Yugoslav Journal of Operations Research
33 (2023), Number 3, 341–366
DOI: https://doi.org/10.2298/YJOR2111015030M

APPLICATION AND ASSESSMENT OF DIVIDE-AND-CONQUER-BASED HEURISTIC ALGORITHMS FOR SOME INTEGER OPTIMIZATION PROBLEMS

Fernando A. MORALES
Escuela de Matemáticas, Universidad Nacional de Colombia, Sede Medellín.
Carrera 65 #59A–110, Medellín, Colombia
famoralesj@unal.edu.co

Received: November 2021 / Accepted: November 2022

Abstract: In this paper three heuristic algorithms using the Divide-and-Conquer paradigm are developed and assessed for three integer optimizations problems: Multidimensional Knapsack Problem (d-KP), Bin Packing Problem (BPP) and Travelling Salesman Problem (TSP). For each case, the algorithm is introduced, together with the design of numerical experiments, in order to empirically establish its performance from both points of view: its computational time and its numerical accuracy.

Keywords: Divide-and-conquer method, multidimensional knapsack problem, bin packing problem, traveling salesman problem, Monte Carlo simulations, method's efficiency.

MSC: 90C10, 68Q87.

1. INTRODUCTION

Broadly speaking, the Divide-and-Conquer method for Integer Optimization problems aims to reduce the computational complexity of the problem, at the price of loosing accuracy on the solutions. So far, it has been successfully introduced in [1] by Morales and Martínez for solving the 0-1 Knapsack Problem (0-1KP). Its success lies on the fact that it is a good **balance** between computational complexity and solution quality. It must be noted that the method does not **compete** with existing algorithms, it **complements** them. In addition, the method is ideally suited for parallel implementation, which is crucial in order to attain its full advantage. The aim of this paper is to extend the Divide-and-Conquer method and asses its performance to three Integer Optimization problems: Multidimensional Knapsack Problem (d-KP), Bin Packing Problem (BPP) and the Traveling Salesman Problem (TSP).

Throughout the problems presented here (as well as 0-1KP in [1]), the Divide-and-Conquer method consists in dividing an original integer optimization problem Π, in two subproblems, namely Π_{left}, Π_{right}, using a **greedy function** as the main criterion for such subdivision. The process can be recursively iterated several times on the newly generated subproblems, creating a tree whose nodes are subproblems and whose leaves have an adequate (previously decided) size. The latter are the subproblems to be solved, either exactly or approximately by an **oracle algorithm** (an existing algorithm) chosen by the user according to the needs. Finally, a **reassemble process** is done on the aforementioned tree to recover a feasible solution for the original problem Π; this is the solution delivered by the method.

The 0-1KP has been extensively studied from many points of view, in contrast d-KP has been given little attention despite its importance. Among the heuristic methods for approximating d-KP the papers [2, 3, 4] work on greedy methods, see [5, 6] for heuristics based on the dual simplex algorithm (both pursuing polynomial time approximations), see [7] for a constraint-relaxation based strategy, see [8] for a Lagrange multipliers approach and [9] for a genetic algorithm approach. Finally, the generalized multidemand multidimensional knapsack problem is treated heuristically in [10] using tabu search principles to detect cutting hyperplanes. As important as all these methods are, it can be seen that none of them introduces the Divide-and-Conquer principle; matter of fact, this paradigm was introduced to the 0-1KP for the first time in [1]. It follows that the strategy introduced in Section 2 is not only new (to the Author's best knowledge) but it is complementary to any of the aforementioned methods.

The BPP ranks among the most studied problems in combinatorial optimization. Most of the efforts are focused on describing its approximation properties (see [11, 12]) or finding lower bounds for the worst case performance ratio of the proposed heuristic algorithms (see [13, 14]). It is important to stress that among the many heuristics introduced to solve BPP, the algorithm presented in [15] follows the Divide-and-Conquer strategy, although it uses a harmonic partition of the items; which is different from the partition presented in Section 3. Yet again, the analysis presented by the Authors concentrates on lower bounds determination for the worst case scenario. Furthermore, from the worst case performance point of view, the classic Best Fit and First Fit algorithms have the same behavior as a large class of packing rules. It follows that a expected performance ratio analysis is needed in order to have more insight on the algorithm's performance; however, little attention has been given in this direction (see [16]), which is the target of our analysis here. As before the Divide-and-Conquer method combines perfectly with any of the preexisting ones, although the analysis here will be limited to its interaction with the most basic algorithms.

Finally, from the perspective of the TSP, the Divide-and-Conquer method is included in the family of heuristics reducing the size of the problem using some criteria, fixing edges [17], introducing multiple levels of reduction [18, 19] or the introduction of sub-tours from a previously known one [20]. The sub-problems are solved using a known algorithm (or oracle) whether exact [21, 22, 23] or approximate [24, 25]. Next, a merging tour algorithm [26, 27] is used to build a global tour. The present work applied to TSP in Section 4 has the structure just described.

The paper is organized as follows. In the rest of this section the notation is introduced, as well as the common criteria for designing the numerical experiments. Each of the analyzed problems has a separate section, hence Section 2 is dedicated for d-KP, Section 3 for BPP and Section 4 is devoted to TSP. In the three cases, the section starts with a brief review of the problem, continues applying the Divide-and-Conquer principle for the problem at hand, next it moves to the corresponding numerical experiments (description and results) and closes analyzing the obtained experimental results. Finally, Section 5 delivers the conclusions and closing discussion of the work.

1.1. Notation

In this section the mathematical notation is introduced. For any natural number $N \in \mathbb{N}$, the symbol $[N] \overset{\textbf{def}}{=} \{1, 2, \ldots, N\}$ indicates the sorted set of the first N natural numbers. A particularly important set is $\mathcal{S}_N$, where $\mathcal{S}_N$ denotes the collection of all permutations in $[N]$. For any real number $x \in \mathbb{R}$ the floor and ceiling functions are given (and denoted) by $\lfloor x \rfloor \overset{\textbf{def}}{=} \max\{k : k \leq x, k \text{ integer}\}$, $\lceil x \rceil \overset{\textbf{def}}{=} \max\{k : k \geq x, k \text{ integer}\}$, respectively. Given an instance of a problem the symbols z_{DC} and z^* indicate the optimal solution given by the presented Divide-and-Conquer algorithm and the optimal solution respectively. The analogous notation is used for T_{DC}, T^* for the corresponding computational times.

1.2. Numerical Experiments Design

The numerical experiments are aimed to asses the performance of the Divide-and-Conquer algorithms presented here. Our ultimate goal is to reliably compute the expected solution's accuracy and the computational time furnished by the method. To that end the probabilistic approach is adopted, whose construction is based on the Law of Large Numbers and the Central Limit Theorem (which is written below for the sake of completeness). The theorem 1 assures the convergence of the random variables while the theorem 2 yields the number of necessary trials, in order to assure that our computed (averaged) values, lie within a confidence interval centered at the actual expected value.

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

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

i.e., the sequence $\big(\mathbf{Z}^{(n)} : n \in \mathbb{N}\big)$ converges in probability to its expectation $\mathbb{E}(\mathbf{Z}^{(1)})$, in the Cesàro sense.

Proof. The proof and details can be found in [28]. □

Theorem 2. *Let x be a scalar statistical variable with mean $\bar{x}$ and variance σ^2.*

*(i) The **number of trials** necessary to get a 95% confidence interval is given by*

$$n \overset{\textbf{def}}{=} \Big(\frac{1.96}{0.05}\Big)^2 \sigma^2. \tag{2}$$

*(ii) The **95 percent confidence interval** is given by*

$$I_x \overset{\textbf{def}}{=} \left[\bar{x} - 1.96\sqrt{\frac{\sigma^2}{n}}, \bar{x} + 1.96\sqrt{\frac{\sigma^2}{n}}\right]. \tag{3}$$

Proof. The proof is based on the Central Limit Theorem, see [29] for details. □

1.2.1. A note about the Software Implementation

For each of the problems addressed in this paper, codes were implemented in order to asses experimentally the quality of the proposed Divide-and-Conquer based algorithms. In all the cases, parallel implementation was developed in order to take the full advantage of the method at hand.

Modified versions of the codes used for the experiments used in this work are made available for the reader in the link `https://drive.google.com/file/d/1oYnezHqXO5yYSFNF_MopwutxTv25mvP_/view?usp=sharing`

The online available codes only simulate parallel implementation. This, because parallel implementation largely depends on hardware, computer's architecture as well as software environment. Therefore, only a modified version of the codes, simulating parallel implementation is furnished so that the interested reader can run in his/her computer and/or modify it into a tailored parallel implementation.

All of the codes were implemented in Python 3.6, they use libraries such as MIP and python_tsp. The link above contains a compressed folder with three sub-folders, one for each problem: d-KP, BPP and TSP. The first two folders contain three codes (one master, two subordinate), the third contains two codes (one subordinate, one master). The executable codes are: • Master_dKP.py, • Master_BPP.py and • Master_TSP.py respectively. They all run from command line, each has on its header a description of the input as well as the output files. In addition, by executing "Master_*.py -h" (where $* \in$ {dKP, BPP, TSP}), a help menu is displayed.

2. THE MULTIDIMENSIONAL KNAPSACK PROBLEM (d-KP)

In the current section the Multidimensional Knapsack Problem (d-KP) is recalled and a Divide-and-Conquer based algorithm, devised for this problem is presented. The technique uses a strategic choice of efficiency coefficients.

Problem 3 (Multidimensional Knapsack Problem, d-KP). *Consider the problem*

$$z^* \overset{\textbf{def}}{=} \max \sum_{j\,=\,1}^{N} p(j)\, x(j), \tag{4a}$$

subject to

$$\sum_{j\,=\,1}^{N} w(i,j)\, x(j) \leq c(i), \qquad \textit{for all } i \in [D]. \tag{4b}$$

$$x(j) \in \{0,1\}, \qquad \textit{for all } j \in [N]. \tag{4c}$$

Here, $\big(c(i)\big)_{i=1}^{D}$ *is the list of* ***knapsack capacities****,* $\big\{\big(w(i,j)\big)_{j=1}^{N}, i \in [D]\big\}$ *is the list of corresponding weights and* $\big(x(j)\big)_{j=1}^{N}$ *is the list of binary-valued* ***decision variables****. In addition, the* ***weight coefficients*** $\big\{\big(w(i,j)\big)_{j=1}^{N}, i \in [D]\big\}$*, as well as the knapsack capacities* $\big(c(i)\big)_{i=1}^{N}$ *are all positive integers. In the following* $z^*, \mathbf{x}^*$ *denote the* ***optimal*** *objective function value and solution respectively. The parameters* $\big(p(j)\big)_{j=1}^{N} \subseteq (0,\infty]^N$ *are known as the* ***profits****. Finally, in the sequel one of the d-KP instances will be denoted by* $\Pi = \big\langle (c(i))_{i\in[D]}, (p(j))_{j\in[N]}, \big\{w(i,j) : j \in [N], i \in [D]\big\}\big\rangle$.

The following efficiency coefficients are introduced, they are scaled according to the respective constraint capacities

Definition 4 (d-KP efficiency coefficients). *For any instance of d-KP (Problem 3), the* ***efficiency coefficients*** *are defined by*

$$g(j) \overset{\textbf{def}}{=} \frac{p(j)}{\sum\limits_{i\,=\,1}^{D} \dfrac{w(i,j)}{c(i)}}, \qquad j \in [N]. \tag{5}$$

Before continuing the analysis, the next hypothesis is adopted.

Hypothesis 5. *In the sequel it is assumed that the instances* Π *of the d-KP satisfy the following conditions*

$$w(i,j) \leq c(i), \qquad \textit{for all } i \in [N], j \in [D], \tag{6a}$$

$$\sum_{j\,=\,1}^{N} w(i,j) > c(i), \qquad \textit{for all } i \in [D]. \tag{6b}$$

Remark 6 (d-KP Setting). *The condition* (6) *in* Hypothesis *5 guarantees two things. First, every item is eligible to be chosen (Inequality* (6a)*). Second, none of the constraints can be dropped (Inequality* (6b)*).*

2.1. A Divide-and-Conquer Approach

A Divide-and-Conquer method for solving the 0-1KP was introduced in [1]. There, an extensive discussion (both, theoretical and empirical) was presented on the possible strategies to implement it. Here, its conclusions are merely adjusted to yield the following method

Definition 7 (A Divide-and-Conquer Algorithm for d-KP). *Let* $\Pi = \big\langle (c(i))_{i\in[D]}, (p(j))_{j\in[N]}, \big\{w(i,j) : j \in [N], i \in [D]\big\}\big\rangle$ *be an instance of* PROBLEM *3*

(i) Sort the items in decreasing order according to the efficiency coefficients and re-index them so that

$$g(1) \geq g(2) \geq \ldots \geq g(N). \tag{7}$$

(ii) Define $V_{\text{left}} = \{j \in [N] : j \text{ is odd}\}$, $V_{\text{right}} = \{j \in [N] : j \text{ is even}\}$ *and*

$$c_{\text{left}}(i) = \left\lceil \frac{\sum_{j\in V_{\text{left}}} w(i,j)}{\sum_{j=1}^{n} w(i,j)} \right\rceil, \quad c_{\text{right}}(i) = c(i) - c_{\text{left}}(i), \quad \textit{for all } i \in [D].$$

(iii) A Divide-and-Conquer pair of PROBLEM *3 is the couple of subproblems* $\big(\Pi_{\text{side}} : \text{side} \in \{\text{left}, \text{right}\}\big)$, *each with input data* $\Pi_{\text{side}} = \big\langle (c_{\text{side}}(i))_{i\in[D]}, \big(p(j)\big)_{j\,\in\,V_{\text{side}}}, \big(w(i,j)\big)_{j\,\in\,V_{\text{side}}}\big\rangle$. *In the sequel,* $\big(\Pi_{\text{side}}, s = \text{left}, \text{right}\big)$ *is referred as a* ***D&C pair*** z^{*}_{side} *denotes the optimal solution value of the problem* Π_{side}.

(iv) The D&C solution is given by

$$\mathbf{x}^{\text{mt}}_{\text{DC}} \overset{\textbf{def}}{=} \mathbf{x}^{\text{mt}}_{\text{left}} \cup \mathbf{x}^{\text{mt}}_{\text{right}}, \qquad\qquad z^{\text{mt}}_{\text{DC}} \overset{\textbf{def}}{=} z^{\text{mt}}_{\text{left}} + z^{\text{mt}}_{\text{right}}. \tag{8}$$

Here, the index mt *indicates the method used to solve the problem, in this work and particular problem, only* $\text{mt} = *$ *is used. Also, some abuse of notation is introduced, denoting by* $\mathbf{x}^{\text{mt}}_{\text{side}}$ *the solution of* Π_{side} *and using the same symbol as a set of chosen items (instead of a vector) in the union operator. In particular, the maximal possible value occurs when all the summands are at its max i.e., the method approximates the optimal solution by the feasible solution* $\mathbf{x}^{\text{mt}}_{\text{DC}} = \mathbf{x}^{\text{mt}}_{\text{left}} \cup \mathbf{x}^{\text{mt}}_{\text{right}}$ *with objective value* $z^{\text{mt}}_{\text{DC}} = z^{\text{mt}}_{\text{left}} + z^{\text{mt}}_{\text{right}}$.

Example 8 (Divide-and-Conquer Algorithm on d-KP). *Consider the d-KP instance described by the table 1, with knapsack capacities* $c(1) = 16, c(2) = 11$ *and number of items* $N = 6$.
In this particular case the D&C pair is given by

$$\begin{array}{llll}
\Pi_{\text{left}} : & V_{\text{left}} = [3,4,5], & c_{\text{left}}(1) = 8, & c_{\text{left}}(2) = 5, \\
\Pi_{\text{right}} : & V_{\text{right}} = [2,1,6], & c_{\text{right}}(1) = 8, & c_{\text{right}}(2) = 6.
\end{array}$$

Table 1: d-KP problem of EXAMPLE 8, knapsack capacities $c(1) = 16, c(2) = 11$, number of items $N = 6$, $D = 2$

j	1	2	3	4	5	6
$p(j)$	5	11	11	71	2	2
$w(1,j)$	6	7	1	7	7	4
$w(2,j)$	4	1	1	6	1	8
$g(j)$	13.53	41.01	144.03	14.28	7.46	4.14

Proposition 9. *Let* $\Pi = \big\langle (c(i))_{i\in[D]}, (p(j))_{j\in[N]}, \big\{w(i,j) : j \in [N], i \in [D]\big\}\big\rangle$ *be an instance of d-KP (problem 3). Let* $(\Pi_{\text{left}}, \Pi_{\text{right}})$ *be the associated Divide-and-Conquer pair introduced in Definition 7. Then,*

$$\mathbf{x}_{\text{DC}}^{\text{mt}} = \mathbf{x}_{\text{left}}^{\text{mt}} \cup \mathbf{x}_{\text{right}}^{\text{mt}} \textit{ is } \Pi\textit{-feasible}, \qquad \textit{for } \text{mt} = \text{G}, *, \text{LR}, \tag{9a}$$

$$z_{\text{left}}^{\text{mt}} + z_{\text{right}}^{\text{mt}} = z_{\text{DC}}^{\text{mt}} \leq z^{\text{mt}}, \qquad \textit{for } \text{mt} = *, \text{LR}. \tag{9b}$$

Here, G *and* LR *indicate any greedy method and* LR *the linear relaxation.*

Proof. Trivial. □

As already stated, the purpose of this work is to determine experimentally, under which conditions an instance Π of d-KP can be solved using the proposed method in an justifiable way. That is, if the trade between computational complexity and accuracy is acceptable. To this end, a set of parameters is introduced, which has proved to be useful in order to classify the d-KP instances, within these classifications the output of numerical methods is comparable, see [30].

Definition 10 (Tightness Ratios). *Let* $\Pi = \big\langle (c(i))_{i\in[D]}, (p(j))_{j\in[N]}, \big\{w(i,j) : j \in [N], i \in [D]\big\}\big\rangle$ *be an instance of d-KP, then the set of tightness ratios is given by*

$$t(i) \overset{\textbf{def}}{=} \frac{c(i)}{\sum_{j=1}^{N} w(i,j)}, \qquad \textit{for all } i \in [D]. \tag{10}$$

(Observe the differences with respect to the efficiency coefficients introduced in Definition 4.)

2.2. Numerical Experiments

In order to numerically asses the method's effectiveness on d-KP, the numerical experiments are designed according to its main parameters i.e., size ($N \in \{6, 10, 20, 50, 100, 250, 500, 1000\}$), restrictiveness ($D \in \{2, 3, 4, 5, 6\}$) and tightness ($t = \{0.25, 0.5, 0.75\}$). While the values of the first three parameters N, D, t were decided based on experience (see [30]), the number of trials $k = 1000$, was decided based on EQUATION (2). The variance σ^2 was chosen as the worst possible value for a sample of 100 trials, which delivered an approximate value $k = 1000$. (After, executing the 1000 experiments the variance showed consistency with the initial value computed from the 100 trial-sample.)

The random instances $\Pi = \big\langle (\mathbf{C}(i))_{i\in[D]}, (\mathbf{P}(j))_{j\in[N]}, \big\{\mathbf{W}(i,j) : j \in [N], i \in [D]\big\}\big\rangle$ were generated as follows.

a. The profits $(\mathbf{P}(i))_{i\in[N]}$ will be independent random variables with uniform distribution on the interval $[1, N\cdot D]$ (i.e., $\mathbb{P}\big(\mathbf{P}(i) = \ell\big) = \frac{1}{N\cdot D}$ for all $\ell \in [1, N\cdot D]$ and $i \in [N]$).

b. Initial capacities $(\mathbf{C}^{(0)}(i))_{i\in[D]}$ are generated as independent random variables uniformly distributed on the interval $[1, N\cdot D]$.

c. For all $i \in [D]$, the weights $(\mathbf{W}(i,j))_{j\in[N]}$ are almost independent random variables, uniformly distributed in the interval $[1, \mathbf{C}^{(0)}(i)]$. Next, the capacities are computed using $\mathbf{C}_i = t(i)\sum_{j=1}^{N}\mathbf{W}(i,j)$, where $t(i)$ is the tightness ratio introduced in Definition 10 and decided a-priori.

Definition 11 (d-KP Performance Coefficients). *The d-KP performance coefficients are given by the percentage fractions of the numerical solution and the computational time given by the proposed algorithm with respect to the exact solution when using the same method of resolution for both cases, i.e.,*

$$S_f = 100 \times \frac{z_{\mathrm{DC}}}{z^*}, \qquad\qquad T_f = 100 \times \frac{T_{\mathrm{DC}}}{T^*}. \tag{11}$$

The exact solution of the d-KP problem was computed using the public Python library MIP, i.e., in our case the MIP subroutine is playing the role of the oracle solving the subproblems. The tables 2, 3, 4 below, summarize the expected values of the performance coefficients S_f (solution fraction) and T_f (time fraction) (see Definition 11) for the tightness values $t = 0.25, 0.5, 0.75$ respectively and dimensions $D = 2, 4, 6$. The corresponding graphs are depicted in the figures 1, 2 and 3.

2.3. Analysis for the d-KP Results

It can be seen that the numerical quality of the solution S_f improves as the number of items grows, getting very close to the optimal solution. As discussed in [31] and shown in [1] this is a phenomenon to be expected in the classical 0-1 Knapsack Problem (0-1KP), because instances with a large number of items have better chances of containing "small" items which easily fit where they are needed. Hence, it is natural to expect a similar behavior for d-KP when approached with an algorithm inspired in the same Divide-and-Conquer technique.

On the other hand, the time fraction performance T_f it can be seen that the computational time approaches 50% when implemented in parallel. This is to be expected, given that the size of the instances reduces in exactly the same proportion.

Table 2: Performance table for the proposed method, solving d-KP problem 3. Tightness ratio $t = 0.25$. Number of trials $k = 1000$. The performance coefficients S_f and T_f are introduced in Equation (11)

Items	D = 2		D = 4		D = 6	
N	S_f	T_f	S_f	T_f	S_f	T_f
6	51.85	63.25	8.39	46.87	33.18	51.28
10	76.39	54.63	60.88	54.42	70.75	48.34
20	88.98	54.13	82.84	51.43	87	53.65
50	96.36	51.34	93.69	52.39	95.69	51.96
100	98.67	52.09	97.2	51.49	97.53	51.84
250	99.63	53.47	99.03	51.65	99.01	51.86
500	99.84	51.63	99.55	50.9	99.47	51.07
1000	99.94	51.41	99.81	50.23	99.77	50.52

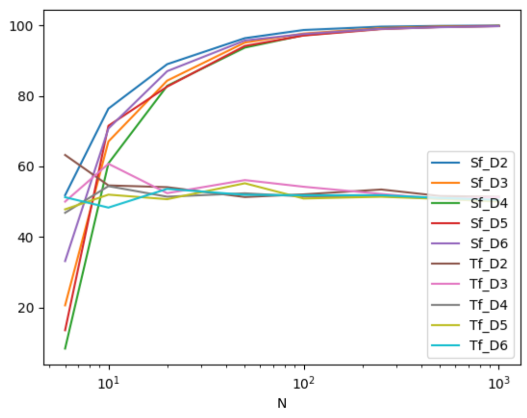


Figure 1: Graphic solution fraction S_f and time fraction T_f for the values displayed in table 2. The domain is given by the number of items N in logarithmic scale

It must be noted that this is a Greedy Heuristic algorithm and, as pointed out in [32] (Chapter 9) no quality certificates (relative performance) were derived for this family of algorithms.

3. THE BIN PACKING PROBLEM BPP

In the current section the Bin Packing Problem (BPP) is introduced and a list of greedy algorithms is reviewed; these will be used in the following to compute/measure the expected performance of a Divide-and-Conquer based proposed method acting on BPP.

The Bin-Packing Problem is described as follows. Given N objects, each of a given weight $(w(j))_{j=1}^{N} \subseteq (0, c]$ and bins of capacity c (at least N of them), the

Table 3: Performance table for the proposed method, solving d-KP problem 3. Tightness ratio $t = 0.5$. Number of trials $k = 1000$. The performance coefficients S_f and T_f are introduced in Equation (11)

Items	D = 2		D = 4		D = 6	
N	S_f	T_f	S_f	T_f	S_f	T_f
6	78.62	54.42	75.57	56.41	68.54	57.77
10	88.69	53.79	84.35	50.98	81.03	52.51
20	94.49	52.57	92.56	49.24	90.45	53.64
50	98.35	52.44	96.96	51.44	96.27	51.3
100	99.2	50.91	98.51	52.66	98.07	51.03
250	99.74	51.95	99.43	51.05	99.21	50.77
500	99.89	50.49	99.73	51.29	99.58	50.57
1000	99.94	50.98	99.89	51.13	99.8	50.72

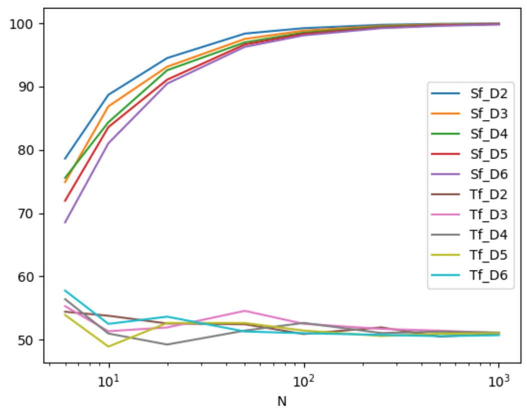


Figure 2: Graphic solution fraction S_f and time fraction T_f for the values displayed in table 3. The domain is given by the number of items N in logarithmic scale

aim is to assign the objects in a way that the minimum number of bins is used and that every object is packed in one of the bins. Without loss of generality, the capacity of the items can be normalized to $c = 1$ and the weight of the items accordingly. Hence, the BPP formulates in the following way

Problem 12 (Bin Packing Problem, BPP). *Consider the problem*

$$z^* \overset{\textbf{def}}{=} \min \sum_{i=1}^{N} y(i), \tag{12a}$$

Table 4: Performance table for the proposed method, solving d-KP problem 3. Tightness ratio $t = 0.75$. Number of trials $k = 1000$. The performance coefficients S_f and T_f are introduced in EQUATION (11)

Items	D = 2		D = 4		D = 6	
N	S_f	T_f	S_f	T_f	S_f	T_f
6	90.76	52.83	84.26	52.76	75.83	48.85
10	93.42	51.71	90.91	52.77	89.93	50.96
20	97.46	50.83	96.16	54.96	95.41	51.21
50	99.04	52.87	98.63	51.7	98.24	50.37
100	99.57	52.33	99.33	52.34	99.13	51.59
250	99.83	51.44	99.7	50.77	99.65	51.54
500	99.94	51.11	99.86	50.67	99.8	51.1
1000	99.97	51.14	99.94	51.16	99.9	50.38

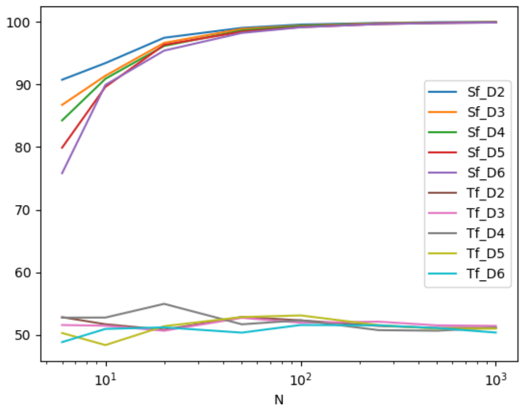


Figure 3: Graphic solution fraction S_f and time fraction T_f for the values displayed in table 4. The domain is given by the number of items N in logarithmic scale

subject to

$$\sum_{j=1}^{N} w(j)\, x(i,j) \le y(j), \qquad \textit{for all } i \in [N]. \tag{12b}$$

$$x(i,j), y(i) \in \{0,1\}, \qquad \textit{for all } i, j \in [N]. \tag{12c}$$

*Here, $\big(w(j)\big)_{j=1}^{N} \subseteq (0,1]$ is the list of **item weights**, $\big\{x(i,j) : i,j \in [N]\big\}$ is the list of binary valued **item assignment decision variables**, with $x(i,j) = 1$ if item j is assigned to bin i and $x(i,j) = 0$ otherwise. Finally, $\big(y(i)\big)_{i=1}^{N}$ is the list of binary valued **bin-choice decision variables**, setting $y(i) = 1$ if bin i is nonempty and $y(i) = 0$ otherwise.*

As it is well-know, most of the literature is concerned with heuristic methods for solving BPP (see [16]). Therefore, this work is focused on the interaction of a Divide-and-Conquer paradigm with the three most basic heuristic algorithms: Next Fit Decreasing (NFD), First Fit Decreasing (FFD) and Best Fit Decreasing (BFD) (see [31] and [33]). For the sake of completeness these algorithms are described below. In the three cases, it is assumed that the items are sorted decreasingly according their weights, i.e. $w(1) \geq w(2) \geq \ldots \geq w(N)$.

a. **Next Fit Algorithm.** The first item is assigned to bin 1. Each of the items $2, \ldots, N$ is handled as follows: the item j is assigned to the current bin if it fits; if not it is assigned to a new bin (next fit).

b. **First Fit Algorithm.** Consider the items increasingly according to its index. For each item j, assign it to the first initialized bin i (first fit) where it fits (if any), and if it does not fit in any initialized bin, assign it to a new one.

c. **Best Fit Algorithm.** Consider the items increasingly according to its index. For each item j, assign it to the bin i where it fits (if any) and it is as full as possible; if it does not fit in any initialized bin, assign it to a new one.

3.1. The Divide-and-Conquer Approach

The NF, FF and BF algorithms work exactly as NFD, FFD and BFD, except that they do not act on a sorted list of items. It has been established that the NF algorithm (and consequently FF as well as BF) has good performance on items of small weight (see [33]). Therefore, it is only natural to deal with the pieces according to their weight in decreasing order i.e., in the BPP problem the weights are used as greedy function (or efficiency coefficients) in order to propose the corresponding Divide-and-Conquer heuristic

Definition 13 (Divide-and-Conquer Method for BPP). *Let* $\Pi = \big\langle (w(j))_{j\in[N]} \big\rangle$ *be an instance of* PROBLEM *12*

(i) Sort the items in decreasing order according to their weights and re-index them so that

$$w(1) \geq w(2) \geq \ldots \geq w(N). \tag{13}$$

(ii) Define $V_{\text{left}} = \{j \in [N] : j \textit{ is odd}\}$ *and* $V_{\text{right}} = \{j \in [N] : j \textit{ is even}\}$.

(iii) A Divide-and-Conquer pair of PROBLEM *12 is the couple of subproblems* $\big(\Pi_{\text{side}} : \text{side} \in \{\text{left}, \text{right}\}\big)$, *each with input data* $\Pi_{\text{side}} = \big\langle (w(j))_{j\,\in\,V_{\text{side}}} \big\rangle$. *In the sequel,* $\big(\Pi_{\text{side}}, s = \text{left}, \text{right}\big)$ *is referred as a* ***D&C pair****. We denote by* $z^{\text{alg}}_{\text{side}}$ *the solution value of the problem* Π_{side}, *while* z^{alg} *denotes the solution value of the full problem* Π, *furnished by the algorithm* alg.

(iv) The D&C solution of PROBLEM *12 is given by*

$$\mathbf{x}_{\text{DC}}^{\text{alg}} \overset{\textbf{def}}{=} \mathbf{x}_{\text{left}}^{\text{alg}} \cup \mathbf{x}_{\text{right}}^{\text{alg}}, \qquad z_{\text{DC}}^{\text{alg}} \overset{\textbf{def}}{=} z_{\text{left}}^{\text{alg}} + z_{\text{right}}^{\text{alg}}. \tag{14}$$

Here, the index alg *indicates the method used to solve the problem. This work, uses* alg $=$ NFD, FFD, BFD*, standing for the Next Fit Decreasing, First Fit Decreasing and Best Fit Decreasing algorithms respectively (described in the previous section). Also, some abuse of notation is introduced, denoting by* $\mathbf{x}_{\text{side}}^{\text{alg}} = \{x_{\text{side}}^{\text{alg}}(i,j) : i,j \in V_{\text{side}}\}$ *the optimal solution of* Π_{side} *and using the same symbol as a set of chosen items (instead of a vector) in the union operator. In particular, the minimal possible value occurs when all the summands are at its minimum i.e., the method approximates the optimal solution by the feasible solution* $\mathbf{x}_{\text{DC}}^{\text{alg}} = \mathbf{x}_{\text{left}}^{\text{alg}} \cup \mathbf{x}_{\text{right}}^{\text{alg}}$ *with objective value* $z_{\text{DC}}^{\text{alg}} = z_{\text{left}}^{\text{alg}} + z_{\text{right}}^{\text{alg}}$.

Example 14 (Divide-and-Conquer Algorithm on BPP). *Consider the BPP instance described by the table 5, number of items* $N = 6$.

Table 5: BPP problem of EXAMPLE 14, number of items $N = 6$

j	1	2	3	4	5	6
$w(j)$	0.5	0.7	0.25	0.1	0.85	0.31

For this particular case the D&C pair is given by

$$\begin{aligned} &\Pi_{\text{left}} : && V_{\text{left}} = [5, 1, 3], \\ &\Pi_{\text{right}} : && V_{\text{right}} = [2, 6, 4]. \end{aligned}$$

Proposition 15. *Let* $\Pi = \big\langle (w(j))_{j \in [N]} \big\rangle$ *be an instance of the BPP problem 12. Let* $(\Pi_{\text{left}}, \Pi_{\text{right}})$ *be the associated Divide-and-Conquer pair introduced in Definition 13. Then,*

$$\mathbf{x}_{\text{DC}}^{\text{alg}} = \mathbf{x}_{\text{left}}^{\text{alg}} \cup \mathbf{x}_{\text{right}}^{\text{alg}} \text{ is } \Pi\text{-feasible}, \qquad \text{for alg} = \text{NFD, FFD, BFD}. \tag{15}$$

Proof. Trivial. □

Remark 16. *Observe that, unlike* PROPOSITION *9, here, there is no claim of an inequality of control such as* (9b)*. This is because the NFD, FFD and BFD methods are all heuristic. Hence, it can happen (and it actually does for some few instances) that* $z_{\text{DC}}^{\text{alg}} \geq z^{\text{alg}}$ *for* alg $=$ NFD, FFD, BFD*; it is worth noticing that this phenomenon is more frequent for the NFD method.*

3.2. Numerical Experiments

In order to numerically asses the effectiveness of the proposed Divide-and-Conquer algorithm, the experiments are designed according to its main parameter i.e., size ($N \in \{20, 50, 100, 250, 500, 1000, 1500, 2000\}$) and number of trials ($k$). The number of items was decided based on experience (more specifically the

Divide-and-Conquer algorithm yields poor results for low values of N as it can be seen in the results), while the number of trials $k = 1500$ was chosen based on EQUATION (2). The variance σ^2 was chosen as the worst possible value for a sample of 100 trials, which delivered an approximate value $k = 1500$. (After, executing the 1500 experiments the variance showed consistency with the the initial value computed from the 100 trial-sample.)

For the random instances $\Pi = \big\langle (\mathbf{W}(j))_{j\in[N]} \big\rangle$, the weights $(\mathbf{W}(j))_{j\in[N]}$ are independent random variables uniformly distributed on the interval $[0,1]$ (notice that $\mathbb{P}(\mathbf{W}(i) = 0) = 0$ for all $i \in [N]$). Next, the numerical and time performance coefficients are introduced.

Definition 17 (BPP Performance Coefficients). *The BPP performance coefficients are given by the percentage fractions of the solution and the computational time given by the Divide-and-Conquer approach with respect to the exact solution, when using the same method of resolution for both cases, i.e.,*

$$S_f^{\text{alg}} = 100 \times \frac{z^{\text{alg}}}{z_{\text{DC}}^{\text{alg}}}, \quad T_f^{\text{alg}} = 100 \times \frac{T_{\text{DC}}^{\text{alg}}}{T^{\text{alg}}}, \quad \text{alg} = \text{NFD, FFD, BFD}. \tag{16}$$

Here, $z^{\text{alg}}, z_{\text{DC}}^{\text{alg}}$ indicate the solution furnished for the full problem and for the Divide-and-Conquer heuristic respectively, when using the algorithm alg.

Remark 18. *The definition of S_f^{alg} (equation (16)) for BPP differs from that given in d-KP (equation (11)) in order to keep normalized the fraction. Since BPP is a minimization problem the Divide-and-Conquer solution will be larger than the exact solution ($z_{\text{DC}}^{\text{alg}} \geq z^{\text{alg}}$) for most of the cases. This contrasts with d-KP which is maximization problem, where the opposite behavior takes place ($z_{\text{DC}} \leq z^*$).*

The table 6 below, summarizes the expected values of the performance coefficients S_f^{alg} (solution fraction) T_f^{alg} (time fraction) (see DEFINITION 17). The corresponding graphs are depicted in the figures 4 (a) for S_f^{alg} and (b) for T_f^{alg}, respectively.

3.3. Analysis for the BPP Results

In the present section we recall some theoretical results for the FFD algorithm, in order to derive theoretical bounds for the proposed Divide-and-Conquer heuristic when using the previous algorithm as an oracle. The following result is presented in [34]

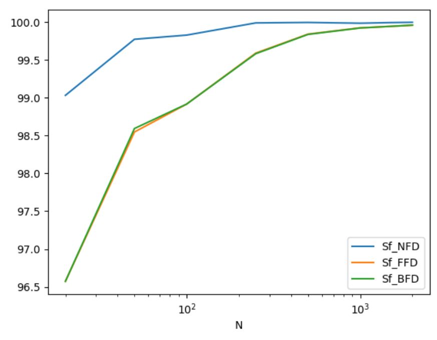
Theorem 19. *Let Π be a BPP instance then*

$$z^{\text{FFD}} \leq \frac{11}{9} z^* + \frac{2}{3}, \tag{17}$$

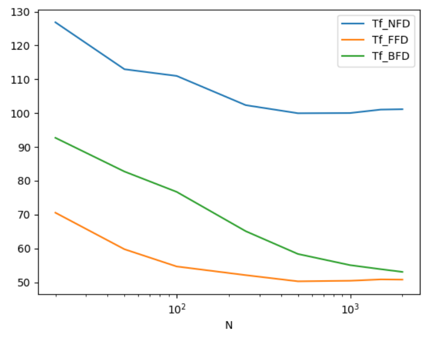
where z^* indicates the optimal solution of the instance Π.

Table 6: Performance table for the Divide-and-Conquer method, when solving the BPP problem with the three methods: NFD, FFD and BFD. Number of trials $k = 1500$. The performance coefficients S_f^{alg} and $T_f^{a\,\text{lg}}$ for alg $= \text{NFD}, \text{FFD}, \text{BFD}$ are introduced in EQUATION (16)

Items	NFD		FFD		BFD	
N	S_f^{NFD}	T_f^{NFD}	S_f^{FFD}	T_f^{FFD}	S_f^{BFD}	T_f^{BFD}
20	99.03	126.86	96.57	70.56	96.57	92.7
50	99.77	112.98	98.55	59.75	98.59	82.73
100	99.83	111.01	98.92	54.66	98.92	76.72
250	99.99	102.36	99.59	52.09	99.58	65.09
500	99.99	99.97	99.84	50.26	99.84	58.35
1000	99.98	100.04	99.92	50.43	99.92	55.04
1500	99.99	101.05	99.94	50.83	99.94	53.85
2000	99.99	101.16	99.96	50.76	99.96	53.05



(a) Solution Fraction S_f Performance



(b) Time Fraction T_f Performance

Figure 4: The graphs above depict the approximate expected performance for $k = 1500$ trials. In figure (a) the solution fraction S_f^{alg}, in figure (b) the time fraction T_f^{alg} coefficients are displayed, for the BPP problem (see DEFINITION 17 and alg $= \text{NFD}, \text{FFD}, \text{BFD}$. The domain N is given in logarithmic scale, the values are extracted from the table 6.

Corollary 20. *Let* Π *be a BPP instance, then*

$$\frac{9}{11}\Big(\frac{z^{\text{FFD}}}{z_{\text{DC}}^{\text{FFD}}} - \frac{2}{3}\Big) \leq \frac{z^*}{z_{\text{DC}}^*}. \tag{18}$$

Proof. Let $(\Pi_{\text{left}}, \Pi_{\text{right}})$ be the D&C pair of Π. Given that FFD is a not exact algorithm it is direct to see that $1 \leq z^* \leq z_{\text{DC}}^* \leq z_{\text{DC}}^{\text{FFD}}$. Hence,

$$\frac{1}{z_{\text{DC}}^{\text{FFD}}} \leq \frac{1}{z_{\text{DC}}^*} \leq 1.$$

Multiplying term-wise the above inequality with (17) we get

$$\frac{z^{\mathrm{FFD}}}{z_{\mathrm{DC}}^{\mathrm{FFD}}} \leq \frac{11}{9}\frac{z^*}{z_{\mathrm{DC}}^*} + \frac{2}{3}\frac{1}{z_{\mathrm{DC}}^*} \leq \frac{11}{9}\frac{z^*}{z_{\mathrm{DC}}^*} + \frac{2}{3}. \tag{19}$$

From here, the result follows directly. □

Remark 21. *The lower bound* (18) *is a useful tool for the worst case analysis of the method. However, we are interested here in the expected behavior. Applying the expectation to both sides of the inequality* (18) *we get*

$$\frac{9}{11}\Big(\mathbb{E}\big(\frac{z^{\mathrm{FFD}}}{z_{\mathrm{DC}}^{\mathrm{FFD}}}\big) - \frac{2}{3}\Big) = \frac{9}{11}\Big(\mathbb{E}(S_f^{\mathrm{FFD}}) - \frac{2}{3}\Big) \leq \mathbb{E}\big(\frac{z^*}{z_{\mathrm{DC}}^*}\big).$$

Recalling from the experiments that $S_f^{\mathrm{FFD}} \to 1$ as $N \to \infty$ (with N being the size of the instance Π), this would give a lower bound for the expected performance of at most $\frac{3}{11}$. The latter value is too low for the expected fraction performance on the exact solutions. This happens because a worst-case analysis was used to derive the inequality (18).

As already pointed out in [33], the NFD, FFD and BFD algorithms have good performance on items of small weight. Hence, as the size of N of the instance grows the uniform distribution of items weight on $[N]$ has wider spectrum, where items of small size complement well items of big size when filling a bin. This phenomenon also reflects upon the D&C pairs of any instance, hence it is natural that expected behavior of the performance fraction S_f tends to 1 as the experiments show.

As for the time fraction performance T_f it has been reported that NFD runs in $\mathcal{O}(N)$ (see theorem 18.4 in [33]). However, T_f shows a really bad behavior which conflicts the initial intuition one has. The underlying reason is that the Divide-and-Conquer algorithm involves other computational costs aside from the solution of the D&C pair. These are: the construction of the D&C pair, the assignment of tasks in parallel, the reassemble of solution and the communication time between parallel nodes. These times, in the case of NFD largely outweigh the NFD-algorithm running times. However, the same times are not as important against the running time of FFD and BFD because they are more sophisticate in their process, as it can be verified in the experiments results.

4. THE TRAVELING SALESMAN PROBLEM TSP

The last problem to be analyzed under a proposed Divide-and-Conquer algorithm is the Traveling Salesman Problem TSP. Some graph-theory definitions are recalled in order to describe it.

Definition 22. *(i) An edge-weighted, directed graph is a triple $G = (V, E, d)$, where V is the set of vertices, $E \subseteq V \times V$ is the set of (directed) edges or arcs and $d : E \to \mathbb{R}^+$ is a distance function assigning each arc $e \in E$ a distance $d(e)$.*

(ii) A path is a list $(u_1, ..., u_k)$ *of vertices* $u_i \in V$ *for all* $i = 1, ..., k$*, holding* $(u_i, u_{i+1}) \in E$ *for* $i = 1, ..., k-1$.

(iii) A Hamiltonian cycle in G *is a path* $p = (u_1, ..., u_k, u_1)$ *in* G *, where* $k = |V|$ *and* $\bigcup_{i=1}^{k} u_i = V$ *(i.e., each vertex is visited exactly once except for* u_1*).*

(iv) Given an edge-weighted directed graph $G = (V, E, d)$ *and a vertex subset* U *of* V*, the induced subgraph, denoted by* $G(U) = (U, E^U, d^U)$ *is the subgraph of* G *whose vertex set is* U*, whose edge set consists on all edges in* G *that have both endpoints on* U *and whose distance function* $d^U = d\big|_{U\times U}$ *is the restriction of the distance function to the set* $U \times U \subseteq V \times V$.

Problem 23 (The Traveling Salesman Problem, TSP). *Given a directed graph* $G = (V, E, d)$ *with* $n = |V|$ *vertices, the TSP is to find a Hamiltonian cycle* $(u_1, \ldots, u_n, u_1)$ *such that the cost* $C(u_1, \ldots, u_n, u_1) = d((u_k, u_1)) + \sum_{i=1}^{k-1} d((u_i, u_{i+1}))$ *(the total sum of the weights of the edges in the cycle) is minimized.*

(i) The TSP is said to be symmetric if $d((u, v)) = d((v, u))$ *for all* $u, v \in V$*. Otherwise, it is said to be asymmetric.*

(ii) The TSP is said to be metric if $d((u, v)) \leq d((u, w)) + d((w, v))$ *for all* $u, v, w \in V$.

In the current work it will be assumed that the graph is complete i.e., all possible edges in both directions are present. In addition, three cases are analyzed:

a. **MS**: when the TSP is metric and symmetric.

b. **MA**: when the TSP is metric but not symmetric.

c. **NMS**: when the TSP is not metric but it is symmetric.

Since it is so widespread and contributes little to our discussion, the formulation of TSP is omitted (see Section 10.3 in [35] for a formulation).

4.1. A Divide-and-Conquer Approach

In order to apply our Divide-and-Conquer approach on the TSP, it is necessary to introduce an efficiency coefficient to break the original problem in two sub-problems. Hence, in the TSP case, the efficiency for each vertex of the graph, is defined as the sum of the distances of all the edges incident on the node.

Definition 24 (TSP efficiency coefficients). *Given an edge-weighted complete directed graph* $G = (E, V, d)$*, for each node* $u \in V$ *its* ***efficiency coefficient*** *is defined as*

$$g(u) \overset{\textbf{def}}{=} \sum_{v \in V - \{u\}} d\big((u, v)\big) + d\big((v, u)\big). \tag{20}$$

Next, a greedy algorithm needs to be introduced in order merge two cycles (the two solution cycles corresponding to each sub-problem: $\Pi_{\text{left}}, \Pi_{\text{right}}$).

Definition 25 (Cycle Greedy Merging). *Let $G = (V, E, d)$ be an edge-weighted complete directed with $|V| = N$. Let $V_{\text{left}}, V_{\text{right}}$ be a subset partition of V with $|V_{\text{left}}| = p$, $|V_{\text{left}}| = q$ and let*

$$C_{\text{left}} = (u_1, ..., u_p, u_1), \qquad\qquad C_{\text{right}} = (v_1, ..., v_q, v_1), \tag{21}$$

be two cycles contained in V_{left} and V_{right} respectively. Let $(u_i, u_{i+1}), (v_j, v_{j+1})$ be the most expensive arcs in C_{left} and C_{right} respectively and break possible ties choosing the lowest index. (Here, the indexes $i, i+1$ and $j, j+1$ are understood in $\mod p$ and $\mod q$ respectively.) Define the merged cycle as

$$C = (u_1, \ldots, u_{i-1}, \underbrace{u_i, v_{j+1}}, v_{j+2}, \ldots, v_q, v_1, \ldots, v_{j-1}, \underbrace{v_j, u_{i+1}}, u_{i+2} \ldots, u_p, u_1). \tag{22}$$

Where the arcs $(u_i, v_{j+1}), (v_j, u_{i+1})$ (marked with underbraces) were included to replace the corresponding arcs (u_i, u_{i+1}) and (v_j, v_{j+1}). (See Figure 5 for an example.)

Remark 26. *It must be observed that the merging process introduced in Definition 25 is well-defined, because the choice of arcs to be removed is unique as well as the inclusion of the new arcs, marked with underbraces in* (22)*. Once the arcs (u_i, u_{i+1}), (v_j, v_{j+1}) are removed, the pair of arcs $(u_i, v_{j+1}), (v_j, u_{i+1})$ are the only possible choice for respective replacement that would deliver a joint cycle, given the orientations of C_{left} and C_{right}. This observation is particularly important for the asymmetric instances of TSP (MA).*

Definition 27 (A Divide-and-Conquer Algorithm for TSP). *Let $\Pi = (V, E, d)$ be an instance of* PROBLEM *23*

(i) Sort the vertices in increasing order according to their efficiencies, that is

$$g(v_{\pi(1)}) \leq g(v_{\pi(2)}) \leq \ldots \leq g(v_{\pi(N)}), \tag{23}$$

where $\pi \in \mathcal{S}_N$ is an adequate permutation and $V = \big\{v_i : i \in [N]\big\} = \big\{v_{\pi(i)} : i \in [N]\big\}$.

(ii) Define $V_{\text{left}} = \{v_{\pi(i)} \in [N] : i \text{ is odd}\}$ and $V_{\text{right}} = \{v_{\pi(i)} \in [N] : i \text{ is even}\}$.

(iii) A Divide-and-Conquer pair of PROBLEM *3 is the couple of subproblems $\big(\Pi_{\text{side}} : \text{side} \in \{\text{left}, \text{right}\}\big)$, each with input data $\Pi_{\text{side}} = G(V_{\text{side}})$, i.e., the induced subgraph introduced in Definition 22. In the following, $\big(\Pi_{\text{side}}, s = \text{left}, \text{right}\big)$ is referred as a* ***D&C pair*** *and denote by z^*_{side} the optimal solution value of the problem Π_{side}.*

(iv) The D&C solution of PROBLEM *23 is given by the greedy merging of cycles $C_{\text{left}}, C_{\text{right}}$ (see Definition 25) furnished by the solution of Π_{left} and Π_{right} respectively. (See Example 28 and Figure 5 for an illustration.)*

Example 28 (Divide-and-Conquer Algorithm on TSP). *Consider the TSP instance described by the table 7, with $N = 6$ vertices. For this particular case the following order holds: $g(v_3) \le g(v_5) \le g(v_2) \le g(v_6) \le g(v_1) \le g(v_4)$. Therefore,*

Table 7: TSP problem of EXAMPLE 28, number of items $N = 6$. Distance matrix table

$v_i \backslash v_j$	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0.00	0.61	0.10	1.08	0.46	0.11
v_2	0.61	0.00	0.53	0.71	0.17	0.54
v_3	0.10	0.53	0.00	0.98	0.39	0.12
v_4	1.08	0.71	0.98	0.00	0.83	1.07
v_5	0.46	0.17	0.39	0.83	0.00	0.38
v_6	0.11	0.54	0.12	1.07	0.38	0.00
$g(v_j)$	4.17	3.93	3.75	5.37	3.77	4.05

$$\begin{aligned} &\Pi_{\text{left}}: && V_{\text{left}} = [v_3, v_2, v_1],\\ &\Pi_{\text{right}}: && V_{\text{right}} = [v_5, v_6, v_4]. \end{aligned}$$

Given that this is a symmetric instance (actually MS) of TSP and that Π_{left} and Π_{right} have both three vertices, there is only one possible solution for each TSP; namely $C_{\text{left}} = (v_1, v_2, v_3)$ and $C_{\text{right}} = (v_4, v_6, v_5)$ respectively. See Figure 5. Next, observe that the most expensive arc in C_{left} is (v_1, v_2), while the heaviest arc in C_{right} is (v_4, v_6). Hence, these two arcs need to be removed (depicted in dashed line in Figure 5) and replaced by (v_1, v_6), (v_4, v_2) (depicted in blue in Figure 5) respectively. Observe that this is the only possible choice in order to preserve the direction of the cycles, in particular, the remaining arcs (depicted in red in Figure 5) would fail to build a global Hamiltonian cycle.

4.2. Numerical Experiments

In order to numerically asses proposed Divide-and-Conquer algorithm's effectiveness on TSP, the numerical experiments are designed according to its main parameters i.e., size ($N \in \{8, 18, 30, 44, 60, 78, 98, 120\}$) and number of trials ($k$). The number of items was decided based on experience and observation of the method's performance; while the number of trials $k = 2500$ was chosen based on EQUATION (2). The variance σ^2 was chosen as the worst possible value for a sample of 150 trials, which delivered an approximate value $k = 2500$. (After, executing the 2500 experiments the variance showed consistency with the initial value computed from the 150 trial-sample.) The exact solution of the TSP problem was computed using the public Python library python-tsp, using dynamic programming i.e., in

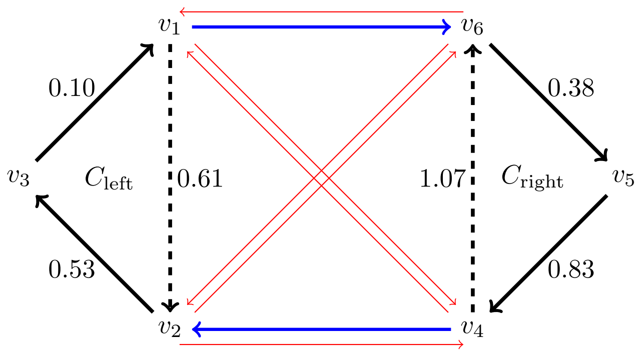


Figure 5: Merging Cycles Schematics for Example 28 (see 25 for a general definition). The cycles $C_{\text{left}} = (v_1, v_2, v_3)$ and $C_{\text{right}} = (v_4, v_6, v_5)$ are to be merged. The most expensive arcs for each, depicted in dashed line are removed and replaced by the only possible pair, in order to build a global Hamiltonian Cycle. Choosing other pair of arcs (depicted in red) would fail to assemble a cycle due to the orientation

our case the solve_tsp_dynamic_programming subroutine is playing the role of the oracle solving the subproblems.

The random instances were be generated as follows

a. For the MS instances of TSP, N points in the square $[0,1] \times [0,1] \subseteq \mathbb{R}^2$ are generated (using the uniform distribution) and compute the distance matrix D_{MS}.

b. For the MA instances of TSP, N points on the unit circle $S^1 = \big\{\vec{x} : |\vec{x}| = 1\big\} \subseteq \mathbb{R}^2$ are generated (using the uniform distribution) and compute the asymmetric distance matrix d_{MA}. Given $\vec{x}, \vec{y} \in S^1$, the asymmetric distance $d_{\text{MA}}(\vec{x}, \vec{y})$ is defined as the length of the shortest clockwise path from $\vec{x}$ to $\vec{y}$ contained in S^1. Clearly $d_{\text{MA}}(\vec{x}, \vec{y}) \neq d_{\text{MA}}(\vec{y}, \vec{x})$, unless $\vec{x}$ and $\vec{y}$ are antipodal points.

c. For the NMS instances of TSP, an $N \times N$ upper triangular matrix M is generated, whose entries are uniformly distributed in $[0,1]$ and its diagonal is null. The distance matrix is defined as $D_{\text{NMS}} = M + M^T$.

The performance coefficients are analogous to those introduced in Definition 17, given that TSP is a minimization problem as BPP (see also Remark 18).

Definition 29 (TSP Performance Coefficients). *The TSP performance coefficients are given by the percentage fractions of the solution and the computational time given by the proposed Divide-and-Conquer algorithm, with respect to the exact solution, when using the same method of resolution for both cases, i.e.,*

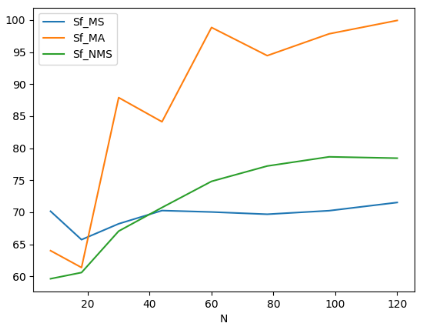
$$S_f^{\text{case}} = 100 \times \frac{z^{\text{case}}}{z_{\text{DC}}^{\text{case}}}, \quad T_f^{\text{case}} = 100 \times \frac{T_{\text{DC}}^{\text{case}}}{T^{\text{case}}}, \quad \text{case} = \text{MS, MA, NMS}. \tag{24}$$

Here, $z^{\text{case}}, z_{\text{DC}}^{\text{case}}$ indicate the solution furnished for the full problem and for the Divide-and-Conquer algorithm respectively, when solving the corresponding case.

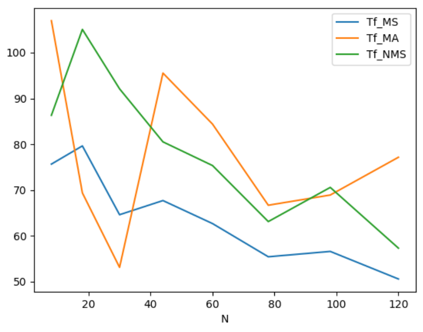
The table 8 below summarizes the expected values of the performance coefficients S_f^{case} (solution fraction) T_f^{case} (time fraction). The corresponding graphs are depicted in the figures 6 (a) for S_f^{case} and (b) for T_f^{case}, respectively.

Table 8: Performance table for the Divide-and-Conquer method, solving the TSP problem for the three types of instances: MS, MA and NMS. Number of trials $k = 2500$. The performance coefficients S_f^{case} and T_f^{case} for case $= \text{MS}, \text{MA}, \text{NMS}$ are introduced in EQUATION (24)

Items	MS		MA		NMS	
N	S_f^{MS}	T_f^{MS}	S_f^{MA}	T_f^{MA}	S_f^{NMS}	T_f^{NMS}
8	70.15	75.66	64	106.98	59.62	86.32
18	65.72	79.64	61.37	69.37	60.59	105.08
30	68.2	64.61	87.9	53.11	67.05	92.1
44	70.26	67.7	84.12	95.54	70.74	80.52
60	70.04	62.68	98.85	84.4	74.82	75.35
78	69.69	55.43	94.45	66.68	77.22	63.11
98	70.24	56.6	97.86	68.91	78.65	70.58
120	71.52	50.57	99.94	77.16	78.44	57.31



(a) Solution Fraction S_f Performance



(b) Time Fraction T_f Performance

Figure 6: The graphs above depict the approximate expected performance for $k = 2500$ trials. In figure (a) the solution fraction S_f^{case}, in figure (b) for the time fraction T_f^{case} coefficients are displayed, for the three types of instance of TSP case $= \text{MS}, \text{MA}, \text{NMS}$. Unlike the previous examples, the domain N is given in linear scale. The values are extracted from the table 8

4.3. Analysis for the TSP Results

In order to discuss the numerical performance fraction (T_f) behavior in the results, we start recalling a classic result for Hamilton cycles, which we prove here for the sake of completeness.

Theorem 30. *For $N \geq 3$ the number of distinct Hamilton cycles in the complete graph K_N is* $\dfrac{(N-1)!}{2}$.

Proof. In a complete graph, every vertex is adjacent to every other vertex. Therefore, if we were to take all the vertices in a complete graph in any order, there will be a path through those vertices in that order. Joining either end of that path gives us a Hamilton cycle. It is well-know, there are $N!$ different ways of picking the vertices of the graph G in some order. However, not all of these orders represent a distinct Hamilton cycle, because for each of these linear orders

a. There are N different places from which each the cycle can start.

b. There are two possible directions that can be chosen.

Therefore, any of these $N!$ linear orders belongs to a set of $2N$ elements (linear orders) involving the same edges. Hence, there are

$$\frac{N!}{2N} = \frac{(N-1!)}{2},$$

distinct Hamilton cycles. □

When applying the proposed Divide-and-Conquer algorithm to a TSP instance Π of N vertices, two TSP instances $\Pi_{\text{left}}, \Pi_{\text{right}}$ of size $\frac{N}{2}$ are generated, which are sub-graphs induced from the original one. As Theorem 30 states, the original complexity is of order $\mathcal{O}\big(\frac{(N-1!)}{2}\big)$, while the complexity of the D&C pair is given by $\mathcal{O}\big((\lceil \frac{N}{2} \rceil - 1)!\big)$. This fact has two implications.

On the bright side, the time fraction T_f has to drop significantly, as the experiments confirm, given that whatever algorithm is used as an oracle, the difference in complexity from Π to $\Pi_{\text{left}}, \Pi_{\text{right}}$ is of the order half-factorial. On the down side, the behavior of the numerical fraction S_f is far from acceptable. This is most likely because the proposed algorithm does not reduce the complexity in half as in d-KP or BPP, but in a dramatic way. The results prove such strategy to be an oversimplification of the initial problem, a reduction which did not take into account the essential combinatorial structure of the problem.

5. CONCLUSIONS AND FINAL DISCUSSION

The present work yields several conclusions which are listed below

(i) For d-KP, the tables 2, 3, 4, as well as their corresponding figures 1, 2, 3 show that the proposed algorithm is recommendable, starting from certain number of items N_0. The latter, according to the required precision for the particular problem at hand. It is also clear that the tightness plays a very important role in defining the number N_0, starting from which the algorithm is useful. Although the dimension D also has an impact on N_0, it is clear that for different values of D, the expected values of the method's performance get close to each other for $N \geq 100$, which is reasonably low.

(ii) In the case of the BPP, it is clear that the proposed method has a poor interaction with the NFD algorithm, because the required computational time is always above 100% and a precision price is always paid. On the other hand, the FFD algorithm complements very well the Divide-and-Conquer algorithm, giving substantial reduction of computational time for a low number of items, namely $N \geq 50$. Finally, the BFD algorithm is a bad complement for the analyzed strategy, when compared with FFD algorithm. The latter, considering that BFD yields a marginal improvement of precision with respect to FFD, while using a significant higher fraction of computational time (although not as much as NFD). It is to be noted that the three algorithms yield significantly high precision values, even for low values of N and that ranking their usefulness is decided based on its computational time. All of the previous can be observed in the table 6 and the figure 4.

(iii) Analyzing the results for TSP, reported in table 8 and figure 6, it is immediate to conclude that the proposed method performs poorly for the MA case. It shows an erratic curve of computational time, although it attains a good precision level for $N \geq 50$ vertices. As for the other two cases, although they show reasonable computational time fraction for $N \geq 80$, the precision fraction never goes above 80%, which is very low value to consider the proposed algorithm as a valid option for solving TSP. Therefore, it follows that the method is not recommendable for TSP.

(iv) For the TSP, it is direct to see that the greedy function introduced, does not capture the structure/essence of the problem. An alternative way would be to define the subsets $V_{\text{left}}, V_{\text{right}}$ using a distance criterion for the metric TSP (MS). Namely, locate a pair of vertices $u, v \in V$ satisfying $\|u - v\| = \text{diameter}(V)$; then define V_{left} as the half of vertices closer to u (breaking ties with the index) and $V_{\text{right}} = V - V_{\text{left}}$. However, the following must be observed. First, this approach can only be used in MS. Second, when implemented its results are no better than those presented in [26, 36, 20] and [37].

(v) It must be also be reported that, without parallel implementation, the proposed algorithms not recommendable for d-KP and/or BPP, due to the involved computational time.

(vi) Although the proposed Divide-and-Conquer strategy has proved to be useful for d-KP, its extension to other generalized Knapsack Problems may not be direct. It must be stressed that the tightness of the problem (introduced in Definition 10) is strongly correlated with the performance of the method for d-KP. Analogously, other generalized Knapsack Problems have to be studied, as it was done here and observe the impact of those parameters containing structural information of the problem.

(vii) Deeper studies from the analytical point of view, analogous to those presented in [38], will be pursued in the future for d-KP and BPP. In order to theoretically establish the expected behavior as well as furnishing a quality certificate for the algorithms presented here, adequate probabilistic models and spaces will be introduced and analyzed.

Acknowledgements. The Author wishes to acknowledge Universidad Nacional de Colombia, Sede Medellín for supporting the production of this work, as well as granting access to Gauss Server, financed by "Proyecto Plan 150x150 Fomento de la cultura de evaluación continua a través del apoyo a planes de mejoramiento de los programas curriculares" (`gauss.medellin.unal.edu.co`), where the numerical experiments were executed. The Author also wishes to thank the anonymous reviewer whose insightful criticism and suggestions significantly improved the quality of this work.

Funding. This research was founded by Universidad Nacional de Colombia, Sede Medellín through the grant Hermes 54748.

REFERENCES

[1] F. A. Morales and J. A. Martínez, "Analysis of Divide-and-Conquer strategies for the 0-1 minimization problem," *Journal of Combinatorial Optimization*, vol. 40, no. 1, pp. 234 – 278, 2020.

[2] G. Diubin and A. Korbut, "Greedy algorithms for the minimization knapsack problem: Average behavior," *Journal of Computer and Systems Sciences International*, vol. 47, no. 1, pp. 14–24, 2008.

[3] A. Fréville and G. Plateau, "Heuristics and reduction methods for multiple constraints 0-1 linear programming problems," *European Journal of Operational Research*, vol. 24, pp. 206–215, 1986.

[4] R. Loulou and E. Michaelides, "New greedy-like heuristics for the multidimensional 0-1 knapsack problem," *Oper. Res.*, vol. 27, pp. 1101–1114, 1979.

[5] A. M. Frieze, M. Clarke *et al.*, "Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses," *European Journal of Operational Research*, vol. 15, no. 1, pp. 100–109, 1984.

[6] O. Oguz and M. Magazine, "A polynomial time approximation algorithm for the multidimensional 0/1 knapsack problem," *Univ. Waterloo Working Paper*, 1980.

[7] B. Gavish and H. Pirkul, "Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality," *Mathematical Programming*, vol. 31, pp. 78–105, 1985.

[8] A. Volgenant and J. Zoon, "An improved heuristic for multidimensional 0-1 knapsack problems," *Journal of the Operational Research Society*, vol. 41, pp. 963–970, 1990.

[9] P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *Journal of Heuristics*, vol. 4, pp. 63–86, 1998.

[10] X. Lai, J.-K. Hao, and D. Yue, "Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem," *European Journal of Operational Research*, vol. 274, no. 1, pp. 35–48, 2019. [Online]. Available: https://ideas.repec.org/a/eee/ejores/v274y2019i1p35-48.html

[11] W. F. de la Vega and G. S. Lueker, "Bin packing can be solved within $1+\epsilon$ in linear time," *Combinatorica*, vol. 1, pp. 349–355, 1981.

[12] N. Karmarkar and R. M. Karp, "An efficient approximation scheme for the one-dimensional bin-packing problem," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 1982, pp. 312–320.

[13] A. van Vliet, "An improved lower bound for on-line bin packing algorithms," *Information Processing Letters*, vol. 43, no. 5, pp. 277–284, 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/002001909290223I

[14] A. C.-C. Yao, “New algorithms for bin packing,” *J. ACM*, vol. 27, no. 2, p. 207–227, Apr. 1980. [Online]. Available: https://doi.org/10.1145/322186.322187

[15] C. C. Lee and D. T. Lee, “A simple on-line bin-packing algorithm,” *J. ACM*, vol. 32, no. 3, p. 562–572, Jul. 1985. [Online]. Available: https://doi.org/10.1145/3828.3833

[16] E. G. Coffman, M. R. Garey, and D. S. Johnson, *Approximation Algorithms for Bin-Packing — An Updated Survey.* Vienna: Springer Vienna, 1984, pp. 49–106. [Online]. Available: https://doi.org/10.1007/978-3-7091-4338-4_3

[17] T. Fischer and P. Merz, “Reducing the size of traveling salesman problem instances by fixing edges,” in *Evolutionary Computation in Combinatorial Optimization*, C. Cotta and J. van Hemert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 72–83.

[18] C. Walshaw, “A multilevel approach to the Travelling Salesman Problem,” *Oper. Res.*, vol. 50, pp. 862–877, 2002.

[19] ——, “Multilevel refinement for combinatorial optimisation problems,” *Annals of Operations Research*, vol. 131, pp. 325–372, 2004.

[20] K. Ishikawa, I. Suzuki, M. Yamamoto, and M. Furukawa, “Solving for large-scale traveling salesman problem with divide-and-conquer strategy,” *SCIS & ISIS*, vol. 2010, pp. 1168–1173, 2010.

[21] B.-R. C. V. C. W. Applegate, David, “On the solution of traveling salesman problems.” *Documenta Mathematica*, pp. 645–656, 1998. [Online]. Available: http://eudml.org/doc/233207

[22] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, “Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems.” *Math. Program.*, vol. 97, no. 1-2, pp. 91–153, 2003. [Online]. Available: http://dblp.uni-trier.de/db/journals/mp/mp97.html#ApplegateBCC03

[23] G. Dantzig, R. Fulkerson, and S. Johnson, “Solution of a large-scale traveling-salesman problem,” *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954. [Online]. Available: http://www.jstor.org/stable/166695

[24] D. S. Johnson, “Local optimization and the traveling salesman problem,” in *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, ser. ICALP ’90. Berlin, Heidelberg: Springer-Verlag, 1990, p. 446–461.

[25] S. Lin and B. W. Kernighan, “An effective heuristic algorithm for the traveling-salesman problem,” *Oper. Res.*, vol. 21, pp. 498–516, 1973.

[26] D. Applegate, W. Cook, and A. Rohe, “Chained Lin-Kernighan for large Traveling Salesman Problems,” *INFORMS Journal on Computing*, vol. 15, no. 1, pp. 82–92, 2003. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/ijoc.15.1.82.15157

[27] W. Cook and P. Seymour, “Tour merging via branch-decomposition,” *INFORMS Journal on Computing*, vol. 15, no. 3, pp. 233–248, 2003. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/ijoc.15.3.233.16078

[28] P. Billinsgley, *Probability and Measure*, ser. Wiley Series in Probability and Mathematical Statistics. New York, NY: John Wiley & Sons, Inc., 1995.

[29] S. K. Thompson, *Sampling*, ser. Wiley Series in Probability and Statistics. New York: John Wiley & Sons, Inc., 2012.

[30] R. Mansini and M. G. Speranza, “Coral: An exact algorithm for the multidimensional knapsack problem,” *INFORMS J. on Computing*, vol. 24, no. 3, p. 399–415, Jul. 2012. [Online]. Available: https://doi.org/10.1287/ijoc.1110.0460

[31] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations.* USA: John Wiley & Sons, Inc., 1990.

[32] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, ser. Discrete Mathematics and its Applications. Ney York: Springer, Berlin, Heidelberg, 2004.

[33] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 5th ed. Springer Publishing Company, Incorporated, 2012.

[34] G. Dósa, R. Li, X. Han, and Z. Tuza, “Tight absolute bound for first fit decreasing bin-packing: Ffd(l)11/9 opt(l)+6/9,” *Theoretical Computer Science*, vol. 510, pp. 13–61, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397513006774

[35] D. Bertsimas and J. N. Tritsiklis, *Introduction to Linear Optimization.* Belmont, MA: Athena Scientific and Dynamic Ideas, LLC, 1997.

[36] B. Fritzke and P. Wilke, “Flexmap-a neural network for the traveling salesman problem with linear time and space complexity,” in *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, 1991, pp. 929–934 vol.2.

[37] D. S. Johnson and L. A. McGeoch, *Experimental Analysis of Heuristics for the STSP*. Boston, MA: Springer US, 2007, pp. 369–443. [Online]. Available: https://doi.org/10.1007/0-306-48213-4_9

[38] F. A. Morales and J. A. Martínez, “Expected performance and worst case scenario analysis of the Divide-and-Conquer method for the 0-1 knapsack problem,” *arXiv*, 2020. [Online]. Available: https://arxiv.org/abs/2008.04124