On the Implementation and Assessment of several Divide & Conquer Matheuristic Strategies for the solution of the Knapsack Problem

Fernando A Morales*, Jairo A Martínez*

*Escuela de Matemáticas Universidad Nacional de Colombia, Sede Medellín
Carrera 65 # 59A–110, Bloque 43, of 106, Medellín - Colombia

Abstract

We introduce and assess a Divide & Conquer heuristic method, aimed to solve large instances of the Knapsack Problem. The method subdivides a large problem in two smaller ones (or recursive iterations of the same principle), to lower down the global computational complexity of the original problem, at the expense of a moderate loss of quality in the solution. Theoretical mathematical results are presented in order to guarantee an algorithmically successful application of the method and to suggest the potential strategies for its implementation. In contrast, due to the lack of theoretical results, the solution’s quality deterioration is measured empirically by means of Monte Carlo simulations for several types and values of the chosen strategies. Finally, introducing parameters of efficiency we suggest the best strategies depending on the data input.

Keywords: Divide and Conquer, Knapsack Problem, Monte Carlo simulations, method’s efficiency.

2010 MSC: 90C59, 90C06, 90C10, 65C05, 68U01

1. Introduction

The Knapsack Problem (KP) is, beyond dispute, one of the fundamental problems in integer optimization for three main reasons. First, due to its simplicity with respect to a general linear integer (or mixed integer) optimization problem. Second, because of its occurrence as a subproblem of an overwhelming number of optimization problems, including a wide variety of real life situations which can be modeled by KP. Third, because it belongs to the NP hard class problems which makes it relevant from the theoretical perspective. As a natural consequence, there is a vast literature dedicated to the KP solution, comprising a broad spectrum between exact algorithms such as Dynamic Programming (DP) and Branch & Bound (B&B) techniques [8], metaheuristic schemes such as Genetic Algorithms (GA), Ant Colony Algorithms (ACO’s) and hybrid algorithms including matheuristics and symheuristics [14, 5, 4, 12, 7]; an early review of non-standard versions of KP is found in [9], a detailed review of some versions is found in the texts [10, 8]. As with any optimization problem, for the KP solution it is crucial to exploit the trade-off between the quality of the solution in terms of the value of the objective function, and the computational effort required to obtain it.

Both exact methods and metaheuristic algorithms have disadvantages. Exact algorithms such as DP and B&B usually are insufficient to address large instances: all dynamic programming versions for KP are pseudo-polynomial, i.e. time and memory requirements are dependent on instance size (N, D). Commonly the computational complexity of the algorithms B&B cannot be explicitly described, as it is not possible to estimate a priori the number of search tree nodes required (see [8, 11]). On the other hand, most metaheuristics lack sufficient theoretical
justification. Despite the widespread success of such techniques, among researchers there is little understanding of the key aspects of their design, including the identification of search space characteristics that influence the difficulty of the problem. There are some theoretical results related to the convergence of algorithms under appropriate probabilistic hypotheses, however these are not useful from the practical point of view. Moreover, it is not possible to argue that any of the particular metaheuristics is on average superior to any other, so the choice of a metaheuristic to address a specific optimization problem depends largely on the user’s experience [12].

As a consequence of the KP’s relevance, it is natural that any proposed method for solving integer optimization problems: theoretical, empirical or mixed, is usually first tested on a Knapsack Problem. This is the case of the present work, where we introduce a Divide and Conquer (D&C) strategy aimed to solved large instances of the Knapsack Problem. More specifically, we focus on a particular variant of KP: the problem of minimizing the number of proctors needed in a massive test, with several rooms of different capacities and different proctoring personnel needs (see Problem 1 Section 2.1 below). Such a problem arises naturally at those universities offering large coordinated lower division courses with multiple sections and/or those ones, administrating admission tests to a vast number of candidates. Furthermore, solving the first example was the starting point of the present paper.

The main goal of the proposed approach is to reduce the computational complexity of the KP by subdividing the original/initial problem in two smaller subproblems, at the price of giving up (to some extent) quality of the solution. Moreover, using multiple recursive D&C iterations the initial problem can be decomposed on several subproblems of suitable size (at the price of further deterioration in the solution’s quality), in a multilevel scheme, see Figures 1 2 3. The multilevel paradigm is not a metaheuristic in itself, on the contrary, it must act in collaboration with some solution strategy, be it an exact or approximate procedure. For the method to be worthy, the loss of quality must lie within an acceptable range. Consequently, the present work first introduces the technique, together with several strategies for its implementation. Next, the quality is defined using several parameters of efficiency. Finally, since no theoretical results can be mathematically shown for measuring the efficiency of the method, we proceed empirically using Monte Carlo simulations and the Law of Large Numbers (see Theorem 8 below) to determine which strategy will likely be the best, if the data input of the problem are regarded as random variables with known probabilistic distribution.

We close this section mentioning that different authors have reported the increased performance of metaheuristic techniques when used in conjunction with a multilevel scheme on large instances. The multilevel paradigm has been used mainly in mesh construction, Graph Partition Problem (GPP), Capacitated Multicommodity Network Design (CMND), Covering Design (CD), Graph Colouring (GC), Graph Ordering (GO), Traveling Salesman Problem (TSP) and Vehicle Routing Problem (VRP) [13], [1]. To the Authors’ best knowledge, the use of a multilevel D&C scheme for solving Knapsack Problem has not been reported.

2. Preliminaries

In this section the general setting and preliminaries of the problem are presented. We start introducing the mathematical notation. For any natural number $N \in \mathbb{N}$, the symbol $[N] \overset{\text{def}}{=} \{1, 2, \ldots, N\}$ indicates the set/window of the first $N$ natural numbers. For any set $E$ we denote by $|E|$ its cardinal and $\mathcal{P}(E)$ its power set. A particularly important set is $S_N$, where $S_N$ denotes the set of all permutations in $[N]$, its elements will be usually denoted by $\pi, \sigma, \tau$, etc. Random variables will be represented with upright capital letters, namely $X, Y, Z, ...$ and its respective expectations with $\mathbb{E}(X), \mathbb{E}(Y), \mathbb{E}(Z), ...$. Vectors are indicated with bold letters, namely $\mathbf{p}, \mathbf{g}, ...$ etc. Particularly important collections of objects will be written with calligraphic characters, e.g. $\mathcal{A}, \mathcal{D}, \mathcal{E}$ to add emphasis. For any real number $x \in \mathbb{R}$ the floor and ceiling function are given (and denoted) by $\lfloor x \rfloor \overset{\text{def}}{=} \max\{z : z \leq x, z \text{ integer}\}$, $\lceil x \rceil \overset{\text{def}}{=} \max\{z : z \geq x, z \text{ integer}\}$, respectively.
2.1. The Problem

Now we introduce the variant of the Knapsack problem to be studied. Consider a list of $N$ rooms together with its respective capacities $c \overset{\text{def}}{=} (c_i : i \in [N])$ and needed proctors i.e., $p \overset{\text{def}}{=} (p_i : i \in [N])$. We analyze the problem of choosing the rooms so that the demand of students $D$ is satisfied, but minimizing the number of proctors needed to run the test. This can be summarized in the following integer programming problem

**Problem 1.**

$$\min \sum_{i \in [N]} p_i x_i , \quad (1a)$$

subject to

$$\sum_{i \in [N]} c_i x_i \geq D, \quad (1b)$$

$$x_i \in \{0, 1\}, \quad \text{for all } i \in [N]. \quad (1c)$$

Here, $x \overset{\text{def}}{=} (x_i : i \in [N])$ is the list of binary valued decision variables. In the sequel, the feasible set is denoted by

$$S \overset{\text{def}}{=} \{x \in \{0, 1\}^N : c \cdot x \geq D\} \quad (2)$$

and the problem can be written concisely as

$$\min \{p \cdot x : x \in S\}. \quad (3)$$

In this particular problem the capacity coefficients $(c_i : i \in [N])$ as well as the costs $(p_i : i \in [N])$ (number of proctors), are all positive integers. Observe that the solution of Problem 1 above can be found using the solution of the following Knapsack Problem

**Problem 2.**

$$\max \sum_{i \in [N]} p_i \xi_i , \quad (4a)$$

subject to

$$\sum_{i \in [N]} c_i \xi_i \leq \sum_{i \in [N]} c_i - D, \quad (4b)$$

$$\xi_i \in \{0, 1\}, \quad \text{for all } i \in [N]. \quad (4c)$$

**Proposition 1.** Let $\xi = (\xi_i : i \in [N]) \in \{0, 1\}^N$ be a solution to Problem 2 and define $x_i \overset{\text{def}}{=} 1 - \xi_i$ for all $i \in [N]$ then, the vector $x = (x_i : i \in [N]) \in \{0, 1\}^N$ is a solution to Problem 1.

**Proof.** First notice that $x_i = 1 - \xi_i \in \{0, 1\}$ because $\xi_i \in \{0, 1\}$ for all $i \in [N]$. Denote by $f(\xi) \overset{\text{def}}{=} \sum_{i \in [N]} p_i \xi_i$ the objective function of Problem 2 and define the function

$$g(\xi) \overset{\text{def}}{=} \sum_{i \in [N]} p_i - f(\xi) = \sum_{i \in [N]} p_i (1 - \xi_i).$$

It is clear that the minimum of $g$ occurs at the maximum of $f$ and recalling change of variable define de objective function $h(\xi) \overset{\text{def}}{=} \sum_{i \in [N]} p_i x_i$ i.e., the objective function (1a) of Problem 1. Next, consider the following
equivalences of the capacity constraint (4b)
\[
\sum_{i \in [N]} c_i \xi_i \leq \sum_{i \in [N]} c_i - D \quad \Leftrightarrow \\
- \sum_{i \in [N]} c_i \xi_i \geq - \sum_{i \in [N]} c_i + D \quad \Leftrightarrow \\
\sum_{i \in [N]} c_i (1 - \xi_i) \geq D \quad \Leftrightarrow \\
\sum_{i \in [N]} c_i x_i \geq D.
\]

In the expression above, the last equivalence holds merely recalling the definition of \(x\). Consequently the capacity constraint (4b) of Problem 2 is equivalent to the constraint capacity (1b), which completes the proof. \(\square\)

2.2. Greedy Algorithm vs Linear Optimization Relaxation

In this section, we explore the relationship between the solution of the natural linear relaxation of Problem 1 and the solution provided by the natural Greedy Algorithm. First we introduce the following definitions

Definition 1. Let \(c = (c_i : i \in [N]), p = (p_i : i \in [N])\) be the lists of capacities and prices (or needed proctors) respectively, we define the list of specific weights by
\[
\gamma_i \overset{\text{def}}{=} \frac{c_i}{p_i}, \quad \text{for all } i \in [N]. \tag{5}
\]
Consider now the Greedy Algorithm 1 to find a feasible solution to Problem 1.

Algorithm 1 Greedy Algorithm, returns feasible solution to Problem 1 \((y_i : i \in [N])\) and corresponding value of objective function \(\sum_i p_i y_i\)

```
1: procedure Greedy Algorithm (Prices: \(p = (p_i : i \in [N]),\) Capacities: \(c = (c_i : i \in [N]),\) Demand: \(D\))
2: if \(\sum_{i \in [N]} c_i < D\) then print “Feasible region is empty” \(\triangleright\) Checking if the problem is infeasible
3: else
4: \hspace{1em} compute list of specific weights \((\gamma_i : i \in [N])\) \(\triangleright\) Introduced in Definition 1
5: \hspace{1em} sort the list \((\gamma_i : i \in [N])\) in descending order
6: \hspace{1em} denote by \(\sigma \in S[N]\) the associated ordering permutation, i.e.,
7: \hspace{2em} \(\gamma_{\sigma(i)} \geq \gamma_{\sigma(i+1)}\), \quad \text{for all } i \in [N-1]. \tag{6}
8: \hspace{1em} \(y_i = 0\) for all \(i \in [N]\), \(\text{capacity} = 0\) \(\triangleright\) Initializing feasible solution and capacity
9: \hspace{1em} while \(\text{capacity} \geq D\) do \(y_{\sigma(i)} = 1, \quad \text{capacity} = \text{capacity} + c_{\sigma(i)}\)
10: \hspace{1em} return \((y_i : i \in [N]), \sum_{i \in [N]} p_i y_i) \quad \triangleright\) Feasible solution and corresponding value of objective function
11: end if
12: end procedure
```

Due to the loop starting condition \(\sum_{i \in [N]} c_i \geq D\), whenever the loop starts, it will stop a finite number of iterations. Next we introduce

Definition 2. The natural linear relaxation of Problem 1 is given by
Problem 3.

\[
\min \sum_{i \in [N]} p_i \xi_i, \quad (7a)
\]

subject to

\[
\sum_{i \in [N]} c_i \xi_i \geq D, \quad (7b)
\]

\[
0 \leq \xi_i \leq 1, \quad \text{for all } i \in [N]. \quad (7c)
\]

i.e., the decision variables \((\xi_i : i \in [N])\) are now real-valued.

Next we introduce a convenient notation

Definition 3. Let \(\xi = (\xi_i : i \in [N])\) be a solution of Problem 3 define the index sets

\[
P \overset{\text{def}}{=} \{ i \in [N] : \xi_i > 0 \}, \quad Z \overset{\text{def}}{=} \{ i \in [N] : \xi_i = 0 \}. \quad (8)
\]

Define its associated integer solution \(x^\xi = (x^\xi_i : i \in [N])\) by

\[
x^\xi_i \overset{\text{def}}{=} \begin{cases} 1 & i \in P, \\ 0 & i \in Z. \end{cases} \quad (9)
\]

Theorem 2. Let \(\xi = (\xi_i : i \in [N])\) be a solution of Problem 3 and let \(x^\xi\) be as in Definition 3 above. Then, \(x^\xi = (x^\xi_i : i \in [N])\) is the solution furnished by the Greedy Algorithm 1.

Proof. Let \((\xi_i : i \in [N])\) be a solution of Problem 3 furnished by the Simplex Algorithm and let \((\gamma_i : i \in [N])\) be the list of specific weights as introduced in Definition 1. We claim that

\[
\gamma_k \geq \gamma_\ell, \quad \text{for all } k \in P \text{ and } \ell \in Z. \quad (10)
\]

We proceed by way of contradiction. Let \(k \in P\) be fixed and suppose there exists \(\ell \in Z\) such that \(\gamma_k < \gamma_\ell\). If the constraint \((7b)\) were not active, it would be possible to lower the values \(\xi_i\) for \(i \in P\) while still verifying the constraints \((7b)\) and \((7c)\). But this would also lower the value of the objective function and \((\xi_i : i \in [N])\) would not be an optimal solution. Therefore, the constraint \((7b)\) is active, in particular

\[
c_k \xi_k = D - \sum_{i \in P - \{k\}} c_i \xi_i. \quad (11)
\]

Since the values above are positive, there exist values \(y_k \in (0, \xi_k)\) and \(y_\ell \in (0, 1)\) such that

\[
c_k y_k + c_\ell y_\ell = c_k \xi_k = D - \sum_{i \in P - \{k\}} c_i \xi_i. \quad (12)
\]

Observe that \(\gamma_k < \gamma_\ell\) implies that \(\frac{1}{\gamma_k} y_k > \frac{1}{\gamma_\ell} y_\ell > 0\), in particular \(\frac{p_k}{c_k} c_\ell y_\ell > p_\ell y_\ell\) and

\[
\frac{p_k y_k}{c_k} c_\ell y_\ell > p_\ell y_k + p_\ell y_\ell.
\]

Combining the first equality in Equation \((12)\) with the expression above, it follows that \(p_k \xi_k > p_k y_k + p_\ell y_\ell\), and

\[
\sum_{i \in [N]} p_i \xi_i = p_k \xi_k + \sum_{i \in P - \{k\}} p_i \xi_i > p_k y_k + p_\ell y_\ell + \sum_{i \in P - \{k\}} p_i \xi_i.
\]
But then, the sequence \( \eta_i : i \in [N] \) defined by

\[
\eta_i \overset{\text{def}}{=} \begin{cases} 
    \xi_i, & i \in [N] - \{k, \ell\}, \\
    y_k, & i = k, \\
    y_\ell, & i = \ell,
\end{cases}
\]

satisfies the constraints (7b), (7c) and

\[
\sum_{i \in [N]} p_i \eta_i < \sum_{i \in [N]} p_i \xi_i.
\]

Therefore \( (\xi_i : i \in [N]) \) would not be optimal, which is a contradiction. Finally, due to Statement (10), it is clear that the indexes of the non-null decision variables contained in \( P \), furnished by the Simplex Algorithm have the first \(|P|\)-indexes of the list \( (\gamma_i : i \in [N]) \) when sorted in descending order. Recalling (9), the integral constraint (1c) and the capacity constraint (1b) is satisfied because \( \xi_i \leq 1 = x_i^\xi \) for all \( i \in P \), moreover, the capacity constraint is not satisfied for any proper subset of \( P \). For if \( \sum_{i \in Q} c_i x_i^\xi \geq D \) with \( Q \subseteq P \), the sequence \( (\eta_i : i \in [N]) \) defined by

\[
\eta_i \overset{\text{def}}{=} \begin{cases} 
    x_i^\xi, & i \in Q, \\
    0, & \text{otherwise},
\end{cases}
\]

satisfies \( \sum_{i \in [N]} p_i \eta_i < \sum_{i \in [N]} p_i \xi_i \) and \( (\xi_i : i \in [N]) \) would not be optimal, which is a contradiction. Consequently, the Greedy Algorithm \( 1 \) chooses exactly the first \(|P|\)-indexes of the list \( (\gamma_i : i \in [N]) \) when sorted in descending order which concludes the proof. \( \Box \)

Remark 1. It is important to stress that the Greedy Algorithm \( 1 \) may not produce an optimal solution as the following example shows. Consider Problem \( 1 \) for \( D = 40 \) and the following data.

| Room | Capacity | Proctors | Specific Weigh: \( \gamma \) |
|------|----------|----------|------------------|
| 1    | 100      | 4        | 25               |
| 2    | 40       | 2        | 20               |

Table 1: Remark 1 Data

Clearly, the Greedy Algorithm \( 1 \) would choose the solution \( x = (1, 0) \) with \( p \cdot x = 4 \) while \( y = (0, 1) \) gives \( p \cdot y = 2 \) and \( x \) is not optimal. Moreover, the linear relaxation of this problem would yield \( x^* = (0.4, 0) \) with \( p \cdot x^* = 1.6 \) and associated integer solution \( x^\xi = (1, 0) \), i.e., the solution produced by the Greedy Algorithm \( 1 \).

2.3. Introducing a Student-Proctor Rate

In the sequel we adopt a relationship between capacities \( c \) and proctors \( p \) as it is usually done.

Definition 4 (Rate Proctors Students). Let \( r \in [1, \max_i c_i] \) be the maximum number of students proctored by a single person, then

\[
p_i \overset{\text{def}}{=} \left\lceil \frac{c_i}{r} \right\rceil, \quad \text{for all } i \in [N].
\]

In the following, we refer to \( r \) as the student-proctor rate.

Before proving the main result of this part (Theorem 4) we need an intermediate result.

Proposition 3. Let \( (\xi_i : i \in [N]) \) be an optimal solution to Problem 3 produced by the Simplex Algorithm, and let \( I \overset{\text{def}}{=} \{ i \in [N] : \xi_i \in \mathbb{N} \} \). Then, \(|I| \in \{N - 1, N\} \) i.e., at most one decision variable is not integer.
Proof. Let \((\xi_i : i \in [N])\) be a solution of Problem 3 furnished by the Simplex Algorithm. Recall that this solution must occur on the extremes of the polyhedron defined by the inequalities (7b) and (7c), moreover at least \(N\) of them must be satisfied actively. Consequently, at least \(N - 1\) constraints of the type (7c) are active. If \(N\) of these inequalities are active then \(|I| = N\), if not the inequality (7b) must be active and exactly one decision variable is not an integer. \(\square\)

Remark 2. Observe that the case \(|I| = N\) does not exclude Inequality (7b) been active as the polyhedron could degenerate on that particular extreme.

Theorem 4. Let \((c_i : i \in [N])\) be a given list of room capacities, let the list of prices \((p_i : i \in [N])\) be computed by the map (13) and let \(r\) be the student-proctor rate introduced in Definition 4.

(i) The Greedy Algorithm 1 produces the exact solution for \(r \geq \max_i c_i\).

(ii) Let \(r | c_i\) for all \(i \in [N]\) (i.e., a common divisor of all the capacities). Then, the effectiveness of the Greedy Algorithm 1 is entirely random.

(iii) Let \(r | c_i\) for all \(i \in [N]\) (i.e., a common divisor of all the capacities). Let \(\xi = (\xi_i : i \in [N])\) be a solution of Problem 3 furnished by the Simplex Algorithm and let \(x^\xi\) be as in Definition 3. Then, \(x^\xi\) is a random element of the set

\[ K \overset{\text{def}}{=} \{ x \in S : \sum_{i \in A} c_i x_i < D, \forall A \subseteq [N] : x_i = 1 \} \text{,} \]

where, \(S\) is the set of feasible solutions to Problem 1 introduced in Expression (2).

Proof. (i) Clearly \(p_i = \left\lceil \frac{c_i}{r} \right\rceil = 1\) for all \(i \in [N]\), since \(r \geq \max_i c_i\). Therefore, the objective function (1a) in Problem 1 reduces to

\[ \min_{x \in S} \sum_{i \in [N]} x_i, \]

with \(S\) the feasible set introduced in Expression (2). Then, the minimum is attained when the cardinal \(|\{i \in [N] : x_i = 1\}|\) is minimum i.e., the number of occupied rooms has to be minimum. On the other hand, \(\gamma_i = \frac{c_i}{p_i} = c_i\) for all \(i \in [N]\) and then the Greedy Algorithm 1 sorts the rooms in decreasing capacity.

Let \(\sigma \in \mathcal{S}(N)\) be a permutation such that \(c_{\sigma(i)} \geq c_{\sigma(i+1)}\) for all \(i \in [N - 1]\); define

\[ m \overset{\text{def}}{=} \min \{ n \in [N] : \sum_{i \in [n]} c_{\sigma(i)} \geq D \}. \]

Clearly, \(m\) is the minimum possible number of rooms needed to host \(D\) students. To see this, take any \(|I| \subseteq [N]\) such that \(|I| \leq m\) then sorting the list \((c_i : i \in I)\) decreasingly it can be seen that

\[ \sum_{i \in I} c_i \leq \sum_{i = 1}^{\lfloor |I| / m \rfloor} c_{\sigma(i)} \leq D, \]

where the last inequality holds by the definition of \(m\). Finally, recalling that the Greedy Algorithm furnishes the solution

\[ x_i = \begin{cases} 1, & \sigma(i) \leq m, \\ 0, & \sigma(i) > m, \end{cases} \]

the proof follows.
(ii) Notice that if \( r \) is a common divisor of \( \{ c_i : i \in [N] \} \), then
\[
\gamma_i = \frac{c_i}{p_i} = \frac{c_i}{c_i/r} = r, \quad \text{for all } i \in [N].
\]

Given that all the specific weights are equal, the sorting in the Greedy Algorithm [1] produces a list randomly ordered and consequently, the effectiveness of the algorithm is entirely random.

(iii) Observe that if \( r \mid c_i \) for all \( i \in [N] \), Problem 3 becomes
\[
\min \left\{ \frac{1}{r} \sum_{i \in [N]} c_i \xi_i : \xi \in [0, 1]^N, \quad \sum_{i \in [N]} c_i \xi_i \geq D \right\} = \frac{1}{r} D.
\]

A solution \( \xi \) of this problem must be active on the constraint (7b), i.e.
\[
\sum_{i \in A} c_i \xi_i < D, \quad \forall A \subseteq \{ i \in [N] : \xi_i = 1 \},
\]
consequently \( x^\xi \in K \). On the other hand, given \( x \in K \) arbitrary, we are to prove that \( x = x^\xi \) for some \( \xi \) solution furnished by the simplex algorithm. Due to Proposition 3, \( \xi \) should satisfy actively the constraint (7b) and have at most one fractional entry. Choose any entry in the set \( P(x) \overset{\text{def}}{=} \{ i \in [N] : x_i = 1 \} \), namely \( J \). Since \( x \in K \), it must hold that \( \sum_{i \in P(x) \setminus \{ J \}} c_i x_i < D \) and \( \sum_{i \in P(x)} c_i x_i \geq D \). Define
\[
\xi_i \overset{\text{def}}{=} \begin{cases} x_i, & \text{if } i \neq J, \\ \frac{1}{c_J} \left( D - \sum_{i \in P(x) \setminus \{ J \}} c_i x_i \right), & \text{if } i = J. \end{cases}
\]

Observe that \( \xi \) is optimal since \( \sum_{i \in [N]} p_i \xi_i = \frac{1}{r} D \) and it satisfies the conditions of Proposition 3. Therefore, \( \xi \) is eligible by the Simplex Algorithm and \( x^\xi \) (as in Definition 3) equals \( x \), which completes the proof. □

**Remark 3.** (i) Observe that if \( r \mid c_i \) for all \( i \in [N] \), the Problem 3 becomes
\[
\min \frac{1}{r} \sum_{i \in [N]} c_i x_i, \quad \{ x \in \{0, 1\}^N : \sum_{i \in [N]} c_i x_i \geq D \}.
\]

Hence, it reduces to a problem of approximating and integer from above using an integer partition of \( \sum_i c_i \) in \( N \) blocks.

(ii) Notice that if \( r = \frac{d}{q} \) with \( d \) a common divisor of the room capacities, the conclusion of Theorem 4 part (ii) holds.

(iii) Since the Greedy Algorithm effectiveness becomes entirely random when \( r \) is a common divisor of the capacities, we would like to use another criterion to distinguish the rooms. To this end, the only possibility is to sort them according to its capacities. However, using the capacity as Greed function may not produce the exact solution as the Greedy Algorithm produced for the case \( r \geq \max_i c_i \). Consider the following example
3. A Divide & Conquer Approach

In the present section we introduce the Divide and Conquer method together with some theoretical results to assure the successful implementation of the method, from the algorithmic point of view. We begin with the following definition

**Definition 5 (Divide & Conquer pairs and trees).**

(i) Let \( c = (c_i : i \in [N]) \) and \( p = (p_i : i \in [N]) \) be the data associated to Problem 1. A subproblem of Problem 1 is an integer problem with the following structure

\[
\min \sum_{i \in A} p_i x_i, \quad A \subseteq [N],
\]

subject to

\[
\sum_{i \in [N]} c_i x_i \geq D^A, \quad D^A \leq D,
\]

\[
x_i \in \{0, 1\}, \quad \text{for all } i \in [A].
\]

(ii) Let \((A^0, A^1)\) be a set partition of \([N]\) and let \((D^0, D^1)\) be an integer partition of \(D\) i.e., \(D = D^0 + D^1\). We say a Divide and Conquer instance of Problem 1 is the pair of subproblems \((\Pi^b : b \in \{0, 1\})\), defined by

**Problem 4** \((\Pi^b, b = 0, 1)\).

\[
\min \sum_{i \in A^b} p_i x_i, \quad (15a)
\]

subject to

\[
\sum_{i \in [N]} c_i x_i \geq D^b, \quad (15b)
\]

\[
x_i \in \{0, 1\}, \quad \text{for all } i \in A^b. \quad (15c)
\]

In the sequel, we refer to \((\Pi^b, b = 0, 1)\) as a **D&C pair**. Defining

\[
c^b_i \overset{\text{def}}{=} \begin{cases} c_i, & i \in A^b, \\ 0, & i \notin A^b \end{cases}, \quad p^b_i \overset{\text{def}}{=} \begin{cases} p_i, & i \in A^b, \\ 0, & i \notin A^b \end{cases}
\]

the corresponding feasible sets and the D&C pair can be respectively written

\[
\min \{ p^b \cdot y : y \in S^b \}, \quad \text{with} \quad S^b \overset{\text{def}}{=} \{ y \in \{0, 1\}^N : c^b \cdot y \geq D^b \}. \quad (16)
\]
(iii) A D&C tree (see Figures 1, 2, and 3 below) for Problem 1 is a binary tree satisfying the following
(a) Every vertex of the tree is a subproblem of Problem 1.
(b) The root of the tree is the problem itself.
(c) Every vertex $V$ which is not a leaf has a left and right children, $V_l, V_r$, respectively, such that $(V_l, V_r)$ is a D&C pair for the problem $V$.

Theorem 5. Suppose that Problem 1 is feasible, then
(i) A feasible solution $y$ of Problem 1 can be infeasible for at most one problem of the D&C pair.
(ii) At most one problem of the D&C pair is infeasible.
(iii) Let $(A^b : b \in \{0, 1\})$ be a fixed partition of $[N]$ then, both Problems $\Pi^b (Π^b : b \in \{0, 1\})$ are feasible if and only if
$$D - \sum_{i \in A^{1-b}} c_i \leq D^b \leq \sum_{i \in A^b} c_i,$$
for $b = 0, 1$. (17)

(iv) Let $(A^b : b \in \{0, 1\})$ be a fixed partition of $[N]$ and define
$$D^0 = \left\lfloor \frac{D}{\sum\{c_i : i \in [N]\}} \sum_{i \in A^0} c_i \right\rfloor, \quad D^1 = D - D^0.$$ (18)
Then, if
$$\sum_{i \in [N]} c_i y_i + 1 \leq \sum_{i \in A^1} c_i,$$
both Problems $\Pi^0 (Π^0 : b \in \{0, 1\})$ are feasible.

(v) The following inclusions for the feasible sets $S^0, S^1, S$ hold
$$S \subseteq S^0 \cup S^1, \quad S^0 \cap S^1 \subseteq S.$$ (20)

Proof. (i) Let $y$ be a feasible solution of Problem 1 then $\sum_{i \in [N]} c_i y_i \geq D$; equivalently
$$\sum_{b \in \{0, 1\}} \sum_{i \in A^b} c_i y_i \geq D^0 + D^1.$$

Then, if $y$ is $Π^b$-infeasible we have $\sum_{i \in A^b} c_i y_i < D^b$ and the expression above writes
$$\sum_{i \in A^{1-b}} c_i y_i \geq D^{1-b} + D^b - \sum_{i \in A^b} c_i y_i > D^{1-b},$$
i.e., $y$ is $Π^{1-b}$-feasible. Since $b \in \{0, 1\}$ was arbitrary, the claim of this part follows.

(ii) Since Problem 1 is feasible, the vector $y \in \{0, 1\}^N$ having all its entries equal to one is also feasible, due to the previous part the result follows.

(iii) Fix $b \in \{0, 1\}$ arbitrary, then it is trivial to see that the second inequality in (17) is necessary and sufficient condition for the problem $Π^b$ to be feasible, as well as the condition $D^{1-b} \leq \sum_{i \in A^{1-b}} c_i$ is necessary and sufficient for $Π^{1-b}$ to be feasible. Recalling that $D^b = D - D^{1-b}$, the first inequality in (17) follows.
Proof. (i) Since Problem 1 is feasible then
\[ D^0 = \left\lfloor \frac{\sum_{i \in [N]} c_i - 1}{D} \right\rfloor \sum_{i \in A^0} c_i \leq \sum_{i \in A^0} c_i, \]
i.e., the problem Π^0 is feasible. On the other hand,
\[ D^0 = \left\lfloor \frac{\sum_{i \in [N]} c_i - 1}{D} \right\rfloor \sum_{i \in A^0} c_i \geq \sum_{i \in A^0} c_i - 1. \]
Since \( D^1 = D - D^0 \) we have
\[ D^1 \leq D - \frac{D}{\sum_{i \in [N]} c_i} \sum_{i \in A^0} c_i + 1 \]
\[ = \frac{D}{\sum_{i \in [N]} c_i} \sum_{i \in A^0} c_i + 1 \]
\[ \leq \sum_{i \in A^1} c_i. \]

Where the last bound holds due to Inequality 19. Hence, the problem Π^1 is also feasible.

(v) Due to the first part if \( y \in S \) then it must be Π^0 or Π^1-feasible. Equivalently, it belongs to \( S^0 \cup S^1 \), i.e. \( y \in S \). Finally, if \( y \in S^0 \cap S^1 \) then \( \sum_{i \in A^0} c_i y_i \geq D^b \) for \( b = 0, 1 \). Adding both inequalities yields
\[ \mathbf{c} \cdot \mathbf{y} = \sum_{i \in [N]} c_i y_i = \sum_{i \in A^0} c_i y_i + \sum_{i \in A^1} c_i y_i \geq D^0 + D^1 = D, \]
i.e., \( y \) belongs to the set \( S \) and the proof is complete. \( \square \)

Remark 4. Observe that Inequality 19 in (iv) in Theorem 5 is a mild hypothesis, it is equivalent to
\[ D \leq \sum_{i \in [N]} c_i - 1 \sum_{i \in [N]} c_i, \]i.e., Inequality 19 demands a reasonable slack \( \sum_{i \in [N]} c_i - D \) between total capacity and demand.

Proposition 6. Let \( \mathbf{c} = (c_i : i \in [N]), \mathbf{p} = (p_i : i \in [N]) \) be the data associated to Problem 1 and let \( (\Pi^0, \Pi^1) \) be a D&C pair.

(i) Let \( \mathbf{x} \) be an optimal solution to Problem 1 and let \( \mathbf{y}^0, \mathbf{y}^1 \) be optimal solutions to Problems 4 Π^0, Π^1-respectively. Then
\[ \sum_{i \in [N]} p_i x_i \leq \sum_{j \in A^0} p_j y_j^0 + \sum_{j \in A^1} p_j y_j^1. \]

(ii) Let \( \mathbf{x} \) be an optimal solution to Problem 1 which is both Π^b and Π^{1-b}-feasible, then \( \mathbf{x} \) is a Π^b and Π^{1-b}-optimal solution.

Proof. (i) Since \( \mathbf{y}^b \) is an optimal solution of Π^b, define the vector \( \mathbf{y} \in \{0, 1\}^N \) by
\[ y_i = \begin{cases} y_i^0 & i \in A^0, \\ y_i^1 & i \in A^1. \end{cases} \]
Then, \( \mathbf{p} \cdot \mathbf{y} = \sum_{j \in A^0} p_j y_j^0 + \sum_{j \in A^1} p_j y_j^1 \) and \( y \) is both Π^0 and Π^1-feasible i.e., \( y \in S^0 \cap S^1 \). Recalling the feasible sets inclusion (20) and that \( \mathbf{x} \) is optimal, we have \( \mathbf{p} \cdot \mathbf{x} = \min \{ \mathbf{p} \cdot \mathbf{z} : \mathbf{z} \in S \} \leq \mathbf{p} \cdot \mathbf{y} \) i.e., the result follows. \( \square \)
(ii) Let \( x \) be an optimal solution to \textsc{Problem} \( 1 \) which is also \( \Pi^b \)-feasible. Suppose that \( x \) is not an optimal solution of \textsc{Problem} \( \Pi^b \) and let \( y^b \) be its optimal solution, therefore \( \sum_{j \in A^b} p_j y^b_j < \sum_{j \in A^b} p_j x_j \). Define \( y \in \{0, 1\}^N \) by

\[
y_i \overset{\text{def}}{=} \begin{cases} y^b_i, & i \in A^b, \\ x_i, & i \in A^{1-b}. \end{cases}
\]

Observe that

\[
c \cdot y = \sum_{j \in A^b} c_j y^b_j + \sum_{j \in A^{1-b}} c_j x_j \geq D^b + D^{1-b}.
\]

Here, the inequality holds because \( y^b \) is \( \Pi^b \)-feasible and \( x \) is \( \Pi^{1-b} \)-feasible. Therefore, \( y \) is feasible for \textsc{Problem} \( 1 \) but then

\[
p \cdot y = \sum_{j \in A^b} p_j y^b_j + \sum_{j \in A^{1-b}} p_j x_j < \sum_{j \in A^b} p_j y^b_j + \sum_{j \in A^{1-b}} p_j x_j = p \cdot x
\]

and \( x \) would not be an optimal solution, which is a contradiction. Since the above holds for any \( b \in \{0, 1\} \) the proof is complete. \( \square \)

**Remark 5.** Notice that in \textsc{Proposition} \( 6 \) (ii) the hypothesis requiring the optimal solution \( x \) being both \( \Pi^b \) and \( \Pi^{1-b} \)-feasible can not be relaxed as the following example shows. Consider \textsc{Problem} \( 1 \) for \( D = 150 \) and the following data

| Room | Capacity: \( c \) | Proctors: \( p \) |
|------|------------------|-----------------|
| 1    | 100              | 2               |
| 2    | 50               | 1               |
| 3    | 100              | 2               |
| 4    | 50               | 1               |

Table 3: \textsc{Remark} \( 5 \) Data

An optimal solution is given by \( x = (1, 1, 0, 0) \) with \( p \cdot x = 3 \). Consider \( A^0 \overset{\text{def}}{=} \{1, 2\}, A^1 \overset{\text{def}}{=} \{3, 4\} \) with \( D^0 = 50, D^1 = 100 \). Then, \( x \) is \( \Pi^0 \)-feasible but it is not \( \Pi^1 \)-feasible, moreover \( x \) is not \( \Pi^0 \)-optimal because \( y = (0, 1, 1, 0) \) is \( \Pi^0 \)-feasible and

\[
\sum_{i \in A^0} p_i y_i = 1 < 3 = \sum_{i \in A^0} p_i x_i.
\]

Consequently, the optimal solution has to be both \( \Pi^0, \Pi^1 \)-feasible to guarantee that \textsc{Proposition} \( 6 \) (ii) holds.

On the other hand if we take the previous setting but replacing \( D^0 = 60, D^1 = 90 \), then \( y^0 = (1, 0, 0, 0), y^1 = (0, 0, 1, 0) \) are \( \Pi^0 \) and \( \Pi^1 \) optimal solutions, however

\[
\sum_{i \in A^0} p_i x_i = 2 + 1 < 2 + 2 = \sum_{i \in A^0} p_i y^0_i + \sum_{i \in A^0} p_i y^1_i,
\]

i.e., a global optimal solution can not be derived from the local solutions of the D&C pair. Finally, if we choose \( A^0 = \{1, 2, 3\}, A^1 = \{4\}, D^0 = 90, D^1 = 60 \), the problem \( \Pi^1 \) is not feasible.

**Remark 6.** The introduction of a D&C pair is of course aimed to reduce the computational complexity of the \textsc{Problem} \( 1 \) given that \textsc{Problem} \( 2 \) (\( \Pi^0 \)) can be regarded as a problem in \( \{0, 1\}^{|A^1|} \) with \( b \in \{0, 1\} \), instead of a problem in \( \{0, 1\}^N \). Therefore, in terms of an exhaustive search approach a D&C pair reduces order of complexity from \( O(\Gamma(N + 1)) \) to \( O(\Gamma(\frac{1}{2} N + 1)) \); in terms of a dynamic programing approach, the D&C strategy
reduces from $O(ND)$ to $2O\left(\frac{1}{4} D^2\right) = O\left(\frac{1}{2} ND\right)$. However, from the discussion above, it follows that the choice of $D^0, D^1$ is crucial when designing the pair $(\Pi^0, \Pi^1)$. Ideally, Inequality (22) would be an equality for the optimal solutions $x, y^0, y^1$, this observation motivates the definition introduced below.

**Definition 6.** Let $c = (c_i : i \in [N])$ and $p = (p_i : i \in [N])$ be the data associated to Problem 1. Let $(A^0 : b \in \{0, 1\}), (D^b : b \in \{0, 1\})$ be partitions of $[N]$ and $D$ respectively.

(i) We say the demands are **partition-dependent** if both satisfy the relationship (18) and we denote this dependence by

$$D^b = D^b(A^0, A^1), \quad b = 0, 1.$$  \hfill (23)

(ii) The D&C pair $(\Pi^b : b \in \{0, 1\})$ is said to be a **feasible pair** if both Problems 4 are feasible.

(iii) If the D&C pair $(\Pi^b : b \in \{0, 1\})$ is feasible we define its **efficiency** as

$$\text{eff}((A^0, A^1), (D^0, D^1)) \overset{\text{def}}{=} 100 \times \frac{\min \sum_{x \in S} c_i x_i - \min \sum_{y^0 \in S^0} c_i y^0 - \min \sum_{y^1 \in S^1} c_i y^1}{\min \sum_{x \in S} c_i x_i}.$$  \hfill (24)

(iv) Given $A = (A^i : j \in [J])$ and $D = (D^i : j \in [J])$ be partitions of $[N]$ and $D$ respectively such that $\Pi^j$ is feasible for all $j \in [J]$, then, its associated efficiency is defined by

$$\text{eff}(A, D) \overset{\text{def}}{=} 100 \times \frac{\min \sum_{x \in S} c_i x_i - \sum_{j \in J} \min \sum_{y^i \in S^i} c_i y^i}{\min \sum_{x \in S} c_i x_i}.$$  \hfill (25)

**Remark 7.** Notice that for any feasible pair, it holds that $0 < \text{eff}((A^0, A^1), (D^0, D^1)) \leq 1$, due to Inequality (22). In the case of general partitions $A$ and $D$, an inequality analogous to (22) can be derived using induction on the cardinal of $A$.

Before introducing the definition of efficiency for D&C trees we recall a classic definition from Graph Theory (see Section 2.3 in [6]).

**Definition 7.** Let $T = (V, E)$ be a tree and let $U \subseteq V$ be a subset of vertices. The **subtree induced** on $U$, denoted by $T(U)$, is the tree whose vertices are $U$ and whose edge-set consists on all those edges in $E$ such that both endpoints are contained in $U$.

**Definition 8.** Let $c = (c_i : i \in [N]), p = (p_i : i \in [N])$ be the data associated to Problem 1 and $DCT$ be a D&C tree associated to it and let $H$ be its height.

(i) The tree $DCT$ is said to be feasible if all its nodes are feasible problems.

(ii) Let $h \in [H]$ arbitrary, the tree pruned at height $h$ is given by

$$DCT_h = \text{subtree of } DCT \text{ induced on the set} \{V \subseteq V : \text{height}(V) \leq h\}.$$  \hfill (26)

We denote by $L(DCT_h)$ the set of leaves of the tree $DCT_h$ i.e., those vertices whose degree is equal to one.

(iii) We say that a set of leaves $L(DCT_h)$ for a given $h \in [H]$ is an **instance** of the D&C approach applied to the problem [1]
(iv) Let $DCT$ be feasible with $H$, the global and stepwise efficiencies of the tree are defined by

\[
\text{GbE}(h) \overset{\text{def}}{=} \frac{\text{soln}(V_0) - \sum \{\text{soln}(V) : V \in L(DCT_h)\}}{\text{soln}(V_0)}, \quad h \in [H]. \tag{27a}
\]

\[
\text{SwE}(h) \overset{\text{def}}{=} \frac{\sum \{\text{soln}(V) : V \in L(DCT_h)\} - \sum \{\text{soln}(V) : V \in L(DCT_{h-1})\}}{\sum \{\text{soln}(V) : V \in L(DCT_h)\}}, \quad h \in [H] - \{0\}. \tag{27b}
\]

Here, soln($V$) indicates the optimal solution of the problem associated to the vertex $V$ in the tree $DCT$ and $L(DCT_h)$ stands for the set of leaves in the tree $DCT_h$.

Next we prove that the definition[8] above makes sense.

**Theorem 7.** Let $DCT$ be a D&C tree with height $H$ and let $DCT_h$, $L(DCT_h)$ for $h \in [H]$ be as in Definition 3(ii) above. Then, \{A$^V$ : $V \in L(DCT_h)$\} is a partition of $[N]$, where $A^V$ is the set of rooms for the problem $V$.

**Proof.** We proceed by induction on the height of the tree. For $H = 0$ the result is trivial and for $H = 1$ the tree is merely consists of $V_0$ and its left and right children which, by definition are a D&C pair \{(Pi : b \in \{0, 1\}\), in particular, the sets $A^0$, $A^1$ are a partition of $[N]$. Now assume that the result is true for $H \leq k$ and let $DCT$ be such that its height is $k + 1$. Consider $DCT_k$ and $L(DCT_k)$, since the result is true for heights less or equal than $k$ we have that \{A$^V$ : $V \in L(DCT_k)$\} is a partition of $[N]$. We classify this set as follows

\[
\{A^V : V \in L(DCT_k)\} = \{A^V : V \in L(DCT_k) \cap L(DCT_{k+1})\} \cup \{A^V : V \in L(DCT_k) - L(DCT_{k+1})\}. \tag{28}
\]

However if $V \in L(DCT_k) - L(DCT_{k+1})$ it means that its left and right children $V_l$, $V_r$ belong to $L(DCT_{k+1})$. Moreover, since $(V_l, V_r)$ is a D&C pair of $V$, then $(A^{V_l}, A^{V_r})$ is a partition of $A^V$, i.e.,

\[
\{A^V : V \in L(DCT_k) - L(DCT_{k+1})\} = \{A^{V_l} : V_l \text{ left child of } V \in L(DCT_k) - L(DCT_{k+1})\} \cup \{A^{V_r} : V_r \text{ right child of } V \in L(DCT_k) - L(DCT_{k+1})\}. \tag{29}
\]

Putting together Expressions (28) and (29) the result follows. \qed

**Remark 8.** Clearly, due to Theorem 7 a set of leaves $L(DCT_h)$ for $h \in [H]$ is a potential instance of the D&C method applied to Problem 1 as (iii) Definition 8 states. It is also direct to see that the global and stepwise efficiencies $GbE(h)$ and $SwE(h)$ respectively, introduced in (iv) Definition 8 compute the ratios adding the solutions found for different partitions of the rooms.

In view of the previous discussion a natural question is how to choose D&C efficiency-optimal pairs (at least for one step and not for a full D&C tree) however, allowing complete independence between the pairs ($A^0$, $A^1$) and ($D^0$, $D^1$) i.e., the partitions of $[N]$ and $D$ respectively would introduce an overwhelmingly vast search space. Consequently, from now on, we limit our study to partition-dependent demands, see Definition 6(ii).

In the next section several ways to generate partitions ($A^0$, $A^1$) will be introduced, which will be regarded as strategies to implement the D&C approach. However, other strategies will be explored, such as the student-proctor rate $r$ and the occupancy fraction $\alpha = \frac{1}{D} \sum \{c_i : i \in [N]\}$. These are related to the problem setting (availability of resources), rather than of the choice of D&C pairs. The assessment of all the aforementioned strategies will be done using Monte Carlo simulations, when the list of capacities is regarded as a random variable ($C$ instead of $c$) with known probabilistic distribution.
4. Strategies and Heuristic Method

Since no theoretical results can be found so far for the Divide and Conquer method, its efficiency has to be determined heuristically. To that end numerical experiments will be conducted with randomly generated data, according to classical discrete distributions. Next, several strategies will be evaluated in these settings (see Figure 6). It is important to stress that the type of strategies, as well as their potential values (numerical in most of the cases) presented here, were chosen in order to simulate plausible instances of the initial problem rather than abstract, arbitrary instances of Problem 1.

4.1. Random Setting

A Random Setting Algorithm generates lists of rooms according to certain parameters defined by the user, namely the number of rooms, the distribution of its capacities (Uniform, Poisson, Binomial) and the occupancy fraction of students with respect to total capacity of the rooms; the occupancy fraction $o$ will vary between 0.5 and 0.9, this will guarantee the hypotheses of (iv) Theorem 6 are satisfied. If $C$ denotes the random variable having the capacity of the rooms, the code uses the following parameters for the distributions

(i) **Uniform.** Possible sizes $[40, 120] \cap \mathbb{N}$ i.e., $P(C = n) = \frac{1}{80}$, for $n \in [40, 120] \cap \mathbb{N}$.

(ii) **Poisson.** Average $\lambda = 65$, then $P(C = n) = \frac{\lambda^n}{n!} \exp(-\lambda)$, for $n \in \mathbb{N} \cup \{0\}$.

(iii) **Binomial.** Sample space $[480]$, success probability, $p = 0.2$, i.e., $P(C = n) = \binom{480}{n} p^n (1 - p)^{480-n}$, for $n \in [0, 480] \cap \mathbb{N}$.

An example of 4 realizations, each consisting in 8 rooms, uniformly distributed with fraction occupancy of 0.9 is displayed in Table 4 below. The Random Setting Algorithm produces a table analogous to Table 4 and exports it as the Excel file Available_Rooms.xls. Since this algorithm is trivial, we do not include its pseudocode in this paper.

| Room | Realization 1 | Realization 2 | Realization 3 | Realization 4 | Realization 5 |
|------|--------------|--------------|--------------|--------------|--------------|
| 0    | 113          | 47           | 84           | 58           | 53           |
| 1    | 54           | 67           | 119          | 49           | 104          |
| 2    | 95           | 65           | 64           | 109          | 119          |
| 3    | 89           | 95           | 91           | 78           | 61           |
| 4    | 85           | 72           | 94           | 72           | 56           |
| 5    | 87           | 60           | 62           | 70           | 94           |
| 6    | 76           | 110          | 71           | 73           | 118          |
| 7    | 105          | 108          | 51           | 49           | 72           |
| $\sum_{i=0}^{7} c_i$ | 704  | 624          | 636          | 558          | 677          |
| $D$  | 633          | 561          | 572          | 502          | 609          |

Table 4: Example of Random Setting Data: 5 Realizations, 8 rooms with uniformly distributed capacity and 0.9 student fraction occupancy

Remark 9. (i) Since the successive application of the D&C approach generates binary trees, for practical reasons, the numerical experiments will have that the number of rooms is a power of two, i.e., $N = 2^k$ for some $k \in \mathbb{N}$.

(ii) When generating a D&C tree we want to distribute the demand between left and right children according to the relation (18). Then, the inequality (21) (equivalent to the hypothesis (19) of part (iv) Theorem 5) must be satisfied. To that end a occupancy fraction $o \in \{0.5, 0.55, 0.6, ..., 0.9\}$, furnishes a reasonable domain.
4.2. Tree Generation

There will be two ways of generating a D&C tree. Every vertex $V = (A^V, D^V)$ of the tree is a list of rooms $A^V \subseteq \{N\}$ together with an assigned demand $D^V$. Denote by $V, V_l, V_r$ a vertex together with its left and right children respectively and by $|V|, |V_l|, |V_r|$ its corresponding cardinalities. The trees are constructed using the Left Pre-Order i.e., the stack has the structure [root, left-child, right-child] (see Algorithm 3.3.1 in \textsuperscript{[3]} for details). The assigned demands to the left and right children will be given by the \textbf{Expression (18)}. All the difference between the algorithms is the way left-child and right-child are defined.

I. \textbf{First Case: Head-Left Subtree}. Select the following parameters

(i) Select a sorting criterion: Specific Weight $\gamma$, Capacity $c$, Proctors Quantity $p$ or Random.

(ii) Select a fraction for the head-left subtree, i.e. $f \in [0, 1]$.

(iii) Define the minimum size of a list, i.e. the number of rooms in a leaf of the D&C tree, namely $m = 1, 2, \ldots$, etc.

Once the list of rooms is sorted according to criterion $s \in \{p, c, \gamma, \text{random}\}$, the left-child $V_l$ is defined as the first rooms of list $V$ such that $|V_l| = \lfloor f \times |V| \rfloor$ i.e., the head of the list. The right-child is defined as the complement of the left-child i.e., $V_r \equiv V - V_l$. The tree is constructed recursively as \textbf{Algorithm 2} shows.

\begin{algorithm}
\caption{Head-Left Subtree Algorithm, returns a D&C tree}
\begin{algorithmic}
\Procedure{Head-Left Subtree Generator}{Rooms’ List. Prices: $p$, Capacities: $c$.}
\State Demand: $D$. Sorting: $s \in \{p, c, \gamma, \text{random}\}$. Head-left subtree fraction: $f \in [0, 1]$. Minimum list size: $m \in [1, \#\text{Rooms’ List} \cap \mathbb{N})$
\If{$s = \gamma$} \Comment{Asking if is necessary to compute specific weight}
\State \textbf{compute} list of specific weights $(\gamma_i : i \in [N])$ \Comment{Introduced in Definition 1}
\EndIf
\State $V_0 = \text{sorted (Rooms’ List)}$ according to chosen criterion $s$
\State $V \equiv V_0$ \Comment{Initializing the root of the D&C tree}
\State $\text{D&C tree} \equiv \emptyset$ \Comment{Initializing D&C tree as empty list}
\Function{Branch}{V, f, m, D, c} \Comment{Push list V as node of the D&C tree}
\If{$|V| \geq m$} \Comment{Defining the size of the left child}
\State $lcs = \lfloor f \times |V| \rfloor$ \Comment{Computing the left child}
\State $V_l = (R_l : 1 \leq i \leq lcs)$ \Comment{Computing the left demand}
\State $D_l = \frac{\sum\{c_i : 1 \leq i \leq lcs\}}{\sum\{c_i : 1 \leq i \leq |V|\}} D$ \Comment{Recursing for the left subtree}
\State Branch($V_l, f, m, D_l, c \equiv (c_i : 1 \leq i \leq lcs)$) \Comment{Computing the right child}
\State $V_r = (R_l : lcs < i \leq |V|)$ \Comment{Computing the right demand}
\State $D_r \equiv D - D_l$ \Comment{Recursing for the right subtree}
\State Branch($V_r, f, m, D_r, c \equiv (c_i : lcs \leq i \leq |V|)$)
\State \Return{D&C tree}
\Else
\State $V \rightarrow \text{D&C tree}$ \Comment{Push list V as node of the D&C tree}
\State \Return{D&C tree}
\EndIf
\EndFunction
\EndProcedure
\end{algorithmic}
\end{algorithm}
In the table 5 below, we present a binary tree for the first column of Table 4 (Realization 1), with the following parameters: sorting by specific weight \((s = \gamma)\), \(f = 0.5\), \(m = 2\); Figure 1 shows its graphic representation. Finally, Figure 2 depicts a tree generated for the same realization, but with parameters \(s = \gamma\), \(f = 0.4\) and \(m = 2\); the corresponding table is omitted.

| Room | Vertex 0 | Vertex 1 | Vertex 2 | Vertex 3 | Vertex 4 | Vertex 5 | Vertex 6 |
|------|----------|----------|----------|----------|----------|----------|----------|
| 1    | 1        | 1        | 1        | 1        | 0        | 0        | 0        |
| 7    | 1        | 1        | 1        | 0        | 0        | 0        | 0        |
| 2    | 1        | 1        | 0        | 1        | 0        | 0        | 0        |
| 3    | 1        | 1        | 0        | 1        | 0        | 0        | 0        |
| 5    | 1        | 0        | 0        | 0        | 1        | 1        | 0        |
| 4    | 1        | 0        | 0        | 0        | 1        | 1        | 0        |
| 6    | 1        | 0        | 0        | 0        | 1        | 0        | 1        |
| 0    | 1        | 0        | 0        | 0        | 1        | 0        | 1        |

\[D = 633\quad 309\quad 144\quad 165\quad 324\quad 155\quad 169\]

Table 5: Algorithm 2 tree generated for Realization 1 of Table 4. Parameters: sorting by specific weight \(\gamma\), left subtree fraction \(f = 0.5\), minimum size leaf \(m = 2\).

![Figure 1](image1.png)

**Figure 1:** Algorithm 2 tree generated for Realization 1 of Table 4. Parameters: sorting by specific weight \((s = \gamma)\), left subtree fraction \(f = 0.5\), minimum size leaf \(m = 2\).

![Figure 2](image2.png)

**Figure 2:** Algorithm 2 tree generated for Realization 1 of Table 4. Parameters: sorting by specific weight \((s = \gamma)\), left subtree fraction \(f = 0.4\), minimum size leaf \(m = 2\).

II. Second Case: Balanced Left-Right Subtrees. Select the same parameters as in the previous case except for the fraction head \(f \in [0, 1]\) since this will be 0.5 by default. Once the list of rooms is sorted according to
criterion $s \in \{p, c, \gamma\}$, the left-child $V_l$ is defined as the rooms in even positions on the sorted list $V$. The right-child is defined as the complement of the left-child i.e., $V_r \triangleq V - V_l$ i.e., the left and right children are as balanced as possible according to $s$. Again, the tree is constructed recursively as the Algorithm 3 shows. In Table 6 below, we present a binary tree for the first column (Realization 1) of Table 4, with the

**Algorithm 3** Balanced Left-Right Subtrees Algorithm, returns a D&C tree

1: **procedure** BALANCED LEFT-RIGHT SUBTREES GENERATOR((Rooms’ List; Prices: $p$, Capacities: $c$, Demand: $D$). Sorting: $s \in \{p, c, \gamma, \text{random}\}$, Minimum list size: $m \in [1, \#\text{Rooms’ List}] \cap \mathbb{N}$)

2: if $s = \gamma$ then

3: compute list of specific weights ($\gamma_i : i \in [N]$) \>
   Introduced in Definition 1.

4: end if

5: $V_0 \triangleq \text{sorted (Rooms’ List) according to chosen criterion } s$

6: $V \triangleq V_0$ \>
   Initializing the root of the D&C tree

7: D&C tree $DCT = \emptyset$ \>
   Initializing D&C tree as empty list

8: **function** BRANCH($V, m, D, c$)

9: if $|V| > m$ then

10: $V \rightarrow DCT$ \>
    Push list $V$ as node of the D&C tree

11: $V_l = (R_i : 1 \leq i \leq |V|, i \text{ even })$

12: $D_l \triangleq \left[ \frac{\sum\{c_i : 1 \leq i \leq |V|, i \text{ even }\}}{\sum\{c_i : 1 \leq i \leq |V|\}} \right] D$ \>
    Computing the left child

13: BRANCH($V_l, m, D_l, c_l \triangleq (c_i : 1 \leq i \leq |V|, i \text{ even }))$ \>
    Recursing for the left subtree

14: $V_r = (R_i : 1 \leq i \leq |V|, i \text{ odd })$

15: $D_r \triangleq D - D_l$ \>
    Computing the right child

16: BRANCH($V_r, m, D_r, c_r \triangleq (1 \leq i \leq |V|, i \text{ odd }))$ \>
    Recursing for the right subtree

17: return $DCT$ \>
    return the D&C tree

18: else

19: $V \rightarrow DCT$ \>
    Push list $V$ as node of the D&C tree

20: return $DCT$ \>
    return the D&C tree

21: end if

22: end function

23: end procedure

following parameters: sorting by specific weight ($s = \gamma$), $m = 2$; the corresponding graphic representation is displayed in Figure 3.

| Room | Vertex 0 | Vertex 1 | Vertex 2 | Vertex 3 | Vertex 4 | Vertex 5 | Vertex 6 |
|------|----------|----------|----------|----------|----------|----------|----------|
| 1    | 1        | 1        | 1        | 0        | 0        | 0        | 0        |
| 7    | 1        | 0        | 0        | 0        | 1        | 1        | 0        |
| 2    | 1        | 1        | 0        | 1        | 0        | 0        | 0        |
| 3    | 1        | 0        | 0        | 0        | 1        | 0        | 1        |
| 5    | 1        | 1        | 1        | 0        | 0        | 0        | 0        |
| 4    | 1        | 0        | 0        | 0        | 1        | 1        | 0        |
| 6    | 1        | 1        | 0        | 1        | 0        | 0        | 0        |
| 0    | 1        | 0        | 0        | 0        | 1        | 0        | 1        |
| $D$  | 633      | 281      | 127      | 154      | 352      | 171      | 181      |

Table 6: **Algorithm** [3] tree generated for Realization 1 of Table 4. Parameters: sorting by specific weight $\gamma$, minimum size leaf $m = 2$.  

18
4.3. Efficiency Quantification

In this section we describe the general algorithm to compute the efficiency of the D&C tree approach. The efficiencies will be measured according to Definition 8, moreover the computations will be done based on three values:

1. Exact solution of Problem 1 using the technique of dynamic programming (see Section 11.3 in [2] for details), denoted by $DPS$ in the following.

2. Upper bound furnished by the Greedy Algorithm 1, denoted by $GAS$ in the sequel.

3. Lower bound, given by the solution of Problem 2, i.e., the natural linear relaxation of the problem 1, from now on denoted by $LRS$.

The effectiveness of upper and lower bounds mentioned above is measured in the standard way i.e.,

$$GAE \overset{\text{def}}{=} 100 \times \frac{GAS - DPS}{DPS}, \quad LRE \overset{\text{def}}{=} 100 \times \frac{DPS - LRS}{DPS}. \quad (30)$$

Here, $GAE, LRE$ respectively indicate, Greedy Algorithm and Liner Relaxation Efficiency. The general structure is as follows

(i) Execute the Random Setting Algorithm described in Section 4.1, according to its parameters of choice and store its results in the file $Available\_Rooms.xls$.

(ii) Loop through the columns of file $Available\_Rooms.xls$, each of them is a random realization (see Table 4).

(iii) For each column/realization,

(a) Retrieve the basic information of Problem 1 i.e., Rooms’ List, Prices: $p$, Capacities: $c$, Demand: $D$.

(b) Build the D&C tree, Head-Left (Algorithm 2) or balanced (Algorithm 3) according to user’s choice.

(c) Loop through the D&C tree nodes, compute the Greedy Algorithm 1 Dynamic Programming and Linear Relaxation solutions and store them in the D&C tree structure.

(d) Loop through the D&C tree heights, compute the global and stepwise efficiencies according to Definition 8 and store them in stack structures of a realizations global table (see, Table 8). Compute the Greedy Algorithm and Linear Relaxation Efficiencies as defined in Equation (30) and store them in stack structures of a realizations global table (see Table 9).
(iv) In the realizations global table, compute the average of the global and stepwise efficiencies.

The steps (ii) and (iii) of the previous description are detailed in the pseudocode 4, an example of its outcome is presented in the table 7 below we present the efficiencies of the method for the Realization 1 of Table 4 with the D&C tree structure presented in Figure 1 and detailed in Table 5.

| Height | LRS | DPS | GAS | GbE<sub>LRS</sub> | GbE<sub>DPS</sub> | GbE<sub>GAS</sub> | SwE<sub>LRS</sub> | SwE<sub>DPS</sub> | SwE<sub>GAS</sub> |
|--------|-----|-----|-----|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 0      | 14.12 | 15  | 16  | 0.00            | 0.00            | 0.00            | 0.00            | 0.00            | 0.00            |
| 1      | 14.25 | 16  | 16  | 0.98            | 6.67            | 0.00            | 0.98            | 6.67            | 0.00            |
| 2      | 14.36 | 16  | 16  | 1.71            | 6.67            | 0.00            | 0.72            | 0.00            | 0.00            |

Table 7: Algorithm height efficiencies for Realization 1 of Table 4. Parameters: sorting by specific weight γ, left subtree fraction f = 0.5, minimum size leaf m = 2, see also Figure 4 and Table 5 for details on the tree structure. The Linear Relaxation, Dynamic Programming and Greedy Algorithm solutions are represented with the initials LRS, DPS and GAS respectively. The Global and Stepwise Efficiencies are represented with the initials GBE, SwE respectively and the subindex affecting them indicates for which of the solutions LRS, DPS, GAS the column values apply.

In addition, for the five realizations of Table 4, the Table 8 presents the result of computing the global and stepwise efficiencies (GbE and SwE) of the Dynamic Programming Solutions (DPS), while Table 9 displays the corresponding values of the Greedy Algorithm and the Linear Relaxation Efficiencies (GAE and LRE). So far,

| Height | GbE<sub>1</sub> | GbE<sub>2</sub> | GbE<sub>3</sub> | GbE<sub>4</sub> | GbE<sub>5</sub> | SwE<sub>1</sub> | SwE<sub>2</sub> | SwE<sub>3</sub> | SwE<sub>4</sub> | SwE<sub>5</sub> |
|--------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0      | 0.00           | 0.00           | 0.00           | 0.00           | 0.00           | 0.00           | 0.00           | 0.00           | 0.00           | 0.00           |
| 1      | 6.67           | 14.29          | 14.29          | 7.14           | 13.33          | 6.67           | 14.29          | 14.29          | 7.14           | 13.33          |
| 2      | 6.67           | 14.29          | 14.29          | 7.14           | 13.33          | 0.00           | 0.00           | 0.00           | 0.00           | 0.00           |

Table 8: Algorithm example of Global Efficiency (GbE) and Stepwise Efficiency (SwE) results for the case Dynamic Programming Solution (DPS) through the 5 realizations of Table 4. Parameters: sorting by specific weight γ, left subtree fraction f = 0.5, minimum size leaf m = 2. The subindex affecting GBE and SwE indicates the corresponding number of realization for which the column applies.

| Height | GAE<sub>1</sub> | GAE<sub>2</sub> | GAE<sub>3</sub> | GAE<sub>4</sub> | GAE<sub>5</sub> | LRE<sub>1</sub> | LRE<sub>2</sub> | LRE<sub>3</sub> | LRE<sub>4</sub> | LRE<sub>5</sub> |
|--------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0      | 6.67           | 0.00           | 0.00           | 7.14           | 0.00           | 5.90           | 0.66           | 0.45           | 6.65           | 2.62           |
| 1      | 0.00           | 0.00           | 0.00           | 0.00           | 10.91          | 11.76          | 11.21          | 11.74          | 12.00          | 12.00          |
| 2      | 0.00           | 0.00           | 0.00           | 0.00           | 10.27          | 10.68          | 10.51          | 10.52          | 10.58          | 10.58          |

Table 9: Algorithm example of Greedy Algorithm Efficiency (GAE) and Linear Relaxation Efficiency (LRE) (see Equation 50 for its definition), through the 5 realizations of Table 4. Parameters: sorting by specific weight γ, left subtree fraction f = 0.5, minimum size leaf m = 2. The subindex affecting GAE and LRE indicates the corresponding number of realization for which the column applies.

we have been using Realization 1 in Table 4 to illustrate the method, however, we close this section presenting an example significantly larger in order to illustrate the method for a richer D&C tree and bigger range of heights.

**Example 1 (The D&C tree of a large random realization).** In Table 10 we present the LRS, DPS, GAS solutions for a D&C tree corresponding to a random realization of 128 rooms, uniformly distributed room capacities, with occupancy fraction of 0.9 (11074 students). The respective D&C tree is constructed using the head-left algorithm 3 sorted by specific weight γ, left subtree fraction f = 0.5 and minimum size m = 1, i.e., its height is 7. To avoid redundancy, we omit tables displaying the corresponding values of GAE, LRE as well as GBE, SwE for LRS, DPS, GAS, analogous to those registered in tables 7 and 9 since they can be completely derived from Table 10 however, we display the graphics corresponding to all such tables.

In Figure 5 we depict the behavior through the heights of a D&C tree, for the solutions LRS, DPS, GAS, the efficiencies GAE, LRE, as well as the global and stepwise efficiencies \{ GBE<sub>LRS</sub>, GBE<sub>DPS</sub>, GBE<sub>GAS</sub> \}, \{ SwE<sub>LRS</sub>, SwE<sub>DPS</sub>, SwE<sub>GAS</sub> \).
Algorithm 4: D&C Efficiency Quantification, returns a list of global and stepwise efficiencies

1: procedure D&C EFFICIENCY QUANTIFICATION ([File Available_Rooms.xls contains: Rooms’ List, Prices: p, Capacities: c, Demand: D.]

   User Decisions: Sorting: s ∈ {p, c, γ, random}, Head-left subtree fraction: f ∈ [0, 1].
   Minimum list size: m ∈ [1, #Rooms’ List] \ [N], Student-Proctor rate: r ∈ [1, max c_i].

   Type of Tree: t ∈ {Head-Left, Balanced}.)

2: for column of Available_Rooms.xls do  \Comment{Each column is a random realization, e.g. TABLE 4}

3:     retrieve from Available_Rooms.xls the information: Rooms’ List, Prices: p, Capacities: c, Demand: D, corresponding to column/realization.

4:     if t = Head-Left then
5:          D&C tree: DCT ≜ call Algorithm 2 (Rooms’ List, p, c, D, s, f, m )
          \Comment{Producing the Head-Left D&C tree}
6:     else
7:          D&C tree: DCT ≜ call Algorithm 3 (Rooms’ List, p, c, D, s, m )
          \Comment{Producing the Balanced D&C tree}
8:     end if
9:     Solutions Tree: ST = \emptyset  \Comment{Initializing Solutions Tree as empty list}
10:    for V ∈ vertices of DCT do
11:          \Comment{Recall that DCT has table format as TABLE 6}
12:     Linear Relaxation Solution: LRS_V ≜ call simplex algorithm solver (Data {p, D}, corresponding to vertex V)
13:     Dynamic Programming Solution: DPS_V ≜ call dynamic programming solver Data {p, D}, corresponding to vertex V)
14:     Greedy Algorithm Solution: GAS_V ≜ call Algorithm 1 (Data {p, D}, corresponding to vertex V)
15:    \Comment{Push the triple [LRS_V, DPS_V, GAS_V] as vertex of the solutions tree ST}
16:     end for
17:     soln ≜ \emptyset  \Comment{Initializing solution values stack as empty list}
18:     GBE ≜ [0]  \Comment{Initializing global efficiency stack; 0 is the first value}
19:     Swe ≜ \emptyset  \Comment{Initializing stepwise efficiency stack as empty list}
20:     GAE ≜ \emptyset  \Comment{Initializing greedy algorithm efficiency stack as empty list}
21:     LRE ≜ \emptyset  \Comment{Initializing linear relaxation efficiency stack as empty list}
22:     H ≜ height of DCT.
23:    for h ∈ [H] do
24:          \Comment{Tree pruned at height h}
25:          DCT_h = subgraph of DCT induced on the set {V ∈ DCT : height(V) <= h }
26:          L(DCT_h) = \{V ∈ DCT_h : deg(V) = 1\}  \Comment{Selecting the leaves of the pruned tree DCT_h}
27:          soln(h) ≜ \sum \{[LRS_V, DPS_V, GAS_V] : V ∈ L(DCT_h)\}  \Comment{Push the total solutions (Linear Relaxation, Dynamic, Greedy) at height h of the three DCT, to the stack}
28:          if h > 0 then
29:              GBE(h) ← 100 × \frac{soln(h) - soln(0)}{soln(0)}  \Comment{Push global efficiency at height h to the stack}
30:              Swe(h - 1) ← 100 × \frac{soln(h) - soln(h - 1)}{soln(h - 1)}  \Comment{Push stepwise efficiency at height h to the stack}
31:          end if
32:          GAE(h) ← 100 × \frac{soln[GAS](h) - soln[DPS](h)}{soln[DPS](h)}  \Comment{Push greedy algorithm efficiencies into the stack, see EQUATION (30)}
33:          LRE(h) ← 100 × \frac{soln[DPS](h) - soln[LRS](h)}{soln[DPS](h)}  \Comment{Push linear relaxation efficiencies into the stack, see EQUATION (30)}
34:    end for
35:     return (GBE, Swe)  \Comment{Efficiencies corresponding to column/realization}
36: end procedure

\Comment{Algorithm 1: Linear Relaxation Algorithm}

1: procedure LINEAR RELAXATION (Data {p, D, r, S, H})

2: Define the problem $\Gamma$ as an $S$-terminal, $H$-layer network with demand r.
3: Define a solution as a $H$-layer flow $\phi$ that satisfies the demands $\phi_{s,t}$:
4: \begin{equation}
5: \sum_{t \in S} \phi_{s,t} = r, \quad \sum_{s \in S} \phi_{s,t} = r.
6: \end{equation}
7: Call a flow solution $\phi$ feasible if $\phi_{s,t} \leq p_{s,t}$ for all $(s,t) \in E$.
8: Define a feasible flow solution $\phi$ as optimal if $\phi_{s,t} \leq p_{s,t}$ for all $(s,t) \in E$.
9: Then, $\phi$ is optimal if and only if the corresponding linear program:
10: \begin{equation}
11: \begin{aligned}
12: \min & \sum_{e \in E} c_e x_e, \\
13: \text{subject to} & \quad \sum_{e \in E} x_e - \sum_{e \in E} x_e = r, \quad \sum_{e \in E} x_e = r, \\
14: & \quad x_e \geq 0, \quad x_e \leq p_{s,t}.
15: \end{aligned}
16: \end{equation}
17: has an optimal solution $x^*$ such that $x^*_e = \phi_{s,t}$ for all $(s,t) \in E$.
18: \Comment{Algorithm 2: Dynamic Programming Algorithm}
19: procedure DYNAMIC PROGRAMMING (Data {p, D, r, S, H})
20: Define a problem $\Gamma$ as an $S$-terminal, $H$-layer network with demand r.
21: Define a solution as a $H$-layer $2$-flow $\phi$ that satisfies the demands $\phi_{s,t}$:
22: \begin{equation}
23: \sum_{t \in S} \phi_{s,t} = r, \quad \sum_{s \in S} \phi_{s,t} = r.
24: \end{equation}
25: Call a flow solution $\phi$ feasible if $\phi_{s,t} \leq p_{s,t}$ for all $(s,t) \in E$.
26: Define a feasible flow solution $\phi$ as optimal if $\phi_{s,t} \leq p_{s,t}$ for all $(s,t) \in E$.
27: Then, $\phi$ is optimal if and only if the corresponding linear program:
28: \begin{equation}
29: \begin{aligned}
30: \min & \sum_{e \in E} c_e x_e, \\
31: \text{subject to} & \quad \sum_{e \in E} x_e - \sum_{e \in E} x_e = r, \quad \sum_{e \in E} x_e = r, \\
32: & \quad x_e \geq 0, \quad x_e \leq p_{s,t}.
33: \end{aligned}
34: \end{equation}
35: has an optimal solution $x^*$ such that $x^*_e = \phi_{s,t}$ for all $(s,t) \in E$.
36: \Comment{Algorithm 3: Greedy Algorithm}
37: procedure GREEDY ALGORITHM (Data {p, D, r, S, H})
38: Define a problem $\Gamma$ as an $S$-terminal, $H$-layer network with demand r.
39: Define a solution as a $H$-layer $2$-flow $\phi$ that satisfies the demands $\phi_{s,t}$:
40: \begin{equation}
41: \sum_{t \in S} \phi_{s,t} = r, \quad \sum_{s \in S} \phi_{s,t} = r.
42: \end{equation}
43: Call a flow solution $\phi$ feasible if $\phi_{s,t} \leq p_{s,t}$ for all $(s,t) \in E$.
44: Define a feasible flow solution $\phi$ as optimal if $\phi_{s,t} \leq p_{s,t}$ for all $(s,t) \in E$.
45: Then, $\phi$ is optimal if and only if the corresponding linear program:
46: \begin{equation}
47: \begin{aligned}
48: \min & \sum_{e \in E} c_e x_e, \\
49: \text{subject to} & \quad \sum_{e \in E} x_e - \sum_{e \in E} x_e = r, \quad \sum_{e \in E} x_e = r, \\
50: & \quad x_e \geq 0, \quad x_e \leq p_{s,t}.
51: \end{aligned}
52: \end{equation}
53: has an optimal solution $x^*$ such that $x^*_e = \phi_{s,t}$ for all $(s,t) \in E$.
As it can be seen in figures (a), (b), GAS is significantly more accurate than LRS to the point that one curve stays below the other through all the height of the D&C tree. In the case of global efficiencies we also observe that the behavior of GbEGAS and GbEDPS are similar, though none is above the other through all the D&C tree heights and GbEDPS stays below both of them. A similar behavior is observed for the case of stepwise efficiencies (SwE) though the curves SwEDPS and SwELRS intersect in this case for $h = 2$. Observe that if $h \geq 4$, the results for DPS, GAS, GAE, GbEDPS, GbEGAS become stable i.e., the D&C method no longer deteriorates the exact solution; since $N = 128$, $h \geq 4$ corresponds to lists of 8 rooms or smaller. Finally, in Figure 5 we present five random realizations, for the efficiencies GbEDPS, SwEDPS, GAE and LRE. We choose depicting this efficiencies because the Dynamic Programming Solution (DPS) is the most important parameter, as it measure the quality of the exact solution and the GAE, LRE efficiencies store the quality of the usual bounds (Greedy Algorithm and Linear Relaxation). The realizations are generated with the same parameters of the previous one (therefore comparable to it) and observe similar behavior amongst them as expected. In particular, notice that for $h \geq 4$ (subproblems of size 8 or smaller) the solutions stabilize.

**Remark 10.** Examples of 128 rooms, with large number of realizations and different distributions (uniform, binomial, Poisson) present similar behavior to the one presented in Example 1. In particular, most of the results stabilize for $h \geq 4$ (subproblems of 8 rooms), for the three distributions.

### 5. Numerical Experiments

In this section, we present the results from the numerical experiments. All the codes needed for the present work were implemented in Python and the databases were handled with Pandas (Python Data Analysis Library). The full scale experiments were run in the supercomputer Gauss at Universidad Nacional de Colombia, Sede Medellín, Facultad de Ciencias.

#### 5.1. The Experiments Design

The numerical experiments are aimed to asses the effectiveness of the heuristic D&C method presented in Section 4. Its whole construction was done in a way such that its effectiveness could be analyzed under the probabilistic view of the Law of Large Numbers (which we write below for the sake of completeness, its proof and details can be found in [3]).

**Theorem 8 (Law of Large Numbers).** Let $(Z^{(n)} : n \in \mathbb{N})$ be a sequence of independent, identically distributed random variables with expectation $E(Z^{(1)})$, then

$$\Pr \left[ \left| \frac{Z^{(1)} + Z^{(2)} + \ldots + Z^{(n)}}{n} - E(Z^{(1)}) \right| > 0 \right] \xrightarrow{n \to \infty} 0,$$

i.e., the sequence $(Z^{(n)} : n \in \mathbb{N})$ converges to $\mu$ in the Cesàro sense.

### Table 10: Example 1. Solutions LRS, DPS and GAS table for a random realization of 128 rooms uniformly distributed and occupancy fraction of 0.9. The D&C tree has height 7, generated by the head-left algorithm 2, sorted by specific weight $\gamma$, left subtree fraction $f = 0.5$ and minimum size $m = 1$.

| Height | LRS | DPS | GAS |
|--------|-----|-----|-----|
| 0      | 233.43 | 234 | 236 |
| 1      | 236.41 | 239 | 239 |
| 2      | 238.02 | 240 | 242 |
| 3      | 238.79 | 248 | 249 |
| 4      | 239.12 | 262 | 266 |
| 5      | 239.25 | 266 | 266 |
| 6      | 239.33 | 266 | 266 |
| 7      | 239.38 | 266 | 266 |


The D&C method introduces several free/decision parameters to analyze the behavior of PROBLEM 1 under different scenarios. We have the following list of domains for each of these parameters

a. Number of rooms: \( N \in \mathbb{N} \).

b. Distribution of rooms’ capacities: \( \text{dist} \in \{ \text{Ud}, \text{Pd}, \text{Bd} \} \) (Ud: uniform, Pd: Poisson, Bd: Binomial).

c. Occupancy fraction: \( \phi \in \{0, 50, 0.55, \ldots, 0.90\} \) (to satisfy hypotheses of (iv) THEOREM 6).

d. Proctors-Students rate: \( r \in \{34, 44, 54, 64, 74\} \) (to avoid hypotheses of THEOREM 4 being satisfied).

e. D&C tree algorithm \( T\text{-alg} \in \{ \text{hlT}, \text{blT} \} \) (hlT head-left Tree ALGORITHM 2, blT balanced-left Tree ALGORITHM 3).

f. Rooms list sorting method: \( s \in \{ p, c, \gamma, \text{random} \} \).

g. Fraction of the left list: \( f \in \{0.35, 0.40, \ldots, 0.65\} \).
Figure 5: Five random realizations of 128 rooms, with uniformly distributed capacities and occupancy fraction of 0.9. The D&C tree has height 7, it is generated by the head-left algorithm sorted by specific weight $\gamma$, left subtree fraction $f = 0.5$ and minimum size $m = 1$.

h. Minimum list size: $m \in \mathbb{N}$.

Remark 11 (Parameters Domains). It is clear that $o$ and $f$ could very well adopt any value inside the interval $[0,1]$, while $r$ could be any arbitrary number in $\mathbb{N}$. However, adopting such ranges is impractical for two reasons. First, its infinite nature prevents an exhaustive exploration as we intend to do. Second, most of its values in such a large range are unrealistic. For instance: $o = 0.1$ means that the capacity of available rooms is 10 times the demand (scenario which will hardly occur), $f = 0$ means no D&C pair was introduced and $r \geq \max_{i \in [N]} c_i$ means there are more proctors than students.

In order to model, an integer problem of type 1 and its D&C solution as random variables, we need to introduce the following definition

Definition 9. Consider the following probabilistic space and random variables.

(i) Denote by $\Omega$ the set of all possible integer problems of the type 1.
(ii) Define the random problem generator variable as

\[ X : \mathbb{N} \times \{ \text{Ud, Pd, Bd} \} \times \{ 0, 50, 0.55, ..., 0, 90 \} \to \Omega \]

\[ (N, \text{dist}, o) \mapsto X(N, \text{dist}, o). \] (32)

Here, \( X(N, \text{dist}, o) \) is an integer problem of type [1].

(iii) Define the D&C solution variable by

\[ S : \Omega \times \{ 34, 44, ..., 74 \} \times \{ \text{hlT, blT} \} \times \{ 0.35, 0.40, ..., 0.65 \} \times \{ \text{random} \} \times \mathbb{N} \to \bigcup_{h \in \mathbb{N}} \mathbb{N}^h \]

\[ (X, r, T\text{-alg}, s, f, m) \mapsto S(X, r, T\text{-alg}, s, f, m). \] (33)

In the expression above, it is understood that \( X = X(N, \text{dist}, o) \) in the random problem generator variable and \( S(X(N, \text{dist}, o), r, T\text{-alg}, s, f, m) \) indicates the solution for the chosen integer problem \( X \in \Omega \), under the D&C tree solution parameters \( r, T\text{-alg}, s, f, m \), i.e. a stack/vector of solutions in \( \mathbb{N}^H \) where \( H \) is the height of the constructed D&C tree. In particular, notice that \( H \) is also a random variable.

Notice that if the parameters \( N, s, m \) are fixed then \( H \) is constant and the D&C solutions random variable \( S(X(N, \text{dist}, o), r, T\text{-alg}, s, f, m) \) indicates the solution for the chosen integer problem \( X \in \Omega \), under the D&C tree solution parameters \( r, T\text{-alg}, s, f, m \), i.e. a stack/vector of solutions in \( \mathbb{N}^M \) where \( H \) is the height of the constructed D&C tree. In particular, notice that \( H \) is also a random variable.

In order to compare the different scenarios without introducing too many possibilities a standard had to be fixed, which is listed below together with the justification behind its choice.

**Definition 10.** In the following we refer to the standard setting of a numerical experiment \( P = (N, \text{dist}, o, r, T\text{-alg}, s, f, m) \) for \( T\text{-alg} = \text{hlT} \), or \( P = (N, \text{dist}, o, r, T\text{-alg}, s, m) \) when \( T\text{-alg} = \text{blT} \) if its parameters satisfy the following values:

(i) Head Fraction, \( f = 0.5 \) (applies for the head-left method only). To make it comparable with the balanced method.

(ii) Occupancy Fraction, \( o = 0.9 \). From experience, it is reasonable to assume a slack capacity of 10% when booking rooms for an activity such as a massive test.

(iii) Proctors-Students rate, \( r = 54 \). From experience, this is a common value.

(iv) Rooms list sorting method, \( s = \gamma \) i.e., specific weight. Because this greedy function is closely related to the solutions furnished by the linear relaxation (LRS) as presented in Theorem 2.

(v) Minimum list size, \( m = 4 \). From multiple random realizations, it has been observed that the D&C method does not yield significant different results for list sizes smaller than \( m = 8 \); see Remark 10. Consequently, we adopt the size \( m = 4 \) in order to capture one step (and only one) of this “stopping behavior”.

(vi) Number of rooms, \( N = 512 \). This size was chosen because for \( m = 4 \) it will produce in most of the studied cases a D&C tree of height 7. The only exceptions will occur for head-left generated trees with head fraction \( f \neq 0.5 \).

In addition the next conventions are adopted.
a. An experiment is defined by a list of parameters, namely $P$; from now on we do not make distinction between the experiment and its list of parameters. Moreover, $P$ has 8 parameters if $\text{T-alg} = \text{hlIT}$ and 7 if $\text{T-alg} = \text{blIT}$. To ease notation, from now on we denote $P = (512, \text{dist}, o, r, \text{T-alg}, s, f, 4)$ for any experiment in general, in the understanding that if $\text{T-alg} = \text{blIT}$ the head fraction $f$ is not present in the list $P$.

b. Each case will be analyzed using 50 randomly generated realizations of 512 rooms with Uniform, Poisson and Binomial distributions respectively i.e., $P = (512, \text{dist}, o, r, \text{T-alg}, s, f, 4)$, see Figure 6.

c. Given a standard setting $P = (512, \text{dist}, o, r, \text{T-alg}, s, f, 4)$ and a variable $v \in \{o, r, s, f\}$, we denote by $P(v)$ the list of experiments where the variable $v$ runs through its whole domain, see Table 11 and Figure 7.

d. The analysis of the efficiencies $\text{GAE}, \text{LRE}, \text{GbE}_{\text{LRS}}, \text{GbE}_{\text{DPS}}, \text{GbE}_{\text{GAS}}, \text{SwE}_{\text{LRS}}, \text{SwE}_{\text{DPS}}$ and $\text{SwE}_{\text{GAS}}$ will be done using their average values, corresponding to the 50 random realizations mentioned above. In the following, we denote by $E$ the list of these efficiencies. Due to the Law of Large Numbers we know this is an approximation of the efficiencies expected values. An example is presented in Table 11 and Figure 7 below.

5.2. Critical Height and hlIT vs. blIT strategies Comparison

As a first step we find a critical height. From the numerical experiments, it is observed that the method heavily deteriorates beyond certain height i.e., after certain number of D&C iterations, as it can be seen in the figures 4 and 5 from Example 1 where it can be observed that beyond $h > 3$ the slope becomes very steep, therefore a critical height needs to be adopted.

**Definition 11.** Given an experiment of 50 realizations with a fixed set of parameters $P = (512, \text{dist}, o, r, \text{T-alg}, s, f, 4)$ and let $v \in \{o, r, s, f\}$ be a variable running through its full domain. Define

(i) For a fixed efficiency $\text{eff} \in E$, denote by $S(\text{eff}, P)$, the average value of 50 random realizations executed with parameters $P$ and by $S(\text{eff}, P, v)$ the list of such values when the variable $v$ runs through its whole domain, see Table 11 and Figure 7 below.

(ii) For a fixed efficiency $\text{eff} \in E$, denote by

$$S'(\text{eff}, P, v)(h) \overset{\text{def}}{=} S(\text{eff}, P, v)(h) - S(\text{eff}, P, v)(h - 1),$$

with $H$ the height of the D&C tree. Denote by $S'_v(\text{eff}, P)(h) = \max\{S'(\text{eff}, P, v)(h) : v \in \text{full domain}\}$.

(iii) For each of the efficiencies $\text{eff} \in E$, its critical height relative to the variable $v$, $h_v(\text{eff}, P)$ is the last height $h$ satisfying $S'_v(\text{eff}, P)(h) \leq 2S'_v(\text{eff}, P)(h - 1)$, see Table 11 and Figure 7.

(iv) The critical height of the experiment relative to the variable $v$, denoted by $h_v(P)$ is given by the mode of the list $\{h_v(\text{eff}, P) : \text{eff} \in E\}$.

(v) In order to compare the experiments $P_{\text{hlIT}} = (512, \text{dist}, o, r, \text{hlIT}, s, 0.5, 4)$ and $P_{\text{blIT}} = (512, \text{dist}, o, r, \text{blIT}, s, 4)$ (head-left vs balanced), relative to the variable $v$, we proceed as follows: set the height $h \overset{\text{def}}{=} \min\{h_v(P_{\text{hlIT}}), h_v(P_{\text{blIT}})\}$ (see Table 13) and compute the $\ell^1$-norm for the array $\{S(\text{eff}, P_{\text{hlIT}}, v)(h) : \text{eff} \in E, h = 1, 2, ..., h\}$ and $\{S(\text{eff}, P_{\text{blIT}}, v)(h) : \text{eff} \in E, h = 1, 2, ..., h\}$ when regarded as lists (not as matrices as Table 11 would suggest). The lowest of these norms yields the best strategy among hlIT and blIT.

**Example 2.** In the Table 11 below we display $\{S(\text{GbE}_{\text{DPS}}, P, r)(h) : h = 0, 1, ..., 7\}$ i.e., the averaged values corresponding to 50 realizations for the efficiency $\text{eff} = \text{GbE}_{\text{DPS}}$ running through the full domain of the processor student rate i.e., $v = r$. The list of parameters is given by $P = (512, \text{Ud}, 0.9, r, \text{hlIT}, \gamma, 0.5, 4)$ with $r \in \{34, 44, 54, 64, 74\}$. The tables corresponding to the intermediate slope variables $\{S'(P, v)(h) : h = 0, 1, ..., 7\}$, $\{S'_v(P)(h) : h = 0, 1, ..., 7\}$ are omitted since they can be completely deduced from Table 11. In this particular example $h_v(\text{eff}) = h_v(\text{GbE}_{\text{DPS}}) = 5$. Finally, the corresponding solution is presented in Figure 7(a), together
First level branching: hlT vs. blT tree generation methods in search for the best strategy. The probabilistic distribution is not specified given that the same set of experiments repeats for $dist = \{U_b, P_d, B_d\}$.

Second level branching for the blT method. The numerical experiments search the optimal strategies $o$, $r$, $s$. The probabilistic distribution is not specified given that the same set of experiments repeats for $dist = \{U_b, P_d, B_d\}$.

Figure 6: Schematics of the set of numerical experiments in search of optimal strategies. The first level (depicted in Figure (a)) branches on the tree generation method: lhT and blT. The second level branches on the remaining strategies: $o$, $r$, $s$ for both {lhT, blT} and $f$ for the lhT method. Figure (b) displays the branching process for the blT method; a similar diagram corresponds for the lhT method.
with its analogous for the efficiencies \( Sw_{DPS}, GAE, LRE \) ((b), (c) and (d) respectively). We chose to present these efficiencies because the Dynamic Programming Solution behavior \( DPS \), is the central parameter to assess the quality of the method for measuring the quality of the solution, while the efficiencies \( GAE, LRE \) measure the expected quality of the usual bounds (Greedy Algorithm and Linear Relaxation) through the D&C tree.

Given that the aim of this section is to compare the generation methods hlT vs. blT, we first find the optimal head fraction value \( f \) for hlT, in order to attain the best possible efficiencies for the hlT method. The results are summarized in Table 12 below; the pointing arrows indicate the the optimal head fraction values

| Head Fraction \( f \) | Uniform \( h_{o}(P_{blT}) = 4 \) | Poisson \( h_{o}(P_{blT}) = 3 \) | Binomial \( h_{o}(P_{blT}) = 3 \) |
|-----------------------|---------------------------|-----------------|------------------|
|                       | \( Gb_{DPS} \) \( \text{eff} \in \mathcal{E} \) | \( Gb_{DPS} \) \( \text{eff} \in \mathcal{E} \) | \( Gb_{DPS} \) \( \text{eff} \in \mathcal{E} \) |
| 0.35                  | 8.27                      | 26.45           | 2.83             |
| 0.35                  | 9.30                      | 28.20           | 3.06             |
| 0.50                  | 9.62                      | 30.51           | 2.94             |
| 0.55                  | 9.72                      | 30.57           | 2.95             |
| 0.60                  | 9.64                      | 30.48           | 3.12             |
| 0.65                  | 9.73                      | 30.63           | 2.82             |

Table 12: Head Fraction Comparison. Table registering values of \( l^1 \)-norms of arrays \( \tilde{S}(\text{eff}, P_{blT}, f)(h) : \text{eff} \in \mathcal{E}, h = 1, 2, ..., h( P_{blT}) \) for each \( h \) and \( f \), with \( \{ B_{DPS}, P_{blT}, f( h) : h = 1, 2, ..., h( P_{blT}) \} \). The values are displayed for \( f \) in full domain and \( dist \in \{ Ud, Bd \} \). The remaining parameters are \( o = 0.9, r = 54, s = \gamma \) and Minimum List Size \( m = 4 \) i.e., \( P_{blT} = (512, dist, o, 54, hlT, r, 0.5, 4) \). The pointing arrows indicate the optimal strategy within its column or family of comparable experiments.

Adopting the optimal heights from the hlT method and recalling the definitions above, the list of critical heights is summarized in the table 13 below. The pointing arrows indicate the comparison height between hlT and blT tree generation methods.

| Var | Uniform, \( f = 0.35 \) | Poisson, \( f = 0.65 \) | Binomial, \( f = 0.35 \) |
|-----|-----------------|-----------------|-----------------|
| \( v \) | Head-Left | Balanced | Head-Left | Balanced | Head-Left | Balanced |
| \( o \) | 5 | \( \Rightarrow 4 \) | 4 | \( \Rightarrow 3 \) | 4 | \( \Rightarrow 4 \) |
| \( r \) | \( \Rightarrow 5 \) | \( \Rightarrow 5 \) | 2 | 3 | \( \Rightarrow 2 \) | 3 |
| \( s \) | \( \Rightarrow 4 \) | 6 | \( \Rightarrow 3 \) | 4 | \( \Rightarrow 3 \) | 5 |

Table 13: Critical Heights table. Each corresponds to the expected values of efficiencies \( \mathcal{E} \) coming from the experiments with parameters \( P = (512, dist, o, r, hlT, s, f, 4) \) \( f = 0.35 \) for \( dist \in \{ Ud, Bd \} \), \( f = 0.65 \) for \( dist = Bd \) or \( P = (512, dist, o, r, blT, s, 4) \). The heights pointed with arrows are the values valid for comparison between the hlT and blT tree generation methods.

Once the heights' comparison values are found, we proceed to compare both methods in analogous conditions i.e., when the remaining variables are equal. The results for the occupancy fraction variable \( o \) running through
Figure 7: Example 2. Averaged values for 50 random realizations. Four particular efficiencies are depicted $GbE_{DPS}$, $SwE_{DPS}$, $GAE$, $LRE$. The notation $Exp_r$ with $r \in \{34, 44, 54, 64, 74\}$ in the graphics’ legends, stands for the expected value for the corresponding $\bar{S}(eff, P, r)$, $eff \in \{GbE_{DPS}, SwE_{DPS}, GAE, LRE\}$.

Its full domain, are summarized in Table 14. Similar tables were constructed for the proctor-student rate $r \in \{34, 44, 54, 64, 74\}$ and the sorting $s \in \{p, \gamma, \text{random}\}$ variables running through their respective full domains which we omit here for the sake of brevity.

It is important to notice that in Table 14 all the values corresponding to the blT method are lower than its corresponding analogous for the hIT algorithm. The same phenomenon can be observed for the table running through the proctor-student rate $r$. The table running through the sorting variable also shows clear predominance of the blT over the hIT method, though it is not absolute (10 out of 12 cases) as in the previous cases. Furthermore, noticing the differences of values, it follows that blT produces significant better results than hIT. Therefore, this choice of strategy when using the D&C approach is clear and the remaining strategies need to be decided based on blT tree generation algorithm results.

Remark 12 (Head Fraction $f$ values). With regard to the optimal head fraction values it is important to notice the following

(i) As expected, the optimal values tend to be on the extremes $f = 0.35$ or $f = 0.65$, since $f = 0$ or $f = 1$
would imply that no D&C pair has been introduced and therefore the efficiency should be 100%.

(ii) Notice that hlt generated D&C tree for \( f \neq 0.5 \) will be deeper than its analogous for blT, see Figures 1 and 3. In particular, the hlt method with optimal head fraction values (\( f = 0.35, f = 0.65 \)) has higher complexity than its blT analogous.

(iii) A similar comparison procedure was done between hlt and blT, when \( f = 0.5 \) i.e., the standard. As expected, the hlt yields poorer results than using the optimal head fraction values and blT is remarkably superior.

| Occupancy | Uniform Head-Left | Poisson Head-Left | Binomial Head-Left |
|-----------|------------------|------------------|-------------------|
| \( o \)   | \( f = 0.35 \)   | \( f = 0.65 \)   | \( f = 0.35 \)   |
| 0.50      | 98.66            | 35.51            | 49.59             |
| 0.55      | 86.04            | 2.65             | 44.37             |
| 0.60      | 74.47            | 2.58             | 40.19             |
| 0.65      | 64.25            | 2.42             | 37.71             |
| 0.70      | 55.42            | 2.01             | 34.41             |
| 0.75      | 47.39            | 1.77             | 30.73             |
| 0.80      | 40.53            | 2.09             | 27.25             |
| 0.85      | 34.35            | 2.30             | 23.93             |
| 0.90      | 26.45            | 2.08             | 18.83             |

Table 14: Occupancy Fraction Comparison. Table registering values of \( \ell^1 \)-norms of arrays \( \{S(\text{eff}, P_{\text{alg}}, o) : \text{eff} \in \mathcal{E}, h = 1, 2, \ldots, \hat{h}\} \) for \( o \in \text{full domain} \), \( P_{\text{alg}} \in \{\text{hlt, blT}\} \) and \( \text{dist} \in \{\text{Ub, Bd, Bd}\} \). The remaining parameters are \( r = 54, s = \gamma \), Left Head Fraction \( f = 0.35 \) for \( \text{dist} \in \{\text{Ub, Bd}\} \), \( f = 0.65 \) for \( \text{dist} = \text{Pd} \) (if \( P_{\text{alg}} = \text{hlt} \)) and Minimum List Size \( m = 4 \) i.e., \( P = (512, \text{dist}, o, 54, T_{\text{alg}}, \gamma, f \in \{0.35, 0.65\}) \). Observe that in all the instances of the problems, the blT method gives better results than the hlt.

5.3. Optimal Strategies

In the previous section, it was determined that blT produces better results than hlt. Consequently, from now on, we focus on finding the best values for the remaining parameters: \( o, r \) and \( s \) conditioned to the blT tree generation method.

First we revisit the pruning height of the tree: given that \( \hat{h} \leq h_o(P_{blT}) \) (as introduced in Definition 11(v)) and the analysis is now narrowed down to the blT method, the computations will be done for these heights because is desirable to stretch the D&C method as far as possible but within the quality deterioration control established by \( h_o(P_{blT}) \) (see Definition 11(iv)).

Second, now we analyze the method from two points of view. A global one, as it has been done so far accounting the overall efficiency of the variables in \( \mathcal{E} \) by computing the \( \ell^1 \)-norm of the array \( \{S(\text{eff}, P_{blT}, c) : \text{eff} \in \mathcal{E}, h = 1, 2, \ldots, \hat{h}\} \) as introduced in Definition 11(v). A specialized and second point of view, uses only the \( \ell^1 \)-norm of the array \( \{S(GbE_{DPS}, P_{blT}, c) : h = 1, 2, \ldots, \hat{h}\} \) i.e., regarding only the behavior of the efficiency \( GbE_{DPS} \), through the variables \( o, r \) and \( s \). This specialized measurement is presented because the efficiency of the Dynamic Programming Solution (DPS) is the most important parameter, given that it contains the behavior of the exact solution.

In Table 15 below the \( GbE_{DPS} \) and the global efficiency \( \text{eff} \in \mathcal{E} \) are presented for the occupancy fraction variable \( o \), running through its full domain. The pointing arrows indicate the optimal strategy within its column or family of comparable experiments. As in the previous stage, milar tables were built for the proctor-student rate \( r \in \{34, 44, 54, 64, 74\} \) and the sorting \( s \in \{p, c, \gamma, \text{random}\} \) variables running through their respective full domains which we omit here for the sake of brevity. Finally, in the tables 16 and 17 below we summarize the optimal strategies from both points of view, the specialized \( GbE_{DPS} \) and the global one \( \text{eff} \in \mathcal{E} \).
The pointing arrows indicate the optimal strategy within its column or family of comparable experiments.

6. Conclusions and Final Discussion

The present work yields the following conclusions. The heuristics of the method can be summarized as

(i) We have proposed a Divide and Conquer method to solve the Knapsack Problem at large scale. The method reduces the computational time at the expense of losing quality in the solution. Consequently, the central goal of the paper is to minimize the quality loss by finding the optimal strategies to use the method.

(ii) The deterioration of the solution’s accuracy and/or other parameters of control (such as upper and lower bounds) is defined as the efficiency of the method, and it is the main quantity to assess the quality of the method.

(iii) The method is heuristic therefore, several scenarios need to be explored in order to assess its efficiency. The

Table 15: Occupancy Fraction Comparison. Table registering values of $l^1$-norms of arrays $\{S(\text{eff}, P_{\text{blT}}, o)(h) : \text{eff} \in E, h = 1, 2, \ldots, h_o(P_{\text{blT}})\}$ and $\{S(\text{GbE}_{\text{DPS}}, P_{\text{blT}}, o)(h) : h = 1, 2, \ldots, h_o(P_{\text{blT}})\}$. The values are displayed for $o \in$ full domain and $\text{dist} \in \{\text{Ub}, P_d, \text{Bd}\}$. The remaining parameters are $r = 54$, $s = \gamma$ and Minimum List Size $m = 4$ i.e., $P = (512, \text{dist}, o, 54, \text{blT}, \gamma, 4)$. The pointing arrows indicate the optimal strategy within its column or family of comparable experiments.

Table 16: Chosen Strategies Table. Summary of best strategies. The expected errors are measured with the $l^1$-norms of arrays $\{S(\text{blT}, P_{\text{Talg}}, o)(h) : \text{eff} \in E, h = 1, 2, \ldots, h_o(P_{\text{blT}})\}$, for each of the variables $o \in \{o, r, s\}$. These were used as decision parameters; given that the norms are computed only for the $\text{GbE}_{\text{DPS}}$ efficiency, this point of view only considers the exact solution. The tree generation method is the blT since it was determined as the best tree generation strategy.

Table 17: Summary of Chosen Strategies. In this case the expected errors are measured with the $l^1$-norms of arrays $\{S(\text{eff}, P_{Talg}, o)(h) : \text{eff} \in E, h = 1, 2, \ldots, h_o(P_{\text{blT}})\}$, for each of the variables $o \in \{o, r, s\}$. These were used as decision parameters; given that the norms are computed through all the efficiencies in $E$, this a global point of view. The tree generation method is the blT since it was determined as the best tree generation strategy.
scenarios are modeled using intermediate variables, some deterministic and some probabilistic, e.g. lhT, blT tree generation methods, distribution of capacities \( dist \in \{ Ud, Pd, Ud \} \) respectively, see Figure 6.

(iv) The assessment of strategies is done statistically using random realizations, computing the respective averages and appealing to the Law of Large Numbers to approximate the expected behavior.

From the results point of view

(i) The D&C method can be applied several times to the original KP and generate a tree of subproblems, as those depicted in Figures 1, 2, 3. However, it is not reasonable to branch the problem beyond a certain number of iterations due to the quality deterioration. Such limit is denoted by \( h_v(P_{blT}) \) and it constitutes the first strategy in applying the D&C method within a reasonable range of efficiency.

(ii) Two methods have been introduced to iterate the D&C method, namely lhT, blT. They are compared after a common limit for the branching has been established: \( \tilde{h} \overset{\text{def}}{=} \min\{h_v(P_{blT}), h_v(P_{hlT})\} \). Next, the efficiencies of both methods are compared from three points of view: occupancy fraction (e.g. Table ??), proctor-student rate and sorting method. It follows that the blT furnishes significantly better results than the hlT in most of the possible scenarios.

(iii) Once the blT algorithm has been determined as the best tree generation method, the remaining optimal strategies are searched from two points of view: a specialized one, focused on the exact solution only \( GbE_{DPS} \), and a global one analyzing also the decay of the bounds of control \( GAE, LRE \). The results are summarized in the tables 16 and 17 above.

(iv) As it can be seen, the optimal strategies disagree from one point of view to the other for most of the cases. It is useful to have these information for both cases because in practice, depending on the method to be used in solving the family of subproblems derived from successive applications of the D&C method, it may be more convenient to prioritize one point of view over the other. For instance, if the family of subproblems will be solved using Dynamic Programming, then \( GbE_{DPS} \) is more important. On the other hand, if the method includes bounds control (quantified in \( GAE \) and \( LRE \)) the global point of view may be preferable.

(v) It is also important to stress that in most of the cases \( GbE_{DPS} \) represents, in average, a fraction of 33% of the global efficiency. This shows that when applying the D&C method, the deterioration of the exact solution’s quality is important with respect to the deterioration of the bounds’ quality.

(vi) A paramount feature is that the D&C method deteriorates within reasonable values. In the case of \( GbE_{DPS} \), a maximum expected error of 2.85% is observed. However, such an error occurs after 6 D&C iterations, which drastically reduce the computational time. On the other hand, he global quantification \( eff \in E \), presents a quality decay of 12.29% in the worst case scenario but again, 6 D&C iterations were used and this value encompasses all the efficiencies. It follows that the proposed method is efficient.

The present paper opens up new research lines to be explored in future work

(i) In principle, the reduction of computational time may be considered as 50% per iteration as exposed in Remark 6, however, these estimates were quantified considering a serial algorithm implementation. A parallel implementation, on the other hand would reduce the computational time nearly to 25% because a D&C iteration produces two fully decoupled optimization problems and the post-process merely reduces to paste together the solutions found in the subproblems. The assessment of computational time for a parallel scheme will be pursued in future work.

(ii) As mentioned above, currently a D&C iteration produces two fully decoupled subproblems. However, another scheme with partial coupling can be proposed namely introducing a pair of problems such as those presented in Definition 5(ii), Problem 4 but such that \( A_0 \cup A_1 = \{ N \} \) and \( A_0 \cap A_1 \neq \emptyset \); with assigned demands \( D_0, D_1 \), computed by rules analogous to Equation (18) i.e., construct artificially an integer problem with the structure
Problem 5 ($\Pi^b, b = 0, 1$).

\[
\min \left[ \sum_{b \in \{0,1\}} \sum_{i \in A^b} p_i x_i - \sum_{j \in A^0 \cap A^1} p_j x_j \right],
\]  
subject to

\[
\sum_{i \in A^0} c_i x_i \geq D^0, \quad \sum_{i \in A^1} c_i x_i \geq D^1, \quad x_i \in \{0, 1\}, \forall i \in [N].
\]

A future line of research is the optimal choice of coupling/overlapping sets $A_0 \cap A_1$ and exploit the structure of the integer programming problem 5 (analogously to the Dantzig-Wolfe decomposition for linear problems with the same structure). Furthermore, the optimality has to be analyzed from the perspective quality vs. computing time.

(iii) In this work, the method used a static choice of strategies, e.g., if the sorting method was $s = \gamma$, it remained constant through all the nodes of the D&C tree (as Table 6, Figure 3 illustrate). A future line of research is to investigate the effect of mixing the strategies, e.g., the sorting parameter $s$ taking different values from \{p, c, $\gamma$, random\} from one node to another or from one height (tree level) to the next.

(iv) The bT algorithm is significantly superior to the b1T method; the numerical evidence suggests that an analytic proof of this conjecture is plausible. A future line of research is to look for a rigorous mathematical proof, which of course, would use probability theory and furnish its results in terms of expected efficiencies.

(v) Finally, a future line of research is the implementation and assessment of the D&C method for the optimization of general linear integer programs. However, such a step should be done only once the aforementioned issues have been deeply studied.

Acknowledgments

The Authors wish to thank Universidad Nacional de Colombia, Sede Medellín for supporting the production of this work through the project Hermes 41491, as well as granting access to its supercomputer Gauss (gauss.medellin.unal.edu.co), where the numerical experiments were executed.

References

[1] Raul Baños, Consolación Gil, Julio Ortega, and Francisco G Montoya. Multilevel heuristic algorithm for graph partitioning. In Workshops on Applications of Evolutionary Computation, pages 143–153. Springer, 2003.
[2] Dimitris Bertsimas and John N. Tsitsiklis. Introduction to Linear Optimization. Athena Scientific and Dynamic Ideas, LLC, Belmont, MA, 1997.
[3] Patrick Billingsley. Probability and Measure. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, Inc., New York, 1995.
[4] Christian Blum, Jakob Puchinger, Günther R Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization: A survey. Applied Soft Computing, 11(6):4135–4151, 2011.
[5] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM computing surveys (CSUR), 35(3):268–308, 2003.
[6] Jonathan L. Gross and Jay Yellen. Graph Theory and Its Applications. Discrete Mathematics and its Applications. Chapman & Hall/CRC. Taylor & Francis Group, Boca Raton, FL, 2006.
[7] Angel A Juan, Javier Faulin, Scott E Grasman, Markus Rabe, and Gonçalo Figueira. A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. Operations Research Perspectives, 2:62–72, 2015.
[8] Hans Kellerer, Ulrich Pferschy, and David Pisinger. Knapsack Problems. Discrete Mathematics and its Applications. Springer, Berlin, Heidelberg, New York, 2004.
[9] Edward Yu-Hsien Lin. A bibliographical survey on some well-known non-standard knapsack problems. INFOR: Information Systems and Operational Research, 36(4):274–317, 1998.
[10] Silvano Martello. Knapsack problems: algorithms and computer implementations. Wiley-Interscience series in discrete mathematics and optimization, 1990.
[11] David Pisinger. Where are the hard knapsack problems? Computers & Operations Research, 32(9):2271–2284, 2005.
[12] S Voss, V Maniezzo, and T Stützle. Matheuristics: Hybridizing metaheuristics and mathematical programming (annals of information systems). 2009.

[13] Chris Walshaw. Multilevel refinement for combinatorial optimisation: Boosting metaheuristic performance. In Hybrid Metaheuristics, pages 261–289. Springer, 2008.

[14] Christophe Wilbaut, Said Hanafi, and Said Salhi. A survey of effective heuristics and their application to a variety of knapsack problems. IMA Journal of Management Mathematics, 19(3):227–244, 2008.