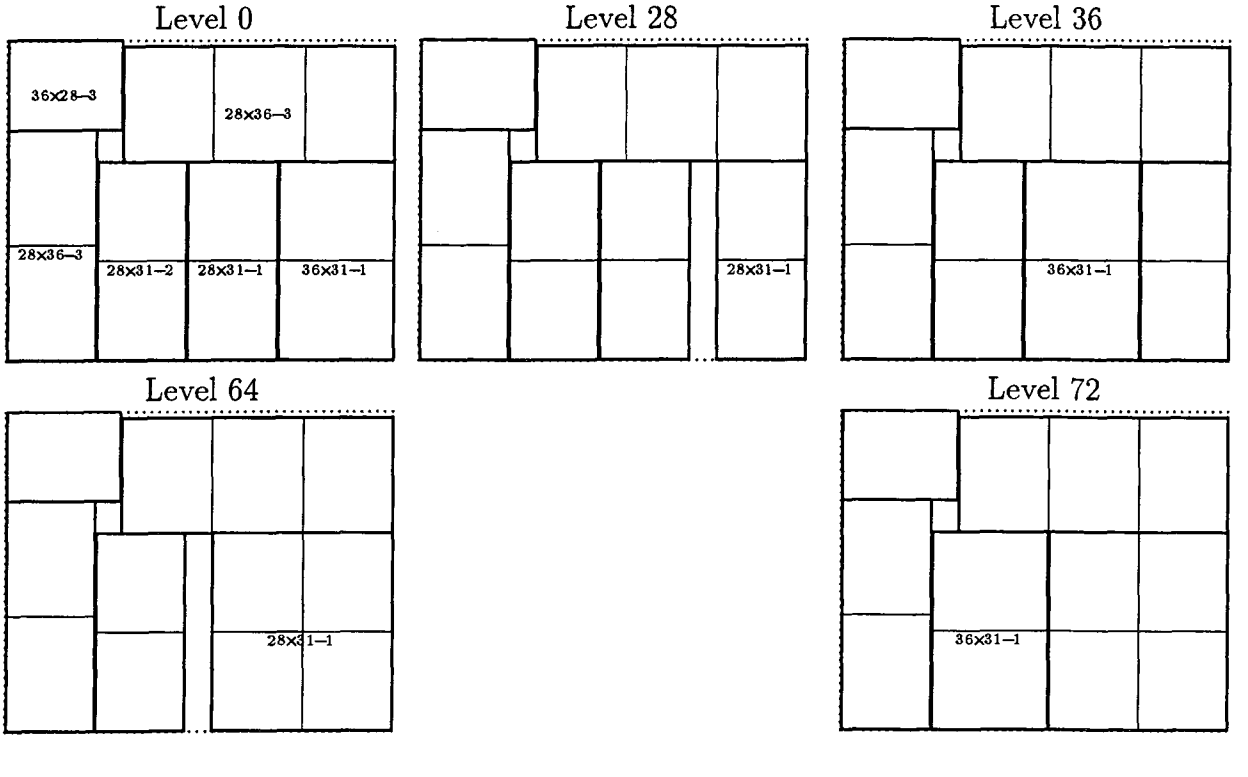


Fig. 22: Problem 2: solution at recursion level 3 — 10 blocks — 36 boxes

Problem 3

Container: 120 × 100 × 100

Box: 30 × 30 × 20

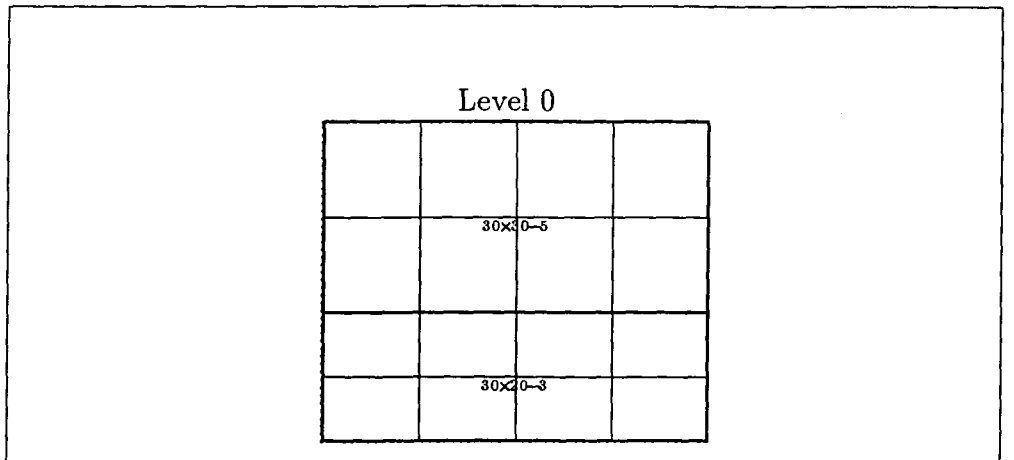


Fig. 23 Problem 3: solution at recursion level 1 — 2 blocks — 64 boxes

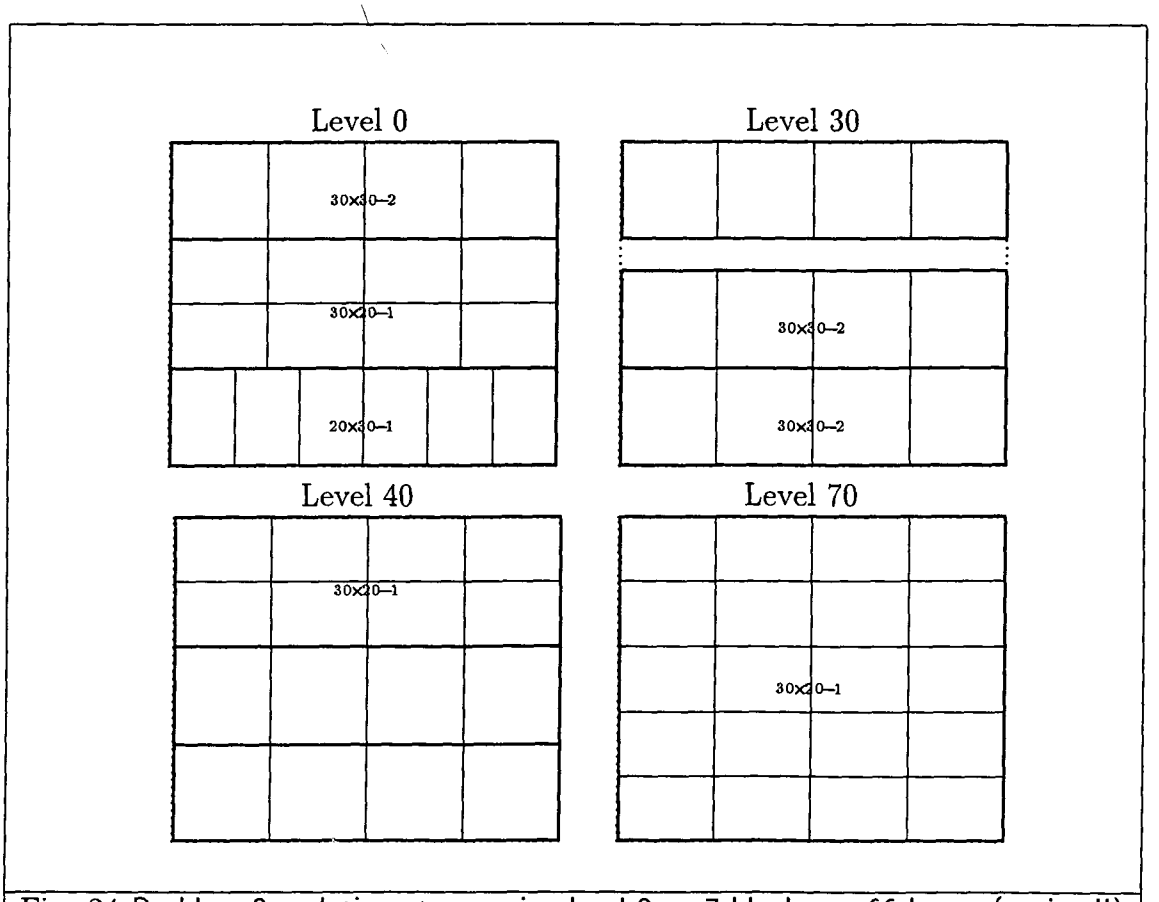


Fig. 24 Problem 3: solution at recursion level 2 — 7 blocks — 66 boxes (optimal!)

Problem 4

Container: 120 × 99 × 99

Box: 36 × 31 × 28

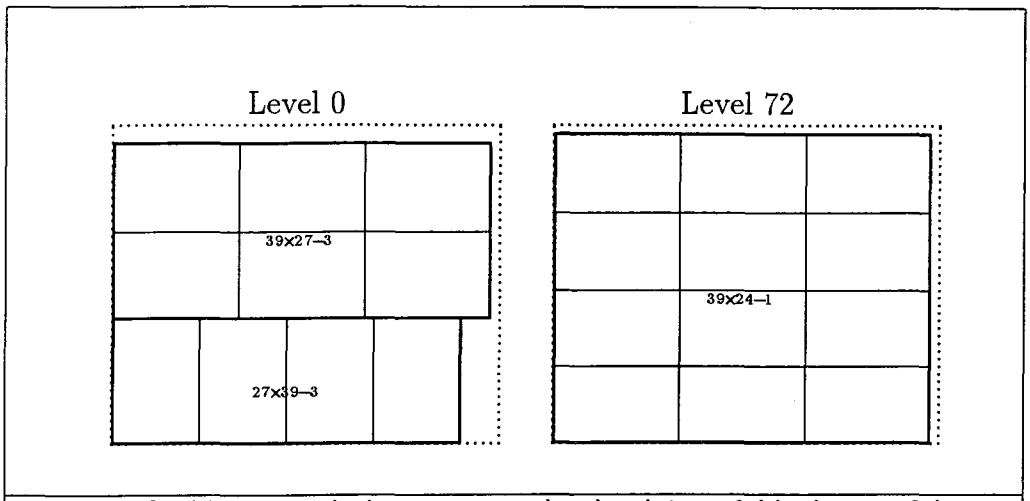


Fig. 25: Problem 4: solution at recusrsion level 1 — 3 blocks — 42 boxes

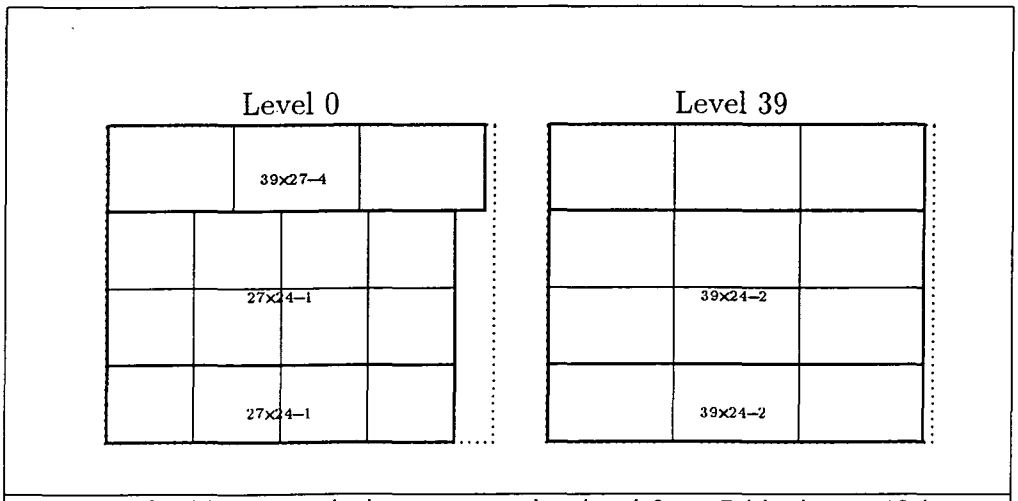


Fig. 26: Problem 4: solution at recursion level 2 — 5 blocks — 42 boxes

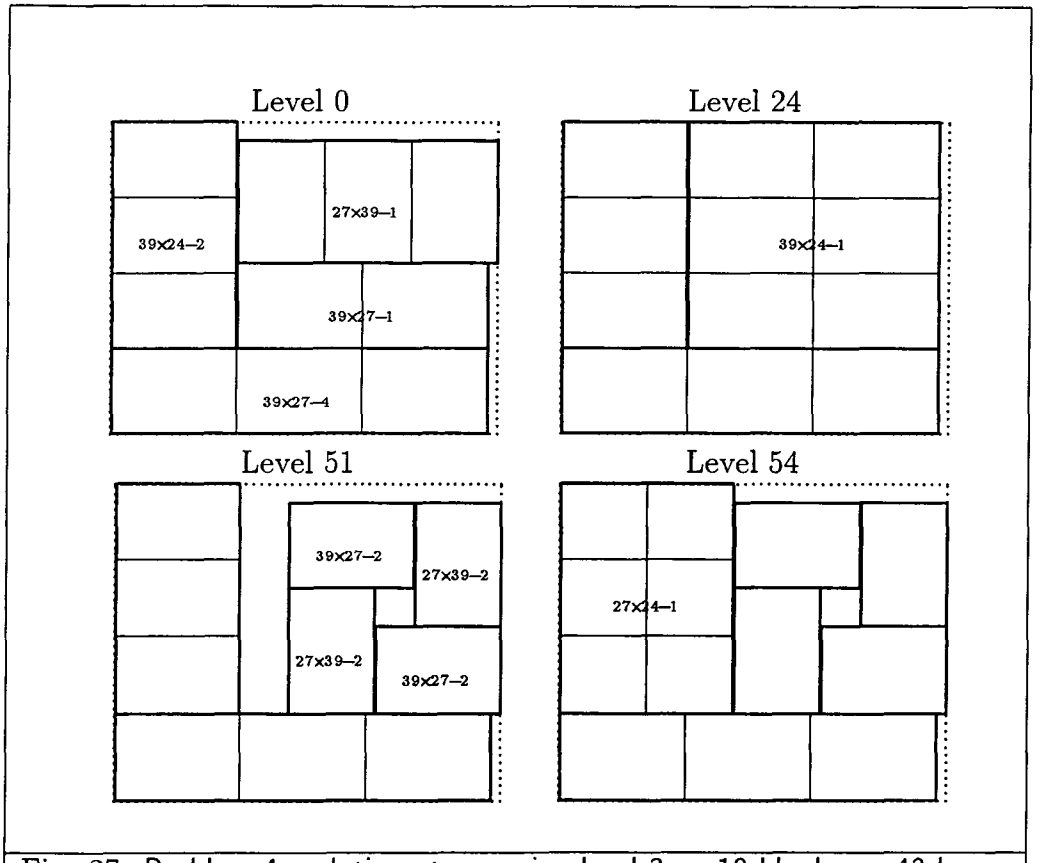


Fig. 27: Problem 4: solution at recursion level 3 — 10 blocks — 43 boxes

Problem 5

Container: 120 × 100 × 100

Box: 32 × 22 × 18

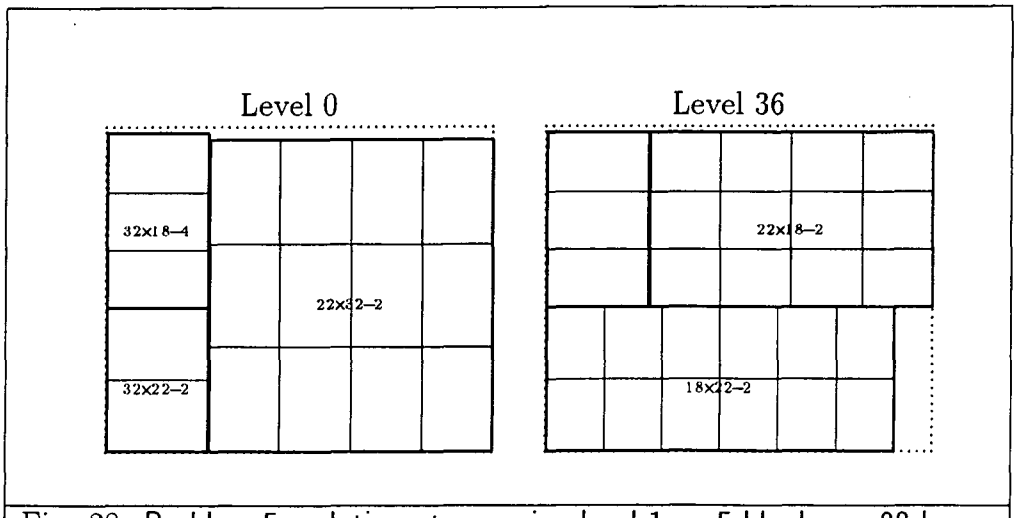


Fig. 28: Problem 5: solution at recursion level 1 — 5 blocks — 88 boxes

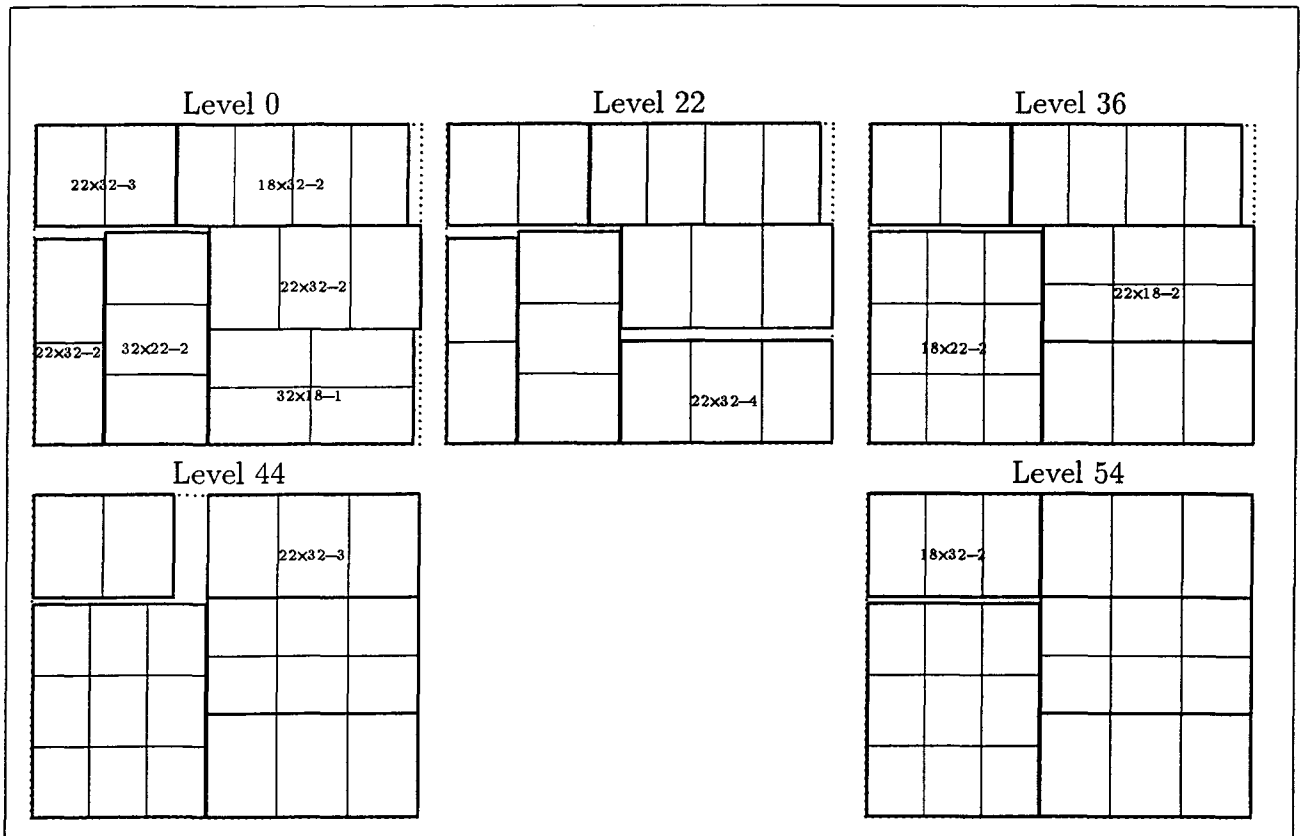


Fig. 29: Problem 5: solution at recursion level 2 — 11 blocks — 91 boxes

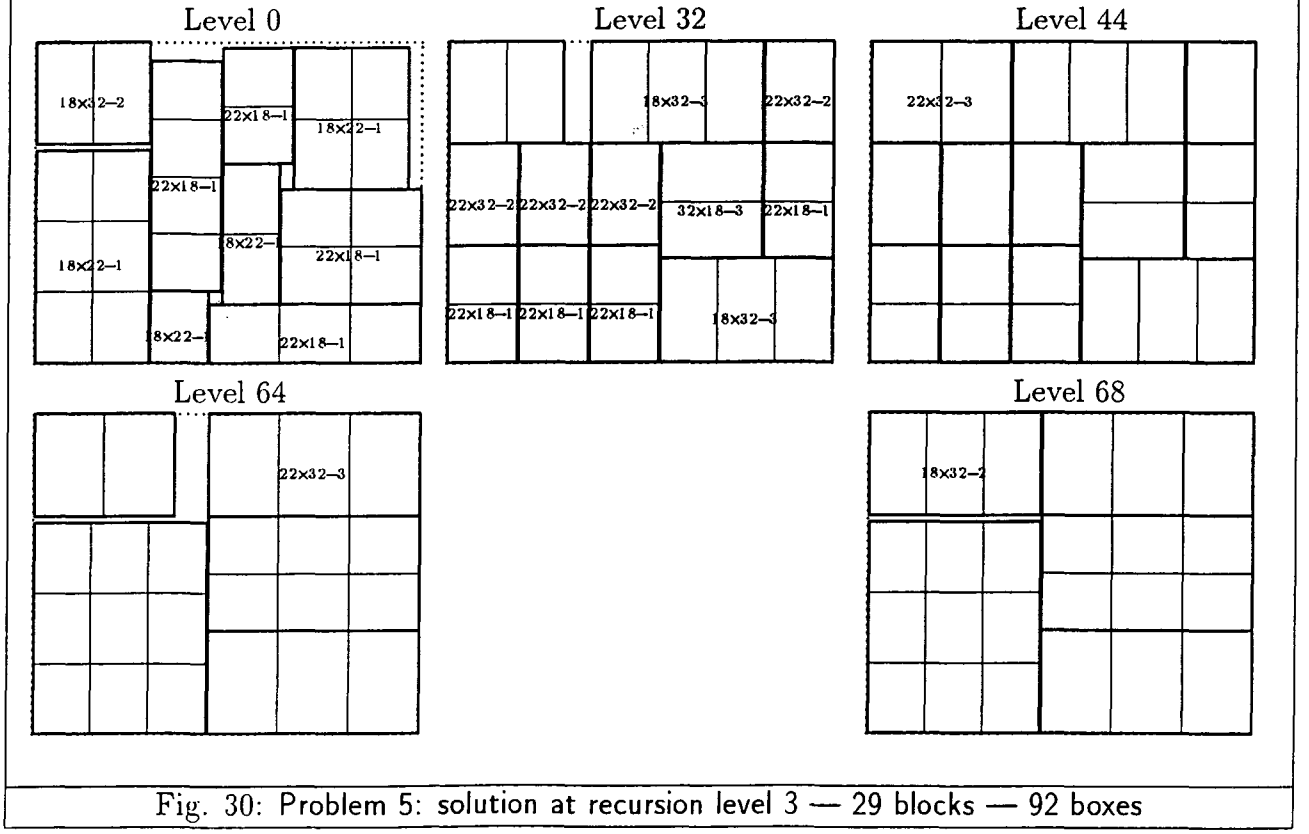


Fig. 30: Problem 5: solution at recursion level 3 — 29 blocks — 92 boxes

Problem 6

Container: 120 × 100 × 100

Box: 30 × 25 × 21

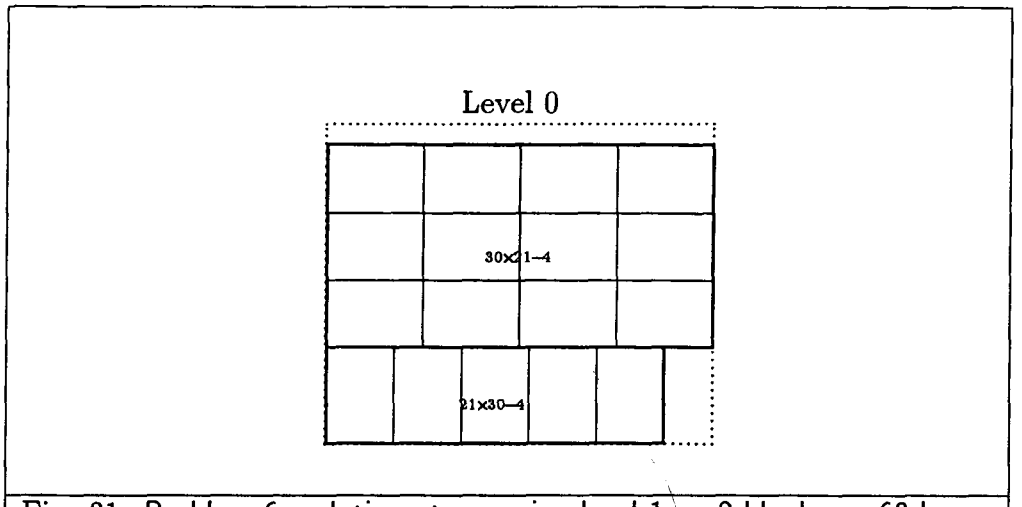


Fig. 31: Problem 6: solution at recursion level 1 — 2 blocks — 68 boxes

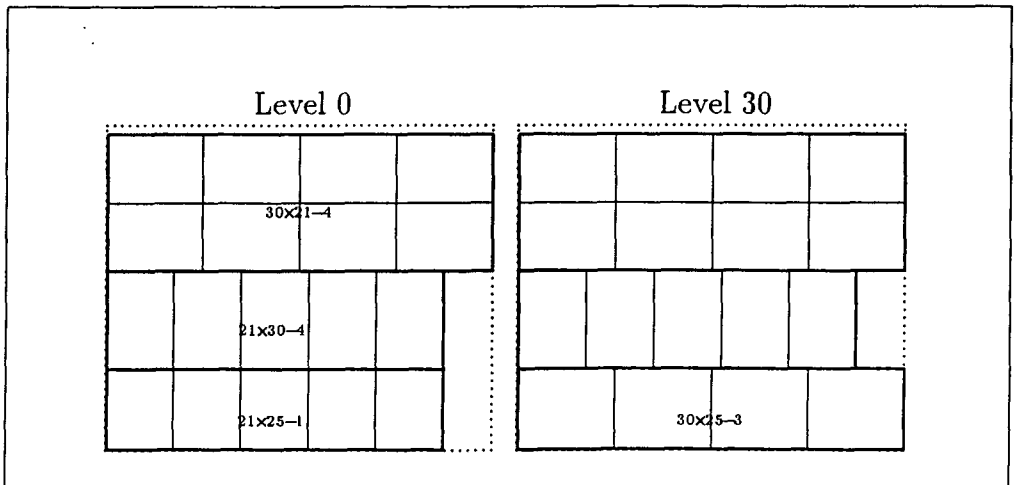


Fig. 32: Problem 6: solution at recursion level 2 — 4 blocks — 69 boxes

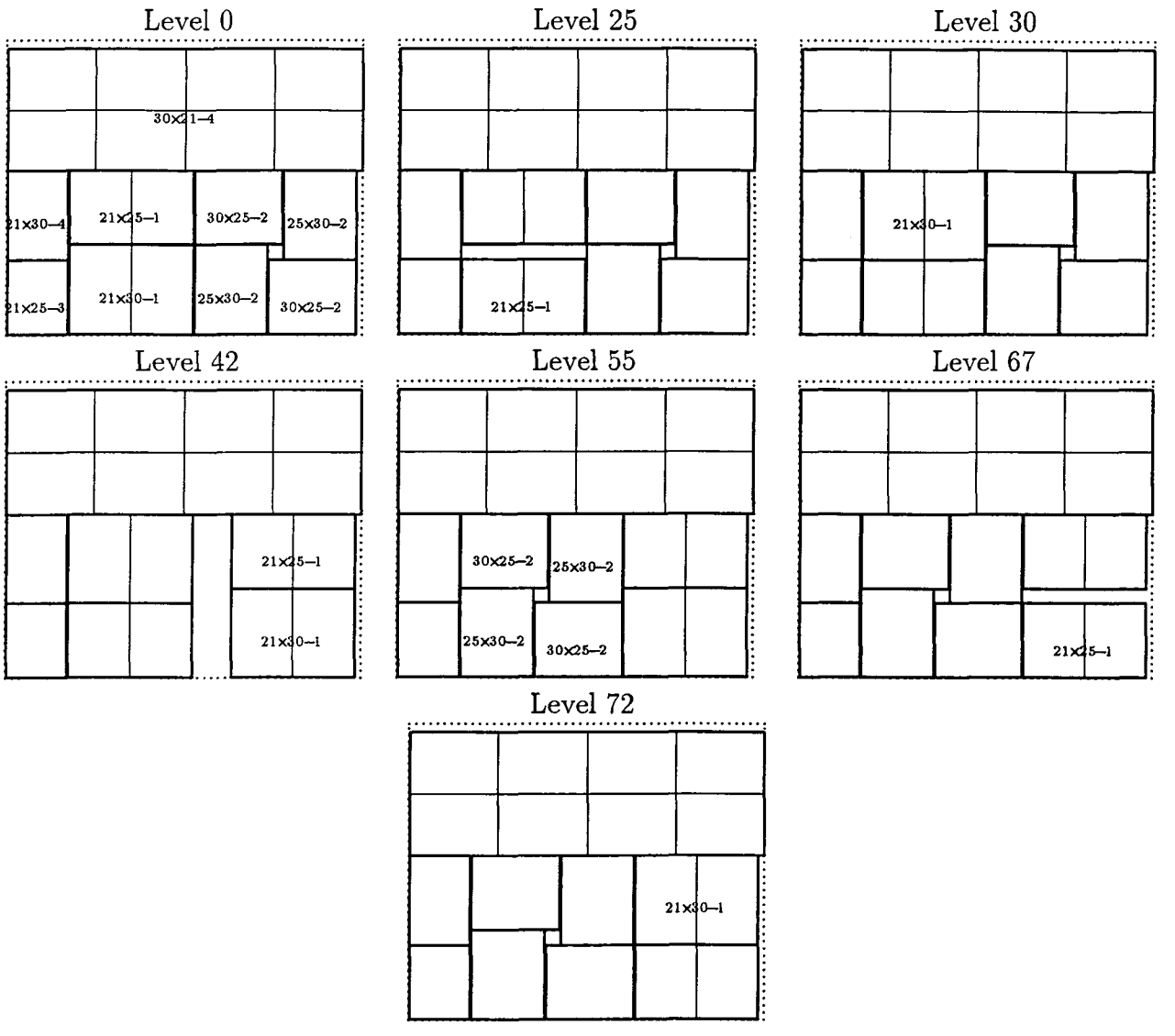


Fig. 33: Problem 6: solution at recursion level 3 — 19 blocks — 71 boxes

NOTAS DO ICMC

SÉRIE COMPUTAÇÃO

047/2000 OLIVEIRA JR., O. N.; MARCHI, A.R.; MARTINS, M.S.; MARTINS, R.T.; NUNES, M.G.V. – A critical analysis of the performance of english-portuguese-english MT systems.

046/2000 SOUZA, P.S.L.; SANTANA, M.J.; SANTANA, R.H.C. – Escalonamento de processos: uma contribuição para a convergência da área.

045/2000 MARTINS, R.T.; RINO, L.H.M.; NUNES, M.G.V.; MONTILHA, G.; OLIVEIRA JR., O. N. – An interlingua diming at communication on the Web: how language-independent can it be?

044/99 FELTRIM, V.D.; FORTES,R.P.M. – Evaluation of a reverse engineering method through the application in a hypermedia system.

043/99 PANSANATO, L.T.E.; NUNES, M.G.V. – EHDM: Método para projeto de hiperdocumentos para ensino.

042/99 MORABITO, R.; ARENALES, M. – Optimizing the cutting of stock plates in a furniture company.

041/98 SANTOS-MEZA, E.; SANTOS, M.O.; ARENALES, M.N. – Lot sizing and scheduling in na automated foundry.

040/98 FELTRIM, V.D.; FORTES, R.P.M. – Uma modelagem do domínio de engenharia reversa de software utilizando o método OOHDM.

039/98 FERREIRA, V.G.; MELLO, O .D.; OLIVEIRA, J.N.; FORTUNA, A . O . - Tópicos teóricos e computacionais em escoamentos de fluidos.

038/98 CAVICHIA, M.C.; ARENALES, M.N. - Piecewise linear programming via interior points.