IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 00.000/ACCESS.2021.DOI

Efficient Computation and Estimation of the Shapley Value for Traveling Salesman Games

CHAYA LEVINGER[1], NOAM HAZON[2], AND AMOS AZARIA [3]
[1]chaya.levinger@msmail.ariel.ac.il
[2]noamh@ariel.ac.il
[3]amos.azaria@ariel.ac.il
Department of Computer Science, Ariel University, Ariel, Israel 4070000

Corresponding author: Noam Hazon (e-mail: noamh@ariel.ac.il).

This research was supported in part by the Ministry of Science, Technology & Space, Israel.

ABSTRACT The traveling salesman game (TSG) consists of dividing the cost of a round trip among several customers. One of the most significant solution concepts in cooperative game theory is the Shapley value, which provides a fair division of the costs for a variety of games including the TSG, based on the marginal costs attributed with each customer. In this paper, we consider efficient methods for computing the Shapley value for the TSG. There exist two major variants of the TSG. In the first variant, there exists a fixed order in which the customers are serviced. We show a method for efficient computation of the Shapley value in this setting. Our result is also applicable for efficient computation of the Shapley value in ride-sharing settings, when a number of passengers would like to fairly split their ride cost. In the second variant, there is no predetermined fixed order. We show that the Shapley value cannot be efficiently computed in this setting. However, extensive simulations reveal that our approach for the first variant can serve as an excellent proxy for the second variant, outperforming the state-of-the-art methods.

INDEX TERMS Shapley value, Traveling salesman games, Vehicle routing problem.

I. INTRODUCTION

An important combinatorial optimization problem is the Vehicle Routing Problem (VRP), where there is a set of customers situated in different locations on a map, and the goal is to find an optimal route for a vehicle to deliver or pick up some goods to the customers [1]. This problem and its variants have been extensively studied over the past 50 years, mainly in the operation research and transportation science communities [2], [3]. In several VRP instances, there is also the need to allocate the cost of the tour among the customers served. For example, a manufacturer that delivers supplies to stores may use the same truck for delivering the supplies to several stores. Each store is required to pay for the delivery, and the total charge should not be more than the total travel cost. The central question is how should the cost be fairly allocated among the stores. Another example may include a travelling circus. In this scenario, a circus traverses many different locations (the customers) according to a predetermined schedule. The problem is again to determine a fair allocation of the travel cost to the customers. We note that in this scenario, it is quite likely that if some customers cancel their show, the circus cannot arrive earlier at other locations. Therefore, the circus may need to follow a tour that is not necessarily the least costly one.

Settings similar to the supplies delivery and the traveling circus, in which the customers are required to cover the travel cost, are formalized as Traveling Salesman Games (TSGs) [4], [5]. In this paper, we study a fair allocation of the cost of the tour for TSGs. We concentrate on the Shapley value [6] as our notion of fair cost allocation. The Shapley value is widely used in cooperative games, and is the only cost allocation satisfying efficiency, symmetry, null player property and additivity. The Shapley value has been even termed the most important normative division scheme in cooperative game theory [7]. However, the Shapley value depends on the travel cost of each subset of the customers. Therefore, as stated by Özener and Ergun [8], "In general, explicitly calculating the Shapley value requires exponential time. Hence, it is an impractical cost-allocation method unless an implicit technique given the particular structure of the game can be found". In the first part of this paper, we tackle this challenge.

There are two variants of TSGs. In the first variant, termed routing games [9], there is a fixed order in which the customers are serviced. In the context of supplies delivery, the order may be attributed to the order in which the requests were received, the urgency of the service, or the frequency of customer usage of the service. In such cases, the order must be preserved when determining the travel cost of the tour with a subset of the customers. In the context of a traveling circus, the order is usually fixed and predetermined. We note that Yengin [9] conjectured that there is no efficient way for computing the Shapley value in routing games. In the more general TSG, there is no predetermined fixed order. In such cases, it is assumed that the tour to a subset of the customers is performed using the shortest (or cheapest) tour that traverses their locations.

In this paper, we show an efficient computation of the Shapley value for routing games. Our method is based on smart enumeration of the components that are used in the computation of the Shapley value. Our approach can be generalized to ride-sharing settings, in which several passengers share a taxi and split the traveling cost. We then move to analyze the more general TSG and show that, unless $P = NP$, there is no polynomial time algorithm for computing the Shapley value. Fortunately, we show through extensive simulations that computing the Shapley value for routing games results in an excellent proxy for the Shapley value of the more general TSGs. We compare our method with three recently studied proxies [10], [11] and show that our method outperforms all of them.

To summarize, the contributions of this paper are two-fold:

1) We show an efficient method for computing the Shapley value of each customer for routing games, which is in contrast to a previous conjecture made in the literature [9].
2) We show that, while there exist no polynomial algorithm for computing the Shapley value of the general TSG (unless $P = NP$), the Shapley value computed for routing games can be used as an excellent proxy for the Shapley value in a TSG, establishing a new state-of-the-art.

II. RELATED WORK

In the first part of our paper, we show an efficient technique for computing the Shapley value in routing games. One variant of routing games is the fixed-route traveling salesman problems with appointments. In this variant, the service provider is assumed to travel back home (to the origin) when she skips a customer. This variant was introduced by Yengin [9], who also showed how to efficiently compute the Shapley value for this problem, but stated that her technique does not carry over to routing games.

The routing game can also be interpreted as a generalization of the airport problem [12] to a two-dimensional plane. In the airport problem, we need to decide how to distribute the cost of an airport runway among different airlines, who need runways of different lengths. In our case, we distribute the cost among customers who need to be serviced at different locations. Indeed, it was shown that the Shapley value can be efficiently computed for the airport problem, however achieving efficient computation of the Shapley value in our setting requires a different technique.

The computation of the Shapley value for the general TSG has rarely received serious attention in the literature, due to its complexity. Notably, Aziz et al. [10] suggested a number of direct and sampling-based procedures for calculating the Shapley value for the TSG. They further surveyed several proxies for the Shapley value that are relatively easy to compute, and experimentally evaluate their performance. Unfortunately, the performance of their proposed proxies is not satisfying, e.g., the deviation from the correct Shapley value is more than 30% in many settings, as we show in Section V-C.

Recently, Popescu and Kilby [11] developed two proxies for computing the Shapley value for the Euclidean TSG, in which the cost associated with any two locations uses the Euclidean distance between their coordinates. Their proxies are also suitable for the general TSG. They experimentally showed that their proxies outperform the best proxies of [10]. Unfortunately, the performance of their first proposed proxy (Appro-1) remains unsatisfying, while their second proposed proxy (Appro-2) requires extensive computation. In this paper, we develop a proxy for the Shapley value for the general TSG problem, which is based on the Shapley value for routing games, and show that it outperforms the current state-of-the-art proxies without requiring extensive computation.

One of the most well-studied applications of the TSG is the domain of shared transportation, in which shippers collaborate and bundle their shipment requests together to achieve better rates from a carrier [13]. However, Özener and Ergun [8] stated that "we do not know of an efficient technique for calculating the Shapley value for the shippers' collaboration game". Indeed, Fiestras-Janeiro et al. [14] developed the line rule, which is inspired by the Shapley value, but requires less computational effort and relates better with the core [15]. However, the line rule is suitable only for a specific inventory transportation problem. Özener [16] described an approximation of the Shapley value when trying to simultaneously allocate both the transportation costs and the emissions among the customers. Frisk et al. [17] and later Sun et al. [18] study axiomatic properties of the Shapley value and other cost sharing methods in the domain of shared transportation. In another domain, Bistaffa et al. [19] introduced a fair payment scheme, which is based on the game theoretic concept of the kernel, for the social ride-sharing problem (where the set of commuters are connected through a social network).

We note that computing the Shapley value is a challenging task in other domains as well, in which scholars try finding efficient methods to estimate the Shapley value. For example, in the field of machine learning, the Shapley value is used for interpreting results obtained by a machine learning model

[20], [21].

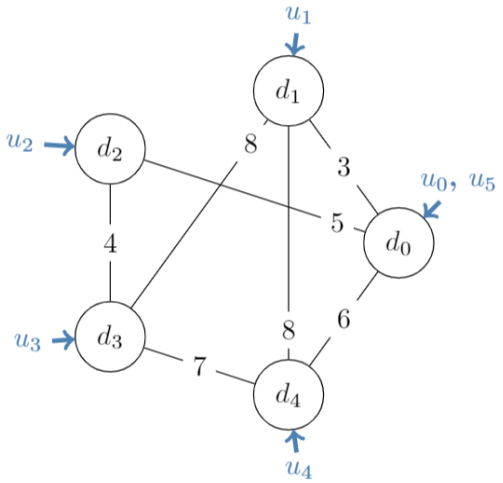
III. PRELIMINARIES

We are given a weighted graph $G(V; E)$ that represents a road network; V is the set of possible locations, and E is a set of weighted edges that represents the set of roads. We are given an ordered set $U = \{u_1; u_2; \ldots; u_n\}$ of customers (users) where customer u_1 must be serviced first, customer u_2 must be serviced second, etc. Each customer u_i has a corresponding location $d_i \in V$. The salesman departures from $d_0 \in V$. Note that we do not require that the set will be ordered according to the optimal order, which minimizes the total tour cost (unlike [5]). We denote by $(u_i; u_j)$ the shortest travel distance between the locations of u_i and u_j in G (d_i and d_j, respectively) and $(u_i; u_i) = 0$. To simplify the notation, we define dummy customers, u_0 and u_{n+1}, associated with d_0. See Figure 1 for a graphical representation of an example with 4 customers. Given a set $S \subseteq U$, let $c(S)$ be the cost associated with the subset S. That is, $c(U)$ is the total cost of the tour. We note that $c(S)$, where $S \subset U$, depends on the order in which the customers are serviced. Therefore, $c(S)$ is defined differently in routing games (in which the original order in U is preserved) and in the traveling salesman game (in which the shortest path is used to determine the order). The Shapley value for a customer u_i is formally defined as [6]:

$$(u_i) = \sum_{S \subseteq U} \frac{(|S| - 1)!(|U| - |S|)!}{|U|!} c(S) - c(S \setminus \{i\}) : \quad (1)$$

That is, the Shapley value is an average over the marginal costs of each customer. For ease of notation, we omit the reference to the function c when referring to the Shapley value, since it is clear from the context.

FIGURE 1. A graphical representation of an example with 4 customers. Note that d_0 is associated with the two dummy customers, u_0 and u_5.



IV. THE SHAPLEY VALUE IN ROUTING GAMES

In this section, we consider the setting of routing games, and efficiently compute the payment for every customer using the Shapley value. Note that this is an unexpected result, since it refutes the conjecture in [9] that there is no efficient way for computing the Shapley value in routing games.

A. NOTATIONS

Given a set $S \subseteq U$, let $Ord(S)$ be the set S ordered in ascending order (according to the order imposed by U), and let $S[i]$ be the customer that is in the i-th position in $Ord(S)$. For ease of notation, we use $S[0]$ and $S[|S| + 1]$ to denote u_0.

Given a set $S \subseteq U$, let $v(S)$ be the shortest travel distance of the tour that starts at the origin d_0, traverses all of the destinations of the customers in S according to an ascending order and returns back to d_0. That is, $v(S) = \sum_{i=0}^{|S|} (S[i]; S[i+1])$. This value ($v(S)$) serves as the cost associated with a subset of customers, $c(S)$, in the computation of the Shapley value. Let R be a permutation on U and let P_i^R be the set of the previous customers to u_i in permutation R.

B. EFFICIENT COMPUTATION OF THE SHAPLEY VALUE

We are interested in determining the payment for each customer, u_i, according to the Shapley value, (u_i). We use the following formula for computing the Shapley value, which is equivalent to Equation 1 [22], to derive an efficient computation.

$$(u_i) = \frac{1}{n!} \sum_{R} v(P_i^R \cup \{u_i\}) - v(P_i^R) : \quad (2)$$

Given a permutation R and a customer u_i, let $u_\ell \in P_i^R$ be a customer such that $\ell < i$ and $\forall u_j \in P_i^R; j \leq \ell$ or $i < j$. If no such customer exists, then u_ℓ is defined as u_0. Similarly, let $u_r \in P_i^R$ be a customer such that $i < r$ and $\forall u_j \in P_i^R; j < i$ or $r < j$. If no such customer exists, then u_r is defined as u_0. We note that for ease of notation, we do not include R and i when referring to u_l and u_r. We use ℓ (and r) to denote the position of u_ℓ (and u_r) in $Ord(P_i^R)$, respectively. If $u_\ell = u_0$ then $\ell = 0$, and if $u_r = u_0$ then $r = |P_i^R| + 1$. We note that $P_i^R[\ell] = u_\ell$, $P_i^R[r] = u_r$ and $r = \ell + 1$.

For example, assume $U = \{u_1; u_2; u_3; u_4; u_5; u_6\}$ and $R = \{u_6; u_2; u_5; u_4; u_3; u_1\}$, we get $P_4^R = \{u_6; u_2; u_5\}$ and thus $Ord(P_4^R) = \{u_2; u_5; u_6\}$, $u_\ell = u_2$ (i.e., $\ell = 1$), $u_r = u_5$ (i.e., $r = 2$), and $P_4^R[\ell] = u_2$. Figure 2 further illustrates the example. See Table 1 for a summary of all notations.

Our first observation is that Equation 2, in our setting, can be rewritten as the sum over the distances between pairs of locations.

Observation 1. $(u_i) = \frac{1}{n!} \sum_{p=0}^{n} \sum_{q=p+1}^{n+1} \alpha_{p;q}^{i} (u_p; u_q)$, *for some* $\alpha_{p;q}^{i} \in \mathbb{Z}$.

FIGURE 2. An illustration of the definitions of u_ℓ and u_r.

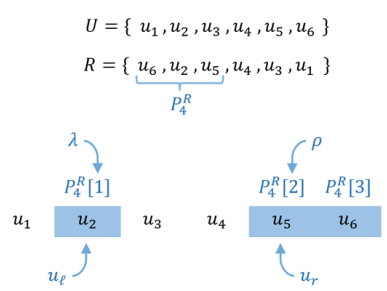


TABLE 1. Notation summary

Notation	Meaning
$U = \{u_1, u_2, \ldots, u_n\}$	An ordered set of customers (users).
$d(u_i, u_j)$	The shortest travel distance between the locations of u_i and u_j in G.
u_i, u_{n+1}	dummy customers associated with d_0.
$C(S)$	the cost associated with the subset S.
$\phi(u_i)$	The Shapley value for a customer u_i.
$Ord(S)$	The set S ordered inascending order, according to the order imposed by U.
$S[i]$	The customer that is in the i-th position in $Ord(S)$.
$v(S)$	$\sum_{j=0}^{\lvert S\rvert} d(S[j], S[j+1])$.
R	A permutation on U.
P_i^R	The set of the previous customers to u_i in permutation R.
u_ℓ	The customer before u_i in $Ord(P_i^R)$.
u_r	The customer after u_i in $Ord(P_i^R)$.
	The position of u_ℓ in $Ord(P_i^R)$.
	The position of u_r in $Ord(P_i^R)$.

Proof. We note that $\phi(u_i)\, n!$ is a sum over $v(S)$ for multiple $S \subseteq U$. By definition, $v(S) = \sum_{j=0}^{\lvert S\rvert} d(S[j], S[j+1])$, such that $S[j] = u_p$ and $S[j+1] = u_q$ where $p < q$. □

We now show that we can rewrite the computation of the Shapley value in our setting as follows.

Lemmaa 1. *For any customer u_i,*

$$\phi(u_i) = \frac{1}{n!} \sum_{R} d(u_\ell, u_i) + d(u_i, u_r) - d(u_\ell, u_r) :$$

Proof. $v(P_i^R) = \sum_{j=0}^{\lvert P_i^R\rvert} d(P_i^R[j], P_i^R[j+1]) = \sum_{j=0}^{\lvert P_i^R\rvert-1} d(P_i^R[j], P_i^R[j+1]) + d(u_\ell, u_r) + \sum_{j=\lvert P_i^R\rvert}^{\lvert P_i^R\rvert} d(P_i^R[j], P_i^R[j+1])$

In addition,

$$v(P_i^R \cup \{u_i\}) = \sum_{j=0}^{\lvert P_i^R\rvert-1} d(P_i^R[j], P_i^R[j+1]) + d(u_\ell, u_i) + d(u_i, u_r) + \sum_{j=\lvert P_i^R\rvert}^{\lvert P_i^R\rvert} d(P_i^R[j], P_i^R[j+1]).$$

By definition,

$$\phi(u_i) = \frac{1}{n!} \sum_{R} v(P_i^R \cup \{u_i\}) - v(P_i^R) =$$

$$\frac{1}{n!} \sum_{R} \sum_{j=0}^{\lvert P_i^R\rvert-1} d(P_i^R[j], P_i^R[j+1]) + d(u_\ell, u_i) + d(u_i, u_r) + \sum_{j=\lvert P_i^R\rvert}^{\lvert P_i^R\rvert} d(P_i^R[j], P_i^R[j+1]) - \sum_{j=0}^{\lvert P_i^R\rvert-1} d(P_i^R[j], P_i^R[j+1]) + d(u_\ell, u_r) + \sum_{j=\lvert P_i^R\rvert}^{\lvert P_i^R\rvert} d(P_i^R[j], P_i^R[j+1]) =$$

$$\frac{1}{n!} \sum_{R} d(u_\ell, u_i) + d(u_i, u_r) - d(u_\ell, u_r) \qquad \square$$

Following Observation 1 and Lemma 1 we now show that we can rewrite the computation of the Shapley value as a sum over distances, that can be computed in polynomial time.

Theorem 1. *For each i, $\phi(u_i) = \sum_{p=0}^{n-1} \sum_{q=i}^{n} \alpha_{p,q}^i \, d(u_p, u_q)$, where $q \neq p$, and $\alpha_{p,q}^i \in \mathbb{Q}$ are computed in polynomial time.*

Proof. By definition, $\ell < i < r$. According to Lemma 1 $\phi(u_i)\, n!$ is a sum over $d(u_p, u_q)$, where $p \neq i \neq q$. There are several terms in this sum:

$\alpha_{0,i}^i$ multiplies $d(u_0, u_i)$. Now, $d(u_0, u_i)$ appears in $\phi(u_i)$ in every permutation R when $u_\ell = u_0$. That is, in all of the permutations where customer u_i appears before any other customer u_x such that $x < i$. We now count the number of such permutations. There are $\binom{n}{i}$ options to place the customers $u_1, u_2, \ldots, u_i$ among the n available positions. For each such option, there are $(i-1)!$ options to order the customers $u_1, u_2, \ldots, u_i$ such that u_i is the first customer among them. Finally, there are $(n-i)!$ options to order the customers $u_{i+1}, u_{i+2}, \ldots, u_n$. Therefore, $d(u_0, u_i)$ appears in $\binom{n}{i} (i-1)! (n-i)! = \frac{n!}{i}$ permutations, and by inserting $\frac{1}{n!}$ into the sum we get that $\alpha_{0,i}^i = \frac{1}{i}$.

For each $q > i$, $\alpha_{0,q}^i$ multiplies $d(u_0, u_q)$. Now, $d(u_0, u_q)$ appears negatively in $\phi(u_i)$ in every permutation R when $u_\ell = u_0$ and $u_r = u_q$. That is, in all of the permutations where customer u_q appears before u_i (i.e., $u_q \in P_i^R$), but any other customer u_x such that $x < q$, appears after u_i. We now count the number of such permutations. There are $\binom{n}{q}$ options to place the customers $u_1, u_2, \ldots, u_i, \ldots, u_q$ among the n available positions. For each such option, there are $(q-2)!$ options to order the customers $u_1, u_2, \ldots, u_i, \ldots, u_q$ such that u_q is the first customer and u_i is the second customer among them. Finally, there are $(n-q)!$ options to order the customers $u_{q+1}, u_{q+2}, \ldots, u_n$. Therefore, $d(u_0, u_q)$ appears negatively in $\binom{n}{q} (q-2)! (n-q)! = \frac{n!}{q(q-1)}$ permutations, and by inserting $\frac{1}{n!}$ into the sum we get that $\alpha_{0,q}^i = \frac{1}{q(q-1)}$.

For each $0 < p < i$, $\alpha_{p,i}^i$ multiplies $d(u_p, u_i)$. Now, $d(u_p, u_i)$ appears in $\phi(u_i)$ in every permutation R when $u_\ell = u_p$. That is, in all of the permutations where

customer u_p appears before u_i (i.e., $u_p \ 2 \ P_i^R$), but any other customer u_x such that $p < x < i$, appears after u_i. We now count the number of such permutations. There are $\binom{n}{i \ p+1}$ options to place the customers $u_p; u_{p+1}; :::; u_i$ among the n available positions. For each such option, there are $(i \ p + 1 \ 2)!$ options to order the customers $u_p; u_{p+1}; :::; u_i$ such that u_p is the first customer and u_i is the second customer among them. Finally, there are $(n \ (i \ p + 1))!$ options to order the customers $u_1; u_2; :::; u_{p \ 1}; u_{i+1}; u_{i+2}; :::; u_n$. Therefore, $(u_p; u_i)$ appears in $\binom{n}{i \ p+1} (i \ p \ 1)! (n \ (i \ p + 1))! = \frac{n!}{(i \ p)(i \ p+1)}$ permutations, and by inserting $\frac{1}{n!}$ into the sum we get that $\binom{i}{p;i} = \frac{1}{(i \ p)(i \ p+1)}$.

For each $q > i$, $\binom{i}{i;q}$ multiplies $(u_i; u_q)$. Now, $(u_i; u_q)$ appears in (u_i) in every permutation R when $u_r = u_q$. That is, in all of the permutations where customer u_q appears before u_i (i.e., $u_q \ 2 \ P_i^R$), but any other customer u_x such that $i < x < q$, appears after u_i. We now count the number of such permutations. There are $\binom{n}{q \ i+1}$ options to place the customers $u_i; u_{i+1}; :::; u_q$ among the n available positions. For each such option, there are $(q \ i + 1 \ 2)!$ options to order the customers $u_i; u_{i+1}; :::; u_q$ such that u_q is the first customer and u_i is the second customer among them. Finally, there are $(n \ (q \ i + 1))!$ options to order the customers $u_1; u_2; :::; u_{i \ 1}; u_{q+1}; u_{q+2}; :::; u_n$. Therefore, $(u_p; u_i)$ appears in $\binom{n}{q \ i+1} (q \ i \ 1)! (n \ (q \ i + 1))! = \frac{n!}{(q \ i)(q \ i+1)}$ permutations, and by inserting $\frac{1}{n!}$ into the sum we get that $\binom{i}{i;q} = \frac{1}{(q \ i)(q \ i+1)}$.

For each $p; q$ such that $p < i < q$, $\binom{i}{p;q}$ multiplies $(u_p; u_q)$. Now, $(u_p; u_q)$ appears negatively in (u_i) in every permutation R when $u_r = u_p$ and $u_r = u_q$. That is, in all of the permutations where customers $u_p; u_q$ appear before u_i (i.e., $u_p; u_q \ 2 \ P_i^R$), but any other customer u_x such that $p < x < q; x \neq i$, appears after u_i. We now count the number of such permutations. There are $\binom{n}{q \ p+1}$ options to place the customers $u_p; u_{p+1}; :::; u_i; :::; u_q$ among the n available positions. For each such option there are $(q \ p+1 \ 3)!$ options to order the customers $u_p; u_{p+1}; :::; u_i; :::; u_q$ such that u_p is the first customer, u_q is the second and u_i is the third customer among them. Similarly, there are $(q \ p + 1 \ 3)!$ options to order these customers such that u_q is the first customer, u_p is the second and u_i is the third. Finally, there are $(n \ (q \ p + 1))!$ options to order the customers $u_1; u_2; :::; u_{p \ 1}; u_{q+1}; u_{q+2}; :::; u_n$. Therefore, $(u_p; u_q)$ appears in $\binom{n}{q \ p+1} 2 (q \ p \ 2)! (n \ (q \ p + 1))! = \frac{2 \, n!}{(q \ p \ 1)(q \ p)(q \ p+1)}$ permutations, and by inserting $\frac{1}{n!}$ into the sum we get that $\binom{i}{p;q} = \frac{2}{(q \ p \ 1)(q \ p)(q \ p+1)}$.

For each $p < i$, $\binom{i}{p;n+1}$ multiplies $(u_p; u_{n+1})$. Now, $(u_p; u_{n+1})$ appears negatively in (u_i) in every permutation R when $u_r = u_p$ and $u_r = u_{n+1}$. That is, in all of the permutations where customer u_p appears before u_i (i.e., $u_p \ 2 \ P_i^R$), but any other customer u_x such that $p < x, x \neq i$, appears after u_i. We now count the number of such permutations. There are $\binom{n}{n \ p+1}$ options to place the customers $u_p; u_{p+1}; :::; u_i; :::; u_n$ among the n available positions. For each such option, there are $(n \ p + 1 \ 2)!$ options to order the customers $u_p; u_{p+1}; :::; u_i; :::; u_n$ such that u_p is the first customer and u_i is the second customer among them. Finally, there are $(p \ 1)!$ options to order the customers $u_1; u_2; :::; u_{p \ 1}$. Therefore, $(u_p; u_{n+1})$ appears negatively in $\binom{n}{n \ p+1} (n \ p + 1 \ 2)! (p \ 1)! = \frac{n!}{(n \ p)(n \ p+1)}$ permutations, and by inserting $\frac{1}{n!}$ into the sum we get that $\binom{i}{p;n+1} = \frac{1}{(n \ p)(n \ p+1)}$.

$\binom{i}{i;n+1}$ multiplies $(u_i; u_{n+1})$. Now, $(u_i; u_{n+1})$ appears in (u_i) in every permutation R when $u_r = u_{n+1}$. That is, in all of the permutations where customer u_i appears before any other customer u_x such that $i < x$. We now count the number of such permutations. There are $\binom{n}{n \ i+1}$ options to place the customers $u_i; u_{i+1}; :::; u_n$ among the n available positions. For each such option, there are $(n \ i + 1 \ 1)!$ options to order the customers $u_i; u_{i+1}; :::; u_n$ such that u_i is the first customer among them. Finally, there are $(i \ 1)!$ options to order the customers $u_1; u_2; :::; u_i$. Therefore, $(u_i; u_{n+1})$ appears in $\binom{n}{n \ i+1} (n \ i)! (i \ 1)! = \frac{n!}{n \ i+1}$ permutations, and by inserting $\frac{1}{n!}$ into the sum we get that $\binom{i}{i;n+1} = \frac{1}{n \ i+1}$. □

We note that routing games are very similar to the setting of the last mile variant of prioritized ride-sharing. In ride-sharing, several passengers share a taxi and split the traveling cost. In the last mile variant, it is assumed that all the passengers are positioned at the same origin location (e.g. an airport), and each has a destination [23]. In the prioritized ride-sharing problem, it is further assumed that there is a fixed priority order in which the passengers are dropped-off (e.g. elderly, disabled, pregnant women, etc.). Clearly, our problem is almost identical: the taxi corresponds to the salesman and the passengers correspond to the customers. The only difference is that in the ride-sharing setting the passengers do not pay the cost of the trip back to the origin. Indeed, the results presented in this section carry over to ride-sharing.

Theorem 2. *The Shapley value in the last mile prioritized ride-sharing problem can be computed in polynomial time.*

Proof. We use our previous definitions and results with the following slight modifications. We re-define the dummy customer u_{n+1} such that it is no longer associated with d_0 and thus for every $i \ 2 \ f0; 1; :::; ng$, $(u_i; u_{n+1}) = 0$. In Observation 1, we need to modify the bound in the outer sum (with the index p) to $n \ 1$ and the bound in the inner sum (with the index q) to n. In addition, we use the proof of Theorem 1, but we modify the bound in the inner sum (with the index q) to n. Therefore, $\binom{i}{p;n+1}$ and $\binom{i}{i;n+1}$ are no longer needed. □

V. THE GENERAL TRAVELING SALESMAN GAME

Similar to routing games we are given an initial order, which the customers are serviced. However, in this variant we do not enforce the initial order for every subset of customers. Instead, given a strict subset of customers S, the cost associated with it, $c(S)$, is the length of the shortest path that traverses all the locations corresponding to the customers in S and traveling back to the origin.

A. THE HARDNESS OF COMPUTING THE SHAPLEY VALUE

In Section IV, we showed that the Shapley value can be efficiently computed for routing games. In essence, the computation could be done efficiently since most of the travel distances cancel out, and only a polynomial number of terms remain in the computation. Unfortunately, this is not the case with TSG, where the Shapley value cannot be computed efficiently unless $P = NP$.

We begin by defining the *len-TSP* problem as the problem of finding the length of the shortest cycle (not necessarily a simple cycle) that starts at a specific node, v_0, traverses all nodes in a graph, and returns to v_0. Clearly, the len-TSP problem cannot be performed in polynomial time, unless $P = NP$[1]. We use len-TSP to show that the Shapley value for the TSG cannot be computed efficiently, unless $P = NP$. We note that Theorem 1 in [10] has a flaw, and therefore cannot be used[2].

Theorem 3. *There is no polynomial time algorithm that computes the Shapley value for a given customer in the TSG unless $P = NP$.*

Proof. Given an instance of the len-TSP on a graph $G(V; E)$ we denote the solution by x. We construct the following instance. We build a graph $G^0(V^0; E^0)$, where we add a node v^0, i.e., $V^0 = V \cup \{v^0\}$. If $e \in E$ then $e \in E^0$ with the same weight, and we also add $(v_0; v^0)$ and $(v^0; v_0)$ to E^0 with a weight of 1. Finally, we set $U = V^0 \cap \{v_0\}$ (where every customer u_i is associated with location v_i), $d_0 = v_0$, and the initial order is arbitrarily chosen. Recall that $c(U)$ is the total travel cost associated with the chosen order. We ask to compute the Shapley value of customer u^0 that is associated with the location v^0.

Clearly, the marginal contribution of u^0 to any strict subset of $U \cap \{u^0\}$ is exactly 2. However, the marginal contribution of u^0 to the complete set $U \cap \{u^0\}$ is exactly $c(U)$ minus x (the length of the shortest cycle starting at v_0, traversing all nodes in V, and returning to v_0). That is,

$$(u^0) = \frac{(jUj \quad 1)!}{jUj!}(c(U) \quad x) + \frac{jUj! \quad (jUj \quad 1)!}{jUj!} \quad 2$$

After some simple mathematical manipulations we get that $x = (jUj \quad 1)2 \quad jUj \quad (u^0) + c(U)$. Therefore, if we can compute (u^0) in polynomial time then we can solve the len-TSP problem in polynomial time, which is not possible unless $P = NP$. □

B. SHAPLEY APPROXIMATION BASED ON A FIXED ORDER

In Section IV, we presented a method for efficiently computing the Shapley value in routing games. In this section, we show that our method may be also applicable to a general TSG as an efficient proxy for the Shapley value. We term our proxy SHAPO: SHapley APproximation based on a fixed Order. Specifically, SHAPO provides a proxy for the Shapley value in TSG by treating the input as an instance of a routing game and applying the formula developed in Theorem 1. More formally,

$$SHAPO(u_i) = \sum_{p=0}^{\quad} \sum_{q=i}^{n+1} {}^{i}_{p;q} \quad (u_p; u_q);$$

where the values of ${}^{i}_{p;q}$ are according to Theorem 1. See Algorithm 1 for an explicit description of SHAPO.

Algorithm 1: SHAPO

Input: An ordered set $U = \{u_1; u_2; \ldots; u_n\}$ of users
An index of a user (i).
A function, , which returns the shortest travel distance between the locations of two users.
Result: A cost associated with user u_i, which serves as a proxy for (u_i).

```
Res  (u0;ui)/i + (ui;un+1)/n i+1
for p = 1 to i  1 do
    Res  Res + (up;ui)/(i p)(i p+1)  (up;un+1)/(n p)(n p+1)
end
for q = i + 1 to n do
    Res  Res + (ui;uq)/(q i)(q i+1)  (u0;uq)/q(q 1)
    for p = 1 to i  1 do
        Res  Res  2 (up;uq)/(q p 1)(q p)(q p+1)
    end
end
return Res
```

[1]The NP-hardness of len-TSP is proved by a reduction from the Hamiltonian path problem. Given a graph $G(V; E)$ to the Hamiltonian path problem, we construct a full graph G^0 where all edges that appear in graph G have a weight of 1, and all edges that do not appear in G have a weight of 2. G^0 includes a new vertex v^0 that has a weight of 1 to all vertices in the graph. v^0 serves as the origin vertex to the len-TSP problem, and G^0 serves as the input graph. Clearly, if the result from the len-TSP problem equals $jVj + 1$, then there exists a Hamiltonian path in G, otherwise, if the result is greater than $jVj + 1$, there is no Hamiltonian path.

[2]The proof shows that if an -approximation of the Shapley value of a location in a TSG exists, for a constant , then it can be used to decide the existence of a Hamiltonian cycle in a graph G. It shows that given G we can construct a transitive closure G^0, in which we can use the -approximation of the Shapley value in order to decide the existence of a Hamiltonian cycle in G. However, consider the following two undirected graphs: G_1 has the following edges: $(v_1; v_2); (v_2; v_3); (v_3; v_4); (v_4; v_1)$. Obviously, there is a Hamiltonian cycle in G_1. G_2, which is a star graph, has the following edges: $(v_1; v_2); (v_1; v_3); (v_1; v_4)$. Clearly, there is no Hamiltonian cycle in G_2. However, the transitive closure of both graphs is the same: a clique of $v_1; v_2; v_3$ and v_4, and thus $G_1^0 = G_2^0$. That is, an -approximation of Shapley on G^0 *does not* decide the existence of a Hamiltonian cycle in G.

We compare SHAPO with the following three state-of-the-art proxies for computing the Shapley value in traveling salesman games.

a: Depot Distance

This method divides the total travel cost proportionally to the distance from the origin (depot), i.e.

$$Depot(u_i) = \frac{(u_0; u_i)}{\sum_{j=1}^{n} (u_0; u_j)} c(U):$$

For example, a customer serviced at a location that is twice as distant from the origin as another customer has to pay twice the cost, regardless of the actual tour. This baseline was selected as it is very simple, yet it was shown to be effective on real data [10].

b: Appro-1

[due to [11]] Appro-1 is a method that generalizes the characteristics of computing the Shapley value on a 1D line to an Euclidean space. Formally, let

$$SC(x_1; x_2; :::; x_k) = \sum_{i=1}^{k} \frac{y_i}{i(i+1)};$$

where the vector $[y_1; y_2; : : : ; y_k]$ represents the vector $[x_1; x_2; : : : ; x_k]$ sorted in descending order of its values. Let $s_{i;j} = (u_0; u_i) + (u_0; u_j) \quad (u_i; u_j)$. Finally, let $IntA1(u_i) = s_{i;i} \quad SC(s_{1;i}; :::; s_{i \quad 1;i}; s_{i+1;i}; :::; s_{n;i})$. Then,

$$Appro\text{-}1(u_i) = \frac{IntA1(u_i)}{\sum_{j=1}^{n} IntA1(u_j)} C(U):$$

Appro-1 was shown to outperform all state-of-the-art proxies for the Shapley value in TSGs [11].

c: Appro-2

[due to [11]] The Appro-2 method is an improvement of Appro-1 that generalizes the shared cost function, SC. This method is more involved and therefore requires additional computation time (see [11] for a complete description of this method). Appro-2 is currently the state-of-the-art method for estimating the Shapley value in TSGs [11].

C. EXPERIMENTAL SETTINGS

In order to evaluate the performance of SHAPO, we compare the payments computed by each of the methods to the ground truth Shapley value. We use two different graphs; the road network of the city of Toulouse and that of New York city. We randomly choose 11 different origins for each graph. The location vertices are randomly sampled using a uniform distribution over all vertices, and each of the methods is evaluated 100 times against the true Shapley value of all customers, resulting in 2200 computations of the Shapley value per customer. Due to the extensive time required to compute the Shapley value we only evaluate the performance of all methods with up-to 15 locations (including the origin). We therefore computed the Shapley value 224; 400 times, in total.

We run two sets of experiments. In the first set of experiments, the set of customers is ordered according to the shortest tour. This is reasonable, since if there is no fixed order, it is very likely that, in order to reduce the overall cost, the service would use the shortest tour (computed once). Note that the other proxies from the literature do not explicitly require the shortest tour since they only compute the induced fractional allocation of the cost of the shortest tour. However, in order to compute an approximation for the Shapley value, the cost of the shortest tour is required for all proxies [10], [11]. In the second set of experiments, we show that SHAPO performs well even when the given order is according to an approximation for the shortest tour (which can be computed efficiently)[3].

D. RESULTS USING AN OPTIMAL TOUR

We evaluate the performance of SHAPO against the three other proxies using 5 different statistical measures (averaged on all 100 iterations). We use $X(u_i)$ to denote the estimated Shapley value by the evaluated proxy.

1) **Percent**: The average percentage of the deviation from the Shapley value. Formally, $Percent = \frac{1}{n} \sum_{i=1}^{n} \frac{jX(u_i) \quad (u_i)j}{(u_i)}$.
2) **MAE**: The mean absolute error, $MAE = \frac{1}{n} \sum_{i=1}^{n} jX(u_i) \quad (u_i)j$.
3) **MSE**: The mean squared error, $MSE = \frac{1}{n} \sum_{i=1}^{n} (X(u_i) \quad (u_i))^2$. This measure gives higher weight to larger deviations.
4) **RMSE**: The root mean squared error, $RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (X(u_i) \quad (u_i))^2}$.
5) **Max-Error**: The maximum deviation among all customers between the real and estimated Shapley value, $Max = \max_{i=1}^{n} (jX(u_i) \quad (u_i)j)$.

TABLE 2. Average percentage of the deviation from the Shapley value (Percent). Averaged over 100 iterations. Lower is better. An asterisk (*) indicates that the difference between SHAPO and the next best proxy is statistically significant.

No. of Locations	SHAPO	Depot Dist.	Appro-1	Appro-2
4	**0.00%**	16.99%	1.90%	**0.00%**
5	0.22%	21.79%	3.33%	**0.15%**
6	**0.57%***	24.93%	4.57%	0.61%
7	**0.93%***	27.12%	5.82%	1.28%
8	**1.29%***	28.62%	6.94%	1.97%
9	**1.65%***	29.20%	8.00%	2.78%
10	**2.08%***	30.37%	8.89%	3.70%
11	**2.41%***	31.60%	9.72%	4.59%
12	**2.78%***	32.27%	10.57%	5.32%
13	**3.11%***	32.59%	11.33%	6.10%
14	**3.43%***	33.07%	12.05%	6.91%
15	**3.71%***	33.76%	12.76%	7.64%
Average	**2.34%**	30.37%	9.35%	4.46%

[3]Simulation software including the code of the algorithms, and the datasets of the two graphs are available online at https://github.com/Chaya-Levinger/ShapleyValue.

TABLE 3. The mean absolute error of the deviation from the Shapley value (MAE). Averaged over 100 iterations. Lower is better. An asterisk (*) indicates that the difference between SHAPO and the next best proxy is statistically significant.

No. of Locations	SHAPO	Depot Dist.	Appro-1	Appro-2
4	**$0.00**	$1.06	$0.15	**$0.00**
5	$0.01	$1.23	$0.22	**$0.01**
6	$0.03	$1.26	$0.26	**$0.02**
7	**$0.04***	$1.27	$0.30	$0.05
8	**$0.05***	$1.25	$0.32	$0.07
9	**$0.06***	$1.21	$0.33	$0.10
10	**$0.07***	$1.18	$0.34	$0.12
11	**$0.08***	$1.17	$0.35	$0.15
12	**$0.09***	$1.14	$0.36	$0.16
13	**$0.09***	$1.10	$0.36	$0.18
14	**$0.1***	$1.08	$0.37	$0.2
15	**$0.1***	$1.06	$0.37	$0.21
Average	**$0.07**	$1.15	$0.33	$0.14

TABLE 4. The mean squared error of the deviation from the Shapley value (MSE). Averaged over 100 iterations. Lower is better. An asterisk (*) indicates that the difference between SHAPO and the next best proxy is statistically significant.

No. of Locations	SHAPO	Depot Dist.	Appro-1	Appro-2
4	**0.000**	1.783	0.055	**0.000**
5	0.002	2.434	0.100	**0.001**
6	**0.004**	2.678	0.130	**0.004**
7	**0.008***	2.806	0.162	0.010
8	**0.011***	2.794	0.182	0.017
9	**0.015***	2.670	0.204	0.025
10	**0.020***	2.615	0.216	0.038
11	**0.023***	2.608	0.222	0.051
12	**0.027***	2.518	0.236	0.061
13	**0.030***	2.400	0.241	0.075
14	**0.034***	2.301	0.254	0.092
15	**0.035***	2.282	0.255	0.149
Average	**0.022**	2.492	0.212	0.06

TABLE 5. The root mean squared error of the deviation from the Shapley value (RMSE). Averaged over 100 iterations. Lower is better. An asterisk (*) indicates that the difference between SHAPO and the next best proxy is statistically significant.

No. of Locations	SHAPO	Depot Dist.	Appro-1	Appro-2
4	**0.000**	1.172	0.163	**0.000**
5	0.014	1.406	0.255	**0.008**
6	0.032	1.499	0.312	**0.029**
7	**0.051***	1.544	0.361	0.059
8	**0.067***	1.552	0.390	0.087
9	**0.083***	1.517	0.418	0.120
10	**0.101***	1.511	0.434	0.156
11	**0.113***	1.510	0.445	0.189
12	**0.125***	1.492	0.461	0.214
13	**0.134***	1.460	0.468	0.238
14	**0.145***	1.432	0.481	0.268
15	**0.151***	1.424	0.485	0.287
Average	**0.105**	1.471	0.425	0.178

TABLE 6. The maximum deviation among all customers between the real and estimated Shapley value (Max-Error). Averaged over 100 iterations. Lower is better. An asterisk (*) indicates that the difference between SHAPO and the next best proxy is statistically significant.

No. of Locations	SHAPO	Depot Dist.	Appro-1	Appro-2
4	**$0.00**	$1.60	$0.22	**$0.00**
5	$0.02	$2.16	$0.40	**$0.01**
6	$0.06	$2.54	$0.52	**$0.05**
7	**$0.09***	$2.84	$0.64	$0.10
8	**$0.13***	$3.04	$0.73	$0.15
9	**$0.17***	$3.14	$0.82	$0.22
10	**$0.21***	$3.32	$0.88	$0.31
11	**$0.25***	$3.44	$0.93	$0.39
12	**$0.28***	$3.53	$0.99	$0.47
13	**$0.32***	$3.59	$1.03	$0.54
14	**$0.35***	$3.61	$1.09	$0.63
15	**$0.38***	$3.73	$1.11	$0.7
Average	**$0.24**	$3.28	$0.89	$0.39

The results are depicted in Tables 2, 3, 4, 5 and 6. The AVG row represents the average across all customers (which is different from the simple average across each column). SHAPO significantly outperforms the other proxies in all measures, with seven or more locations (using a pairwise t-test with = 0:05). Note that the units of MAE and Max-Error are dollars and the average cost per customer was $4:65. That is, as depicted in Table 3, SHAPO deviated by only 7 cents, on average, from the actual Shapley value. The depot distance deviated by $1:15, Appro-1 deviated by 33 cents and Appro-2 deviated by 14 cents. Similarly, the maximal deviation of SHAPO was 24 cents (on average), while the maximal deviation of depot distance, Appro-1 and Appro-2 was $3:28, 89 cents and 39 cents (respectively).

FIGURE 3. Running time, in seconds, required to compute a single instance of the Shapley value (in logarithmic-scale).

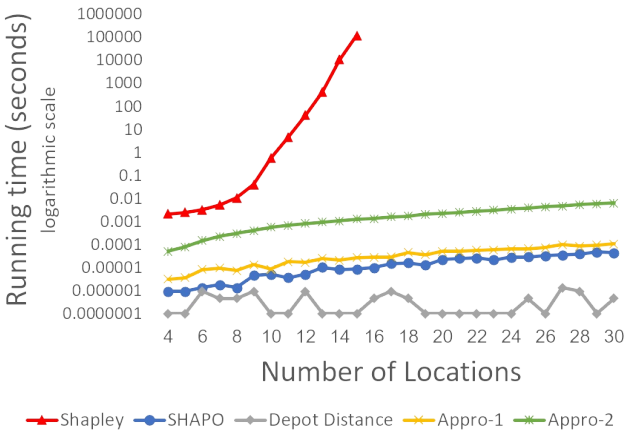


Table 7 shows the average running time of all methods, averaged over 4-15 locations. Figure 3 shows the running time of all methods using a logarithmic scale. We use a 2:2 GHz Intel Xeon CPU with 24 cores for running the simulations; all running time results are reported using a single core. Note that we continue to run SHAPO and the other

TABLE 7. Running time, in seconds, required to compute a single instance of the Shapley value. Averaged over 4 15 locations, 100 iterations, 2 maps and 11 different origins.

	Running time
Shapley	10871.7430584
SHAPO	0.0000043
Depot Dist.	0.0000004
Appro-1	0.0000134
Appro-2	0.0005419

proxies up to 30 locations only for examining the differences in running time. As depicted by the figure, SHAPO can be computed almost instantaneously; even with 15 locations, SHAPO can be computed in 0.0001 seconds, on average. Moreover, SHAPO is between 10 and 100 times faster than Appro-2, and is even faster than Appro-1.

We note that computing the exact Shapley value is done in a reasonable time (up-to 0.005 seconds) when the number of locations is seven or less. Hence, a proxy is needed when the number of customers is larger; therefore, SHAPO is the new state-of-the-art efficient proxy for estimating the Shapley value.

E. SHAPO IN A NEARLY-OPTIMAL TOUR

In this section, we show that the results presented in the previous section carry-out also to situations in which the given tour is very close to being optimal but not necessarily the exact optimal order. Namely, we use a polynomial time heuristic to compute an approximate solution for finding the shortest tour, and then compare the performance of SHAPO to the other proxies. We use the *2-opt* local search algorithm [24], which starts with a random tour and tries to improve it by swapping non-adjacent edges. We limit the number of swapped edges to n^2 for ensuring a polynomial running time. In practice, computing the shortest tour (or its length) once, as required by all the proxies, is not too costly: the average time required to find the optimal shortest tour for 4 to 15 locations was 0.17 seconds, while the average time required to find the tour for 4 to 15 locations with the 2-opt algorithm was only 0.0013 seconds.

TABLE 8. Average percentage of the deviation from the Shapley value (Percent). Averaged over 100 iterations. Lower is better. An asterisk (*) indicates that the difference between SHAPO and the next best proxy is statistically significant.

No. of Locations	SHAPO	Depot Dist.	Appro-1	Appro-2
4	**0.00%**	17.05%	1.86%	**0.00%**
5	0.25%	21.48%	3.39%	**0.18%**
6	0.67%	24.30%	4.71%	**0.65%**
7	**1.20%***	26.73%	5.98%	1.37%
8	**1.78%***	27.79%	7.09%	2.18%
9	**2.28%***	29.64%	8.04%	3.03%
10	**2.96%***	30.32%	9.17%	3.96%
11	**3.57%***	30.77%	10.02%	4.81%
12	**4.13%***	31.50%	11.02%	5.66%
13	**4.63%***	32.03%	11.79%	6.48%
14	**5.18%***	32.73%	12.43%	7.26%
15	**5.75%***	33.14%	13.18%	8.07%
Average	**3.46%**	29.92%	9.65%	4.73%

TABLE 9. The mean absolute error of the deviation from the Shapley value (MAE). Averaged over 100 iterations. Lower is better. An asterisk (*) indicates that the difference between SHAPO and the next best proxy is statistically significant.

No. of Locations	SHAPO	Depot Dist.	Appro-1	Appro-2
4	**$0.00**	$1.08	$0.15	**$0.00**
5	**$0.01**	$1.21	$0.22	**$0.01**
6	**$0.03**	$1.24	$0.27	**$0.03**
7	**$0.05**	$1.25	$0.30	**$0.05**
8	**$0.07***	$1.24	$0.32	$0.08
9	**$0.08***	$1.23	$0.33	$0.11
10	**$0.10***	$1.20	$0.35	$0.13
11	**$0.12***	$1.16	$0.36	$0.16
12	**$0.13***	$1.13	$0.37	$0.17
13	**$0.14***	$1.11	$0.38	$0.19
14	**$0.15***	$1.09	$0.38	$0.21
15	**$0.16***	$1.06	$0.38	$0.23
Average	**$0.11**	$1.15	$0.34	$0.15

TABLE 10. The mean squared error of the deviation from the Shapley value (MSE). Averaged over 100 iterations. Lower is better. An asterisk (*) indicates that the difference between SHAPO and the next best proxy is statistically significant.

No. of Locations	SHAPO	Depot Dist.	Appro-1	Appro-2
4	**0.000**	1.827	0.054	**0.000**
5	0.002	2.383	0.101	**0.001**
6	0.007	2.587	0.137	**0.005**
7	0.014	2.668	0.167	**0.012**
8	0.024	2.757	0.191	**0.020**
9	**0.030**	2.761	0.200	0.031
10	**0.040***	2.694	0.222	0.044
11	**0.052***	2.562	0.235	0.060
12	**0.062***	2.475	0.248	0.070
13	**0.066***	2.408	0.254	0.086
14	**0.077***	2.383	0.266	0.102
15	**0.082***	2.287	0.270	0.122
Average	**0.049**	2.493	0.221	0.062

TABLE 11. The root mean squared error of the deviation from the Shapley value (RMSE). Averaged over 100 iterations. Lower is better. An asterisk (*) indicates that the difference between SHAPO and the next best proxy is statistically significant.

No. of Locations	SHAPO	Depot Dist.	Appro-1	Appro-2
4	**0.000**	1.193	0.163	**0.000**
5	0.016	1.395	0.258	**0.009**
6	0.037	1.473	0.319	**0.031**
7	**0.063**	1.506	0.364	0.064
8	**0.091***	1.532	0.398	0.099
9	**0.110***	1.543	0.416	0.133
10	**0.136***	1.529	0.442	0.167
11	**0.158***	1.496	0.456	0.201
12	**0.180***	1.476	0.472	0.228
13	**0.192***	1.461	0.481	0.256
14	**0.212***	1.456	0.493	0.283
15	**0.225***	1.426	0.499	0.311
Average	**0.148**	1.471	0.434	0.191

The results are depicted in Tables 8, 9, 10, 11 and 12. Similar to the results when using an optimal tour, SHAPO significantly outperforms the other proxies in almost all measures, with nine or more locations. The only exception is the MSE measure, where SHAPO significantly outperforms the other proxies with ten or more locations. Clearly, the

TABLE 12. The maximum deviation among all customers between the real and estimated Shapley value (Max-Error). Averaged over 100 iterations. Lower is better. An asterisk (*) indicates that the difference between SHAPO and the next best proxy is statistically significant.

No. of Locations	SHAPO	Depot Dist.	Appro-1	Appro-2
4	**$0.00**	$1.63	$0.22	**$0.00**
5	$0.02	$2.16	$0.40	**$0.01**
6	$0.06	$2.49	$0.54	**$0.05**
7	**$0.11**	$2.75	$0.65	**$0.11**
8	**$0.17**	$3.00	$0.74	$0.18
9	**$0.22***	$3.19	$0.81	$0.25
10	**$0.28***	$3.34	$0.90	$0.33
11	**$0.35***	$3.42	$0.96	$0.42
12	**$0.41***	$3.50	$1.00	$0.50
13	**$0.46***	$3.58	$1.05	$0.58
14	**$0.53***	$3.69	$1.11	$0.66
15	**$0.56***	$3.74	$1.15	$0.75
Average	**$0.34**	$3.28	$0.91	$0.42

running time of all the proxies when using the near-optimal tour is essentially identical to the running time when using an optimal tour.

The results presented in this section clearly demonstrate that SHAPO significantly outperforms all other proxies for the general TSG, establishing a new state-of-the-art. We believe that SHAPO's performance stems from the fact that it relies on an *exact* solution to a very similar problem, i.e. the routing game.

VI. CONCLUSIONS

The Shapley value is considered one of the most important division scheme of revenues or costs, but its direct computation is often not practical for a reasonable size game. Therefore, Mann and Shapley [25] suggest to consider restrictions and constraints in order to find games where the Shapley value can be efficiently computed. We showed that a routing game is an example of such a game by showing that the Shapley value can be efficiently computed. However, we show that the general TSG cannot be efficiently computed (unless $P = NP$). Interestingly, the method of computing the Shapley value in a routing game can still serve as an efficient proxy for the Shapley value for the TSG.

There are several interesting directions for future work. One possible direction is to adapt our proxy for computing the Shapley value to the domain of sustainable transportation. That is, following the work of Özener [16], we would like to use SHAPO when allocating both the transportation costs and the emissions among the customers. From a theoretical perspective, we showed that computing the Shapley value for the TSG is a hard problem. However, the hardness may be derived also from the hardness of len-TSP. There are several polynomial time approximations and heuristics for TSP that can be adjusted for len-TSP. It is thus interesting to analyze the computational complexity of finding the Shapley value, where $c(S)$ is computed using one of these approximations or heuristics.

ACKNOWLEDGMENT

We would like to thank Dan Popescu and Philip Kilby for providing their code, which was used to run Appro-1 and Appro-2.

REFERENCES

[1] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.
[2] H. N. Psaraftis, M. Wen, and C. A. Kontovas, "Dynamic vehicle routing problems: Three decades and counting," *Networks*, vol. 67, no. 1, pp. 3–31, 2016.
[3] Y. Molenbruch, K. Braekers, and A. Caris, "Typology and literature review for dial-a-ride problems," *Annals of Operations Research*, vol. 259, no. 1, pp. 295–325, 2017.
[4] P. Fishburn and H. Pollak, "Fixed-route cost allocation," *The American Mathematical Monthly*, vol. 90, no. 6, pp. 366–378, 1983.
[5] J. A. Potters, I. J. Curiel, and S. H. Tijs, "Traveling salesman games," *Mathematical Programming*, vol. 53, no. 1-3, pp. 199–211, 1992.
[6] L. S. Shapley, "A value for n-person games," *Contributions to the Theory of Games*, vol. 2, no. 28, pp. 307–317, 1953.
[7] E. Winter, "The Shapley value," *Handbook of game theory with economic applications*, vol. 3, pp. 2025–2054, 2002.
[8] O. Ö. Özener and Ö. Ergun, "Allocating costs in a collaborative transportation procurement network," *Transportation Science*, vol. 42, no. 2, pp. 146–165, 2008.
[9] D. Yengin, "Characterizing the shapley value in fixed-route traveling salesman problems with appointments," *International Journal of Game Theory*, vol. 41, no. 2, pp. 271–299, 2012.
[10] H. Aziz, C. Cahan, C. Gretton, P. Kilby, N. Mattei, and T. Walsh, "A study of proxies for Shapley allocations of transport costs," *Journal of Artificial Intelligence Research*, vol. 56, pp. 573–611, 2016.
[11] D. C. Popescu and P. Kilby, "Approximation of the shapley value for the euclidean travelling salesman game," *Annals of Operations Research*, vol. 289, no. 2, pp. 341–362, 2020.
[12] S. C. Littlechild and G. Owen, "A simple expression for the shapley value in a special case," *Management Science*, vol. 20, no. 3, pp. 370–372, 1973.
[13] M. Guajardo and M. Rönnqvist, "A review on cost allocation methods in collaborative transportation," *International transactions in operational research*, vol. 23, no. 3, pp. 371–392, 2016.
[14] M. G. Fiestras-Janeiro, I. García-Jurado, A. Meca, and M. A. Mosquera, "Cost allocation in inventory transportation systems," *TOP*, vol. 20, no. 2, pp. 397–410, 2012.
[15] R. J. Aumann, "The core of a cooperative game without side payments," *Transactions of the American Mathematical Society*, vol. 98, no. 3, pp. 539–552, 1961.
[16] O. Ö. Özener, "Developing a collaborative planning framework for sustainable transportation," *Mathematical Problems in Engineering*, vol. 2014, 2014.
[17] M. Frisk, M. Göthe-Lundgren, K. Jörnsten, and M. Rönnqvist, "Cost allocation in collaborative forest transportation," *European Journal of Operational Research*, vol. 205, no. 2, pp. 448–458, 2010.
[18] L. Sun, A. Rangarajan, M. H. Karwan, and J. M. Pinto, "Transportation cost allocation on a fixed route," *Computers & Industrial Engineering*, vol. 83, pp. 61–73, 2015.
[19] F. Bistaffa, A. Farinelli, G. Chalkiadakis, and S. D. Ramchurn, "Recommending fair payments for large-scale social ridesharing," in *Proceedings of the 9th ACM Conference on Recommender Systems*, 2015, pp. 139–146.
[20] R. Mitchell, J. Cooper, E. Frank, and G. Holmes, "Sampling permutations for shapley value estimation," *arXiv preprint arXiv:2104.12199*, 2021.
[21] I. Covert and S.-I. Lee, "Improving kernelshap: Practical shapley value estimation using linear regression," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 3457–3465.
[22] G. Chalkiadakis, E. Elkind, and M. Wooldridge, "Computational aspects of cooperative game theory," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 5, no. 6, pp. 1–168, 2011.
[23] S.-F. Cheng, D. T. Nguyen, and H. C. Lau, "Mechanisms for arranging ride sharing and fare splitting for last-mile travel demands," in *Proceedings of the 13th International Conference on Autonomous Agents and Multi-agent Systems*, 2014, pp. 1505–1506.
[24] G. A. Croes, "A method for solving traveling-salesman problems," *Operations research*, vol. 6, no. 6, pp. 791–812, 1958.