

A Symmetry-breaking method for The Unit Stop Number Problem

Estefânia de Sá Moura

Universidade Federal de Ouro Preto
Rua 36, 115 - Loanda, João Monlevade - MG
estefania.moura@aluno.ufop.edu.br

Williane Cristina Ribeiro

Universidade Federal de Ouro Preto
Rua 36, 115 - Loanda, João Monlevade - MG
willianecr@gmail.com

Rafael Colares

Université Clermont Auvergne
Rue de la Chebarde, 63178 Aubière, França
rafael.colares@gmail.com

Hervé Kerivin

Université Clermont Auvergne
Rue de la Chebarde, 63178 Aubière, França
herve.kerivin@uca.fr

ABSTRACT

The resolution of Integer Linear Programming formulations generally requires exponential computational time. One of the factors that can contribute to time explosion is the presence of symmetry. Fortunately, symmetry can usually be spotted and treated efficiently. When addressing combinatorial optimization problems with identical allocating resources (*i.e.*, identical machines on Lot Sizing problems), one simple permutation between them can generate symmetric solutions. Such issue appears within the Unit Stop Number Problem, a dial-a-ride transport problem appearing from the deployment of autonomous electric vehicles. In such problem, identical vehicles travels along a predefined circuit where clients request for rides between stations. The goal is to minimize the number of performed pick-up and delivering operations. In this paper we show how symmetry is spotted in such problem and propose to efficiently treat it by adapting and applying the well known Orbitopal Fixing method. Computational results convincingly show the efficiency of our approach.

KEYWORDS. Symmetry-breaking methods, Orbitopal Fixing, Unit Stop Number Problem

1. Introduction

In the face of the advances of industry 4.0, which is frequently related to the Internet of Things (IoT), Cyber Physical System (CPS), information and communications technology (ICT) [Lu, 2017], there is a constant need for machine automation, which must handle increasingly complex activities with the least human assistance. In order to follow up these progresses, the European companies [Ligier, 2020] and [Easymile, 2020] designed the VIPA (a French acronym for Autonomous Individual Passenger Vehicle). The VIPA is a no-driver electric vehicle whose main purpose is to serve client trips on closed or semi-closed areas such as industrial, medical and commercial ones.

Such vehicles can operate in many different modes. In this study we focus on the so-called 'tram mode' (see [Bsaybes, 2017] for a study on other modes). In such mode of operation, a fleet of identical capacitated vehicles travels along a circuit (always in the same direction) with predefined stations serving clients that wish for a ride between two stations. Clients may request for a ride either through an smartphone application or via a physical terminal.

It should be noted however that due to infrastructure restrictions stations are not placed inside the circuit but rather attached to it (Figure 1 provides an illustration of such circuit). This means that each time a vehicle picks up (or delivers) a client, an extra distance must be traveled. Therefore reducing the overall number of stops induces cost reductions and improves the service quality. The *Stop Number Problem* (SNP) consists of assigning each client demand to a vehicle while respecting its capacity such that the total number of stops is minimized. When every client requests for exactly one seat and vehicles are constrained to serve all client demands within a single tour (*i.e.*, no vehicle may perform a complete circuit lap before serving a client), SNP is called *Unit Stop Number Problem* (USNP). This paper focuses on this latter variation but the methods proposed can be easily extended to the more general case.

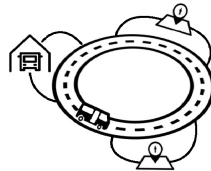


Figure 1: Illustration of the VIPA circuit

SNP and USNP have been introduced by [Pimenta et al., 2017] where SNP was proved to be NP-Hard and USNP was conjectured to remain NP-Hard. Such conjecture was later positively answered by [Baïou et al., 2018]. Both these studies have also proposed exact approaches to solve USNP. While [Pimenta et al., 2017] have proposed a column generation approach, [Baïou et al., 2018] have suggested a Branch-and-Cut framework where symmetry-breaking constraints have also been added to the original Integer Linear Program (ILP) formulation.

Indeed, many assignment problems have to deal with the symmetry issue. According to [Margot, 2009], "*an ILP is symmetric if its variables can be permuted without changing the structure of the problem*". When it comes to USNP, since vehicles are taken to be identical, a simple client-vehicle permutation can generate solutions with same costs, that is, isomorphic solutions. This idea is further discussed in Section 2.

[Margot, 2009] also shows that the presence of symmetry can deeply disturb the resolution performance of Branch-and-Bound based methods (for further details on Branch-and-Bound and Branch-and-Cut based methods, the reader is referred to [Wolsey, 1998]). The intention of this work

is hence to approach and treat the symmetry issue behind USNP. For this we adapt and implement the Orbitopal Fixing method (see [Kaibel et al., 2007]) to fit USNP. A computational study is then conducted, showing that the proposed method is indeed capable of significantly enhancing the ILP's solver performance.

The paper is organized as follows. In Section 2, USNP is formally described and symmetry in integer programs is discussed. Section 3 shows how we adapt the Orbitopal Fixing method proposed by [Kaibel et al., 2007] so it can be applied to the USNP. Section 4 shows the method implementation and computational results obtained when using it. Finally, we conclude the paper in section 5

2. Previous and related works

2.1. The Unit Stop Number Problem

Since vehicles are allowed to perform only one tour in USNP, the circuit can be easily represented as a linear ordered sequence of stations (see [Pimenta et al., 2017]) and hence demands may be represented by intervals between its origin and destination stations in this line. Figure 2 provides a snapshot of such representation.

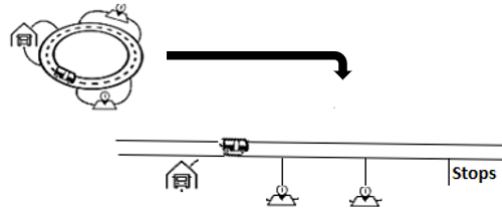


Figure 2: Linear representation of the VIPA circuit

USNP can hence be formally defined as follows. Let $V = \{1, \dots, n\}$ denote the set of stations ordered as they appear in the circuit and let $E = \{e_1, \dots, e_m\}$ be the set of client demands where each demand $e \in E$ is defined by an origin station $o_e \in V$ and a destination station $d_e \in V$, that is, $e = (o_e, d_e)$. Lastly, let $K = \{1, \dots, p\}$ be the set of available identical vehicles, each with capacity $C \in \mathbb{N}$.

In order to check which clients can be simultaneously together on a vehicle, let $\Delta_E(v) = \{e \in E : o_e \leq v \text{ and } d_e \geq v + 1\}$, that is, the set of demands that cross or starts at station v . We say that demand $e \in E$ intersects station $v \in V$ if $e \in \Delta_E(v)$. A feasible solution to USNP consists hence of a partition of E into p subsets $\{E_1, \dots, E_p\}$ such that $|\Delta_{E_i}(v)| \leq C$ for any $i \in K$ and any $v \in V$. Since a vehicle stops at every station spanned by its assigned demands, we can say that a vehicle $i \in K$ stops at every station in $V(E_i)$ and hence the cost of a solution is $\sum_{i=1}^{|K|} |V(E_i)|$. The goal of USNP is to find a feasible solution of minimum cost.

In order to solve USNP, [Pimenta et al., 2017] proposed an ILP formulation that will serve us as a basis and hence is recalled below.

$$\min \sum_{v \in V} \sum_{i \in K} y_v^i \quad (1)$$

$$\text{S.t.:} \quad \sum_{i \in K} x_e^i = 1, \quad \forall e \in E, \quad (2)$$

$$\sum_{j \in \Delta_E(v)} x_e^i \leq C, \quad \forall v \in V, i \in K, \quad (3)$$

$$x_e^i \leq y_v^i, \quad \forall i \in K, e \in E, v \in \{o_e, d_e\}, \quad (4)$$

$$x_e^i \in \{0, 1\}, \quad \forall e \in E, i \in K, \quad (5)$$

$$y_v^i \in \{0, 1\}, \quad \forall v \in V, i \in K. \quad (6)$$

Formulation (1)-(6) is based on two sets of decision variables. Variable x_e^i represents the client-vehicle allocation (if demand e is assigned to vehicle i then $x_e^i = 1$, otherwise $x_e^i = 0$). Variable y_v^i expresses whether vehicle i stops or not at station v . In other words, $y_v^i = 1$ if it stops and $y_v^i = 0$, otherwise.

The objective function (1) minimizes the global number of stops. Constraints (2) impose that every demand must be satisfied by a single vehicle. Constraints (3) guarantee that the vehicle's capacity is respected all along the circuit. Constraints (4) ensure that each vehicle stops at both origin and destination stations of the demands assigned to it. Finally, constraints (5) and (6) impose variable integrality.

Next we show how symmetry can be spotted in combinatorial problems, why it can be harmful and what can be done to handle it.

2.2. Symmetry in optimization problems

As mentioned previously, symmetry arises in an ILP when many different isomorphic solutions can be generated from the permutation of its variables. [Margot, 2009] indicates that symmetric ILPs often appear on classic combinatorial optimization problems such as graph coloring, scheduling of jobs on parallel identical machines, covering design or code construction. It is known that when solving symmetric ILPs through Branch-and-Bound or Branch-and-Cut methods, one is forced to solve an important number of isomorphic subproblems in the enumeration tree, which represents a wasteful duplication of efforts. Reducing an ILP symmetry can hence significantly improve its resolution performance.

Before diving into the symmetry behind USNP, we provide some didactic examples of how symmetry may appear in optimization problems. [Denton, 2010] studied the problem of allocating surgeries to operating rooms. Such problem consists of deciding which operation rooms should be opened as well as assigning surgical blocks to it by minimizing all costs (considering that there is a cost for opening rooms and also a cost for delayed surgeries). Since all rooms are identical, there is a strong symmetry on assigning decisions.

With the idea of reducing the problem's symmetry, [Denton, 2010] applied a lexicographic order to the variables and proposed two families of (polynomial-number) symmetry-breaking constraints. It should be noted that such inequalities are not valid inequalities since they cut off feasible solutions. However, their inclusion preserves at least one optimal solution as feasible. In short, this lexicographic order imposes that (i) the i -th surgery must be assigned to one of the first i operating rooms, and (ii) a surgery cannot be assigned to an empty operating room j if there exists another empty operating room j' such that $j' \prec j$.

Nonetheless, even if the inclusion of such symmetry-breaking constraints allowed to speed up the ILP formulation, they are unable to entirely remove the symmetry. Indeed, for doing so, an exponential number of constraints would be required.

[Scalton, 2012] studied the symmetry behind a single-machine lot sizing problem. The aim of this problem is to sequence production operations into a limited capacity machine in a way that the overall working time (*i.e.*, makespan) is minimized.

Figure 3 was adapted from [Scalton, 2012] to provide an illustration of two different sequences of production that generate isomorphic solutions. Each block represents a production operation where its height symbolizes the time required and its length the resource consumption. To suppress symmetry, he also incorporates constraints that follows a lexicographic order, forcing each task to be assigned to a lot with an index greater than or equal to the index of the task itself. Thus, given a set of tasks $T = \{1, \dots, n\}$ and a set of lots $L = \{1, \dots, m\}$, the task n must only be assigned to the lot m , the task $n - 1$ must be assigned to the lots $n - 1$ and n , and so on.

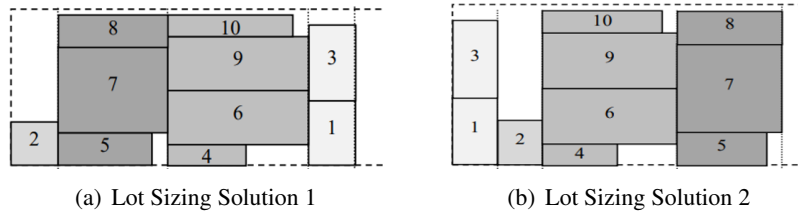


Figure 3: Comparison between two Lot Sizing Solutions

With respect to USNP, since all vehicles are identical, vehicles of a given solution can be easily rearranged to generate isomorphic solutions. Figure 4 illustrates two USNP isomorphic solutions. It follows that for each solution of an USNP instance with p available vehicles, there may exist $p!$ symmetric solutions. In the view of this fact, [Baïou et al., 2018] has adapted and introduced the symmetry-breaking constraints proposed by [Denton, 2010] to avoid redundantly exploring all isomorphic solutions. However, the proposed constraints were also unable of entirely removing symmetry from the formulation (1)-(6).

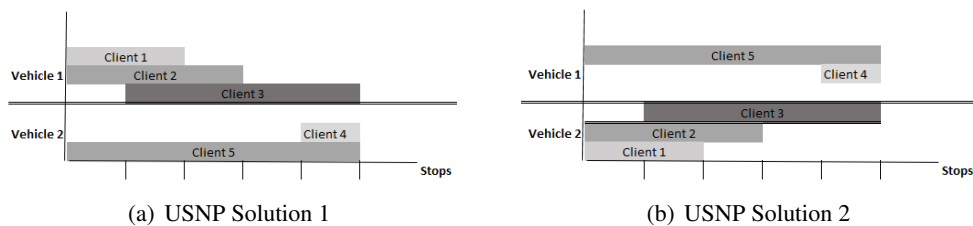


Figure 4: Comparison between two USNP solutions

Next we propose a method for completely removing symmetry (with respect to vehicle permutations) from formulation (1)-(6). Such method is described in the following section.

3. Orbitopal Fixing Method Formulation

[Kaibel et al., 2007] proposed a new method called Orbitopal fixing that allows to completely remove symmetry from assignment problems. This method also uses the idea of imposing a lexicographic ordering on the variables of a problem. With respect to USNP, such lexicographic order amounts to say that

- (i) the i -th demand must be assigned to one of the first i vehicles, and
- (ii) a demand i cannot be assigned to an empty vehicle j if there exists some empty vehicle j' such that $j' < j$.

Imposing such ordering eliminates some feasible solutions. To illustrate it, let x_{lexico} and x_{elim} be two matrices of variables x , where

$$x_{lexico} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad x_{elim} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

and each the line is associated with a demand $e \in E$ and each column with a vehicle $i \in K$. While the solution provided by matrix x_{lexico} respects the lexicographic order, x_{elim} does not and hence can be eliminated. Notice however that for every eliminated solution, there exists an isomorphic solution that respects the lexicographic order imposed. The isomorphic solution x' derived from x_{elim} respecting such ordering would be

$$x' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

However, instead of deriving symmetry-breaking constraints and adding them to the ILP formulation, Orbitopal Fixing applies such rules progressively, by acting on the branching phase of the Branch-and-Bound or Branch-and-Cut procedure. More precisely, on each branching phase the method uses the subset of variables that have already been fixed to integer values (i. e. the partial solution) to force not yet fixed variables to definitely assume integer values, imposing the lexicographic order defined by (i) and (ii). The method proposed by [Kaibel et al., 2007] was adapted so it can be used for the USNP and it is described as follows.

The first step of the algorithm receives a partial solution as input. Such partial solution can be represented by the pair (F_0, F_1) , where F_0 stands for the set of variables that have been fixed to zero and F_1 for the set of variables fixed to one. The algorithm then fixes some variables according to the lexicographic order adopted and provides a new pair (F_0^*, F_1^*) as output, where $F_0 \subseteq F_0^*$ and $F_1 \subseteq F_1^*$.

For this, let $a \in \{1..p\}^m$ be a vector providing for each client, an upper limit on the vehicle index that can be assigned to it. For instance, if $a_j = 5$, then client e_j can only be assigned to one of the first 5 vehicles, and hence $x_{e_j}^i = 0$ for every $i > 5$. It should be noted that from condition (i), $a_1 = 1$ and for every $2 \leq j \leq m$, if $a_{j-1} < p$ then $a_j = a_{j-1} + 1$, otherwise $a_j = p$. Furthermore, it is possible that $x_{e_j}^{a_{j-1}+1}$ has already been set to zero by some branching decision. If this is the case, condition (ii) allows to set $a_j = a_{j-1}$. The first step of the algorithm is formally described in Algorithm 1.

Algorithm 1: Orbitopal Fixing (Setting zeros) [Kaibel et al., 2007] adapted

Input: The sets of previously fixed variables (F_1, F_0)

Output: (F_1^*, F_0^*) with $F_0^* \subseteq F_0$

begin

```

 $a_1 \leftarrow 1$ 
 $F_0^* \leftarrow F_0$ 
for  $j = 2, \dots, m$  do
    if  $a_{j-1} = p$  or  $x_{e_j}^{a_{j-1}+1} \in F_0$  then
         $a_j = a_{j-1}$ 
    else
         $a_j = a_{j-1} + 1$ 
     $F_0^* \leftarrow F_0^* \cup \{x_{e_j}^i : a_j < i \leq p\}$ 

```

The second step of the Orbitopal Fixing searches for contradictions between the partial solution (F_1, F_0) and the output from algorithm 1 if an additional variable is assumed to be zero. For a demonstration, consider the matrix of variables x representing a partial solution, where

$$x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ ? & ? & 0 & 0 \\ 0 & 0 & 1 & 0 \\ ? & ? & ? & ? \end{bmatrix}$$

and variables in F_1 and F_0 are marked with 1 and 0, respectively. Non-fixed variables are marked with a question mark '?'.

If one assumes that $x_2^2 = 0$, algorithm 1 will force variable x_3^3 to be fixed to zero. Since such variable is already fixed to one, a contradiction is detected and hence x_2^2 must be fixed to one. The second step of Orbitopal Fixing method is formally described as follows.

Algorithm 2: Orbitopal Fixing (Setting ones) [Kaibel et al., 2007] adapted

Input: (F_1, F_0)

Output: (F_1', F_0') with $F_1 \subseteq F_1'$ and $F_0 \subseteq F_0'$

begin

```

 $F_1' \leftarrow F_1$ 
 $F_0' \leftarrow F_0$ 
for  $j = 2, \dots, m$  such that  $x_{e_j}^{a_j} \notin F_1' \cup F_0'$  do
     $(F_1'^*, F_0'^*) \leftarrow$  Algorithm 1 with  $(F_1', F_0' \cup \{x_{e_j}^{a_j}\})$  as input
    if  $F_1'^* \cap F_0'^* \neq \emptyset$  then
         $F_1' = F_1' \cup \{x_{e_j}^{a_j}\}$ 
         $F_0' = F_0' \cup \{x_{e_j}^i : i \in K, i \neq a_j\}$ 

```

In the next section we discuss the implementation of this method as well as the computational results obtained with it.

4. Implementation and Computational Results

Our developed framework was written in C++ language and uses the commercial solver ILOG CPLEX 12.8 from [IBM, 2020] as the base ILP solver. The proposed Orbitopal Fixing

algorithm was then implemented through CPLEX's user-callback functions that allows users to intervene in its branching phase. All computations were performed within a time limit of one hour on a single-thread machine equipped with Intel Xeon E7 2.50 GHz and 32 Gb RAM running linux.

In order to evaluate the performances of the proposed method, our experiments are performed over a set of 18 USNP instances representing different scenarios based on the vehicle's capacity C , the number of client demands m , and the density ρ (defined as the number of demands over the number of stations). For each combination of parameters $C \in \{2, 5, 8\}$, $m \in \{30, 40, 50\}$ and $\rho \in \{1.5, 4.5\}$, an instance was randomly generated and named according to the code $I.C.m.\rho$.

Our goal is to compare the performances obtained with default CPLEX and the activation of Orbitopal Fixing algorithm within CPLEX's branching phases. Due to the use of user-callback functions in CPLEX in our implementation, dummy callbacks were used when CPLEX was executed with default settings. This is made necessary in order to get a fair comparison, since the only presence of a user-callback function changes the behavior of CPLEX.

The obtained computational results are displayed in Table 1, where results achieved with default CPLEX are presented under column `def` and the ones obtained through the activation of Orbitopal Fixing appears under column `OF`. For each tested instance, the lower and upper bounds achieved by the end of time limit are displayed under sections `LB` and `UB`, respectively. The time spent in the optimization process is presented under section `Time`, and the remaining relative gap appears under section `Gap`. Finally section `Nodes` shows the number of nodes explored within the enumeration tree.

Table 1: Results for evaluating Orbitopal Fixing

Instance	LB		UB		Time		Gap		Nodes	
	def	OF	def	OF	def	OF	def	OF	def	OF
I.2.30.1.5	20,8	30,5	35	35	3600	3600	40,5	12,8	523200	93490
I.2.30.4.5	8,9	15,5	24	24	3600	3600	62,8	35,3	480100	81321
I.2.40.1.5	27,0	36,7	51	50	3600	3600	47,1	26,6	134099	21730
I.2.40.4.5	10,0	17,8	35	35	3600	3600	71,4	49,2	125600	21446
I.2.50.1.5	34,7	46,8	63	64	3600	3600	44,9	26,9	52860	8222
I.2.50.4.5	12,0	22,2	48	49	3600	3600	74,9	54,6	43855	7644
I.5.30.1.5	19,7	25,0	25	25	3600	1007	21,1	0,0	576800	46686
I.5.30.4.5	9,1	14,0	14	14	3600	2622	35,3	0,0	724778	115738
I.5.40.1.5	28,0	32,9	37	37	3600	3600	24,3	11,2	203720	25877
I.5.40.4.5	10,0	15,0	20	20	3600	3600	50,0	24,8	217100	25000
I.5.50.1.5	33,0	39,0	46	48	3600	3600	28,3	18,8	38400	8772
I.5.50.4.5	11,9	18,4	30	30	3600	3600	60,4	38,8	45600	8451
I.8.30.1.5	23,9	24,0	24	24	3600	9	0,5	0,0	879578	269
I.8.30.4.5	9,5	11,0	11	11	3600	70	13,6	0,0	829680	3167
I.8.40.1.5	27,0	32,0	32	32	3600	87	15,6	0,0	211300	1081
I.8.40.4.5	10,1	14,0	14	14	3600	266	27,8	0,0	250500	3299
I.8.50.1.5	33,0	39,4	41	41	3600	3600	19,5	3,9	68200	11085
I.8.50.4.5	12,4	17,3	22	23	3600	3600	43,4	24,9	47400	9299

Table 1 clearly shows that the method was capable of considerably improving the framework's performance. While default CPLEX could not solve any of the tested instances to optimality within the one hour limit time, the activation of Orbitopal Fixing allowed six instances to be solved. It is worth remarking that four of these instances required less than five minutes to be solved. In this regard, instance I.8.30.1.5 deserves to be highlighted since the activation of the symmetry-breaking method allowed the instance to be solved in less than ten seconds.

The benefits of breaking the problem's symmetry can also be spotted by comparing the number of nodes in the enumeration tree that are required to be explored in order to solve the problem. Indeed, for those instances on which the optimality has been proved, the average number

of nodes explored using the Orbitopal Fixing method was 29,000 nodes, while default CPLEX explored 579,000 nodes and still failed to achieve a proof of optimality. For instance I.8_30_1.5, default CPLEX explored more than 800,000 nodes without obtaining such optimality proof, while the use of Orbitopal Fixing allowed such proof within less than 300 nodes, which represents a reduction of 99.97%.

Since the proposed method allowed to spare the exploration of redundant nodes, more meaningful nodes could be explored, allowing the bounds to converge faster. In this sense, even if some instances could not be solved within the time limit, another striking performance difference appears when comparing the bounds and the gap obtained after the one hour time limit. For such instances, we noticed an average increase of the lower bound values of approximately 46% when Orbitopal Fixing was activated. For dense instances (*i.e.*, instances where $\rho = 4.5$) such improvement reached up to 64% in average. The increase in the convergence speed of lower bounds directly implies an improvement in the final gap values, given that the upper bounds do not vary as much. On the one hand, the average gap value goes from 26.9% to 11.1% with the activation of the proposed method, for sparse instances. For dense instances on the other hand, the average gap value reduces from 48.9% to 25.3%.

It is well known that as one changes the order in which variables are chosen to be branched within a Branch-and-Cut procedure, the solver's performance may vary. Recent studies are even focusing on applying machine learning methods in order to help the decision of which variable to branch (see [Khalil et al., 2016]). For this reason, we propose two different strategies and try to compare them with the CPLEX default's choice.

The first strategy is called *Linear* and consists on looking at the variables in a linear way as they appear in the circuit, that is, the variable chosen for branching is the variable with fractional value that has the smallest origin station. In case of a tie between variables, the variable with smallest destination station is chosen among these. Table 1 shows the computational results obtained using this strategy when Orbitopal Fixing is activated or not.

Table 2: Results when Linear branching strategy is applied

Instance	LB		UB		Time		Gap		Nodes	
	def	OF	def	OF	def	OF	def	OF	def	OF
I.2_30_1.5	21,8	28,9	35	35	3600	3600	37,8	17,3	469781	100335
I.2_30_4.5	9,5	14,7	24	24	3600	3600	60,4	38,6	265700	90747
I.2_40_1.5	28,7	35,6	51	51	3600	3600	43,7	30,1	149163	22229
I.2_40_4.5	11,6	17,0	35	35	3600	3600	66,8	51,5	82458	22644
I.2_50_1.5	34,9	46,7	63	64	3600	3600	44,5	27,0	37862	8298
I.2_50_4.5	12,9	22,6	48	50	3600	3600	73,1	54,7	52600	7726
I.5_30_1.5	20,7	24,6	25	25	3600	3600	17,2	1,6	416144	141902
I.5_30_4.5	9,8	13,5	14	14	3600	3600	30,1	3,8	376628	150878
I.5_40_1.5	29,1	32,4	37	37	3600	3600	21,5	12,3	85900	23475
I.5_40_4.5	10,3	14,6	20	20	3600	3600	48,4	26,8	85814	25848
I.5_50_1.5	34,0	39,4	48	47	3600	3600	29,2	16,1	43600	8644
I.5_50_4.5	13,0	18,7	30	32	3600	3600	56,7	41,5	42536	7408
I.8_30_1.5	23,2	24,0	24	24	3600	8	3,3	0,0	872396	229
I.8_30_4.5	8,8	11,0	11	11	3600	289	20,0	0,0	514903	11673
I.8_40_1.5	27,9	32,0	32	32	3600	1400	12,9	0,0	102967	16950
I.8_40_4.5	10,5	14,0	14	14	3600	631	24,8	0,0	106045	7743
I.8_50_1.5	33,0	38,3	41	41	3600	3600	19,5	6,5	36100	11832
I.8_50_4.5	13,0	16,7	23	23	3600	3600	43,5	27,5	39775	9640

The second strategy is called *Sharing* and it takes variables according to the number of shared stations with each other. In other words, a fractional variable x_j^k is chosen to be branched

before a fractional variable x_i^l if

$$\left| \left\{ e \in E : o_e \in \{o_j, d_j\} \text{ or } d_e \in \{o_j, d_j\} \right\} \right| > \left| \left\{ e \in E : o_e \in \{o_i, d_i\} \text{ or } d_e \in \{o_i, d_i\} \right\} \right|.$$

In case of a tie between variables, the variable with smallest vehicle index is chosen. The results obtained with this strategy are shown in table 3.

Table 3: Results when Sharing branching strategy is applied

Instance	LB		UB		Time		Gap		Nodes	
	def	OF	def	OF	def	OF	def	OF	def	OF
I.2.30.1.5	21,8	30,0	35	35	3600	3600	37,6	14,2	435427	104088
I.2.30.4.5	9,7	15,4	24	24	3600	3600	59,7	35,9	379300	86407
I.2.40.1.5	28,7	36,6	50	50	3600	3600	42,6	26,8	103500	22776
I.2.40.4.5	13,1	17,2	35	35	3600	3600	78,2	50,8	71069	22208
I.2.50.1.5	34,9	47,6	63	63	3600	3600	44,5	24,5	37449	8322
I.2.50.4.5	13,0	22,6	48	50	3600	3600	72,9	54,8	84135	8691
I.5.30.1.5	20,7	25,0	25	25	3600	2425	17,1	0,0	301600	111437
I.5.30.4.5	9,1	12,5	14	14	3600	3600	34,8	10,5	300500	101432
I.5.40.1.5	29,1	33,1	37	37	3600	3600	21,3	10,6	73800	27870
I.5.40.4.5	11,1	14,4	19	20	3600	3600	41,4	28,0	123000	23267
I.5.50.1.5	34,0	39,6	47	48	3600	3600	27,7	17,6	41800	9120
I.5.50.4.5	12,8	17,8	31	31	3600	3600	58,7	42,5	44501	7737
I.8.30.1.5	24,0	24,0	24	24	1957	8	0,0	0,0	630239	265
I.8.30.4.5	8,3	11,0	11	11	3600	784	24,7	0,0	351415	29849
I.8.40.1.5	27,9	32,0	32	32	3600	594	12,9	0,0	82365	8025
I.8.40.4.5	10,5	14,0	14	14	3600	2853	24,8	0,0	114200	35229
I.8.50.1.5	34,0	39,0	41	41	3600	3600	17,1	5,0	39979	12030
I.8.50.4.5	13,0	16,7	23	23	3600	3600	43,5	27,5	46889	7943

Independently of the branching strategy chosen, the Orbitopal Fixing method always allowed better performance results than the ones obtained with default CPLEX. In terms of branching strategies, we notice that there were no consistent performance improvements when using either *Sharing* or *Linear* strategies in comparison with CPLEX default's choice of branching. Nonetheless, it seems that *Linear* strategy performs better when the instance is dense and *Sharing* strategy excels in sparse instances. While instances I.2.50.1.5, I.5.40.1.5 and I.5.50.1.5 obtained the best lower bound results with *Sharing* strategy, instances I.2.50.4.5 and I.5.50.4.5 obtained the best lower bounds with *Linear* strategy. On average, while CPLEX default's choice of branching (with activation of Orbitopal Branching) remains the best strategy for solving dense demands, sparse instances had a slightly increase in the final lower bound value when using *Sharing* strategy in comparison to the other tested strategies which might be a good hint for future studies on branching strategies for the USNP.

5. Conclusion and Future works

Despite appearing recurrently on optimization problems, symmetry is not as approached in the literature. When symmetry is spotted within a problem, it is essential to treat it effectively. In view of its effect on the solution enumeration trees, methods based on Branch-and-Bound (and Branch-and-Cut) may not present the desired performances. Thus, finding ways to improve branching decisions seems to be ideal for these types of problems.

The work of [Baïou et al., 2018] was the key point of our study, as it approaches symmetry in USNP and raises alternatives for suppressing it through the addition of symmetry-breaking constraints. Even if such alternatives have showed to be effective, they were not able of entirely removing the symmetry of the problem. In order to improve the resolution of the USNP as well

as to completely remove its symmetry, we have implemented the Orbitopal Fixing method [Kaibel et al., 2007], which was adapted so that it could fit in the context of USNP.

Indeed the Orbitopal Fixing method allowed to significantly reduce the size of the enumeration tree needed for proving optimality within a Branch-and-Cut (or Branch-and-Bound) framework. Isomorphic solutions were eliminated and the resolution was remarkably accelerated. Since the core of our study lied in how to perform effective branching, we suggested two different strategies for choosing the variable to be branched on. Both strategies did not show to consistently outperform CPLEX standard choice of branching, even if for some specific cases a gain of performance was detected. Perhaps other branching strategies can be used as topics for other studies.

All in all, based on the results obtained we can say that Orbitopal Fixing was successfully applied to USNP. We believe that if some of the instances remain hard to solve to optimality, it is due to the weakness of the proposed ILP formulation. Future studies may focus on developing reinforced extended formulations or identifying strong valid inequalities to strengthen the given formulation. Finally, this paper should serve as an impetus for other works on similar themes, encouraging the development of techniques and innovations for symmetric problems. We believe that the analysis and comparison of methods linked to branching strategies is an area that should be more explored.

6. Acknowledgements

The authors would like to thank the laboratory of excellence IMobS3, the University of Clermont Auvergne, the Federal University of Ouro Preto and the National Council for the Improvement of Higher Education (CAPES) for the financial and material support.

References

- Baïou, M., Colares, R., e Kerivin, H. (2018). The stop number minimization problem: Complexity and polyhedral analysis. *International Symposium on Combinatorial Optimization.*, p. 3;7–8.
- Bsaybes, S. (2017). *Modèles et algorithmes de gestion de flottes de véhicules vipa*. PhD thesis, PhD thesis, Université Clermont Auvergne.
- Denton, B. T. (2010). Optimal allocation of surgery blocks to operating rooms under uncertainty. *Operations research*, 58:802–816.
- Easymile (2020). Autonomous future, today - easymile. <http://www.easymile.com>. Accessed: 2020-04-25.
- IBM (2020). Knowledge center ibm. <http://www.ibm.com>. Accessed: 2020-05-25.
- Kaibel, V., Peinhardt, M., e Pfetsch, M. E. (2007). Orbitopal fixing. *IPCO 2007: The Twelfth Conference on Integer Programming and Combinatorial Optimization*, 4513:74–88.
- Khalil, E. B., Le Bodic, P., Song, L., Nemhauser, G., e Dilkina, B. (2016). Learning to branch in mixed integer programming. In *Thirtieth AAAI Conference on Artificial Intelligence*.
- Ligier (2020). Voitures sans permis ligier & mirocar. <https://www.ligier.fr/>. Accessed: 2020-04-25.
- Lu, Y. (2017). Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, 6:1–10.

- Margot, F. (2009). Symmetry in integer linear programming. *50 Years of Integer Programming, Springer*, p. 1–4;35.
- Pimenta, V., Quilliot, A., Toussaint, H., e Vigo, D. (2017). Models and algorithms for reliability-oriented dial-a-ride with autonomous electric vehicles. *European Journal of Operational Research*, 257:601–613.
- Scalcon, C. G. (2012). Um modelo de otimização para o problema de dimensionamento e programação de lotes de produção em máquina única. Postgraduate dissertation 1, Universidade Federal de Santa Maria.
- Wolsey, L. A. (1998). *Integer Programming*, volume 1. John Wiley & Sons.