Max-Sum with Quadtrees for Decentralized Coordination in Continuous Domains

Dimitrios Troullinos , **Georgios Chalkiadakis** , **Vasilis Samoladas** and **Markos Papageorgiou**
Technical University of Crete, Chania, Greece
dtroullinos@dssl.tuc.gr, gehalk@intelligence.tuc.gr, vsam@softnet.tuc.gr, markos@dssl.tuc.gr

Abstract

In this paper we put forward a novel extension of the classic Max-Sum algorithm to the framework of Continuous Distributed Constrained Optimization Problems (Continuous DCOPs), by utilizing a popular geometric algorithm, namely Quadtrees. In its standard form, Max-Sum can only solve Continuous DCOPs with an a priori discretization procedure. Existing Max-Sum extensions to continuous multiagent coordination domains require additional assumptions regarding the form of the factors, such as access to the gradient, or the ability to model them as continuous piecewise linear functions. Our proposed approach has no such requirements: we model the exchanged messages with Quadtrees, and, as such, the discretization procedure is dynamic and embedded in the internal Max-Sum operations (addition and marginal maximization). We apply Max-Sum with Quadtrees to lane-free autonomous driving. Our experimental evaluation showcases the effectiveness of our approach in this challenging coordination domain.

1 Introduction

The framework of Distributed Constraint Optimization Problems (DCOPs) [Modi *et al.*, 2003] is associated with a broad range of methodologies for distributed problem solving and coordination, such as Factor Graphs [Kschischang *et al.*, 2001] and the Max-Sum message passing algorithm [Farinelli *et al.*, 2008]. Continuous DCOPs [Stranders *et al.*, 2009; Sarker *et al.*, 2021; Hoang *et al.*, 2020], in which the control variables' domain is continuous, are of particular interest corresponding as they do to a multitude of real-world settings, and posing several issues and challenges originating from the continuity of the problem domains. Existing work on Max-Sum in continuous domains [Stranders *et al.*, 2009; Voice *et al.*, 2010] makes certain assumptions regarding the factors, such as the ability to model them in a specific form, or having access to the gradients. In this paper, we propose a novel, more flexible extension of the Max-Sum algorithm for continuous domains: one that lifts such requirements, and can be easily adapted to any Continuous DCOP.

To this end, we utilize Quadtrees [Finkel and Bentley, 1974], which are among the simplest geometric algorithms for data representation of a given set of points, but exhibit in general high efficiency in many applications. Their simplicity makes them more desirable than other existing approximations tied with Max-Sum, such as Continuous Piecewise Linear Functions (CPLFs), which are more restrictive both in terms of construction (with simplices) and management. Specifically, we *embed* Quadtrees in the Max-Sum algorithm, to effectively achieve a fine-grained discretization of the continuous domain—the degree of which is determined by the Quadtree function approximation process itself, in an "online", dynamic fashion. The Quadtrees' embedding in Max-Sum, is achieved via properly re-defining the necessary core Max-Sum operations (*addition* and *marginal maximization*) that are applied on the messages exchanged in order to achieve DCOP optimization.

Now, an emergent highly complex distributed coordination domain is the field of autonomous driving [Pendleton *et al.*, 2017]. The need for coordination is particularly intense when *lane-free vehicle movement* is assumed, as is the case in novel traffic flow paradigms [Papageorgiou *et al.*, 2021]. In lane-free traffic, vehicles are no longer restricted by specific lane positioning, thus they do not perform any lane-changing operation, but rather adjust their lateral placement smoothly. This inherently continuous domain enables much higher utilization of the available road capacity, and gives rise to the design of novel techniques and methodologies, as evident by the results of [Papageorgiou *et al.*, 2021; Troullinos *et al.*, 2021a; Malekzadeh *et al.*, 2021]. A Factor Graph representation with vehicles in the roles of agents controlling acceleration-related variables is natural for this challenging dynamic coordination domain, and allows the use of DCOPs optimization algorithms such as Max-Sum.

Against this background, our main contributions are as follows: we put forward *Max-Sum with Quadtrees (Max-Sum-Quad)*, a novel Max-Sum variant appropriate for any Continuous DCOP domain of interest; the proposed approach does not make use of restrictive assumptions regarding the form of the factors (e.g., access to their gradient, or modelling them as CPLFs), and does not require the determination of a fixed discretized subdomain of the continuous problem at hand; we apply our approach to the lane-free traffic multiagent coordination domain; our experimental evaluation showcases the

effectiveness and adaptability of Max-Sum-Quad in this challenging and dynamic domain.

2 Background and Related Work

Consider a *Factor Graph* [Kschischang *et al.*, 2001] representation of agents, meaning we have a factored representation of an objective function $F(\mathbf{x}) = \sum_{j=1}^{N_F} F_j(\mathbf{s}_j)$, $\mathbf{s}_j \subseteq \mathbf{x}$, to be maximized for a given problem. We seek to obtain the configuration vector $\mathbf{x}^* = \arg\max_{\mathbf{x}} \sum_{j=1}^{N_F} F_j(\mathbf{s}_j)$, where N_F is the number of different factors F_j, and each factor j is connected with a subset of control variables $\mathbf{s}_j$. The Max-Sum algorithm [Farinelli *et al.*, 2008] provides a solution for this distributed optimization problem via message-passing. This requires two key operations, namely an *addition* and a *marginal maximization* one, considering a discrete domain for control variables $x_i \in \mathbf{x}$. First, let us briefly present and discuss the standard algorithm of Max-Sum.

2.1 The Max-Sum Algorithm

Consider a Factor Graph representation as the one in Fig. 1. Note that factors F_j connect different variables, and may potentially associate more than two (e.g., F_4). The corresponding vector $\mathbf{s}_j$ contains all variables x_i that are connected to factor F_j. For instance, on this figure: $\mathbf{s}_4 = [x_2, x_3, x_4]^T$. Thus, in general, the factors can depend on any subset $\mathbf{s}_j \subseteq \mathbf{x}$ of the control variables. To avoid confusion, notice that the indexing of factors and variables is different. The process involves a message-passing operation, and two types of messages are required:

From variable i to factor j:

$$q_{i\to j}(x_i) \quad = \quad a_{ij} + \sum_{k \in M_i \setminus j} r_{k\to i}(x_i) \qquad (1)$$

where M_i is the set of factor indices that variable i is connected to. For example, in Fig. 1, $M_2 = \{2, 3, 4\}$.

So, $q_{i\to j}(x_i)$ measures an estimate of each value of x_i which is sent to factor j. Essentially, $q_{i\to j}(x_i)$ propagates what is received from all the other factors $M_i \setminus j$, and a_{ij} is a normalization constant that satisfies $\sum_{x_i} q_{i\to j}(x_i) = 0$. This normalization is important when the Factor Graph contains cycles since message values would accumulate indefinitely. Convergence in cyclic factor graphs is not guaranteed, but the use of a normalization factor has proven to be quite effective in many studies [Kok and Vlassis, 2006; Murphy *et al.*, 1999; Farinelli *et al.*, 2008].

From factor j to variable i:

$$r_{j\to i}(x_i) \quad = \quad \max_{\mathbf{s}_j \setminus x_i} \Big[F_j(\mathbf{s}_j) + \sum_{k \in N_j \setminus i} q_{k\to j}(x_k) \Big] \qquad (2)$$

where N_j is the set that contains the variable indices connected to factor j, and the maximization process involves the all the variables $\mathbf{s}_j$ connected to factor F_j (besides x_i).

For instance, in Fig. 1, $N_2 = \{1, 2\}$, i.e., variables x_1, x_2 are connected to factor F_2. Now, $r_{j\to i}(x_i)$ is what each factor F_j sends to a connected variable x_i (for each discrete value of x_i), which is the maximum value considering that all other variables x_k of factor F_j will be tuned so as to maximize both F_j and the corresponding message sent from k to factor j ($q_{k\to j}(x_i)$). The message-passing process is illustrated in Fig. 1.

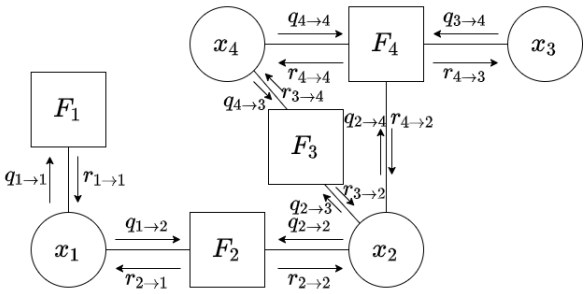

Figure 1: An example of a Factor Graph. Each iteration of Max-Sum involves the message exchange process between factors and variables.

The process described above is iterative, where in every iteration we utilize the last evaluations for all messages $q_{i\to j}(x_i), r_{j\to i}(x_i)$ (we may initiate the process with neutral messages). This operation is performed until either a stopping criterion is met (e.g., time, iterations number), or if all message values converge (given a threshold value).

Finally, each agent can compute the optimal value of x_i^*, as: $x_i^* = \arg\max_{x_i} \sum_{j \in M_i} r_{j\to i}(x_i)$, i.e., each agent selects the value that maximizes the received messages from all nearby factors.

2.2 Related Work

To the best of our knowledge, there are two distinct methodologies to extend Max-Sum to continuous domains. The first one involves the use of CPLFs to represent the factors [Stranders *et al.*, 2009] and by extension the messages that agents exchange. The authors formally define the addition and marginal maximization operations for CPLFs, and can therefore apply Max-Sum with such a function representation. However, the formulation in [Stranders *et al.*, 2009] requires each factor to be modelled as a CPLF. By contrast, in our work factors can have any form; and embedding Quadtrees in Max-Sum equips the latter with a flexible, *dynamic discretization ability* that is key for effective performance in demanding Continuous DCOP settings.

The second line of research involves a hybrid approach [Voice *et al.*, 2010] that utilizes the standard algorithm of Max-Sum, relying on discretization of the continuous values domain; but with the discretized points being updated in each iteration using non-linear programming techniques. This process requires access to each factor's gradient. More recent literature regarding solving Continuous DCOPs involves different methodologies, such as Local Search [Sarker *et al.*, 2021], Distributed Pseudo-tree Optimization Procedures tied with analytical solutions when possible [Hoang *et al.*, 2020], Particle Swarm Algorithms [Choudhury *et al.*, 2020] and even Bayesian Methods [Fransman *et al.*, 2019].

Regarding the lane-free traffic application domain, many

works already exist that propose relevant vehicle movement strategies [Papageorgiou *et al.*, 2021; Troullinos *et al.*, 2021a; Yanumula *et al.*, 2021]. Other work in traffic-related literature decomposing the multiagent system via a graph structure includes [Kuyer *et al.*, 2008; van der Pol and Oliehoek, 2016] that tackle urban traffic lights control problems via a Coordination Graph [Guestrin *et al.*, 2002] scheme; and [Yu *et al.*, 2020] that uses coordination graphs for *lane-based* collaborative autonomous driving.

2.3 Quadtree Representation of a Function

Consider a real function $f(\mathbf{x})$, where $f : \mathbb{R}^n \rightarrow \mathbb{R}$, and all variables $x_i \in \mathbf{x}$ lie within certain bounds, specifically, $\forall x_i \in \mathbf{x}, 0 \leq x_i \leq 1$. If $\mathbf{x} \in \mathbb{R}^2$, the space $\mathbb{R}^2$ is a unit square, and a Quadtree [Har-Peled, 2011] can establish a representation $f^{(Qt)}(\mathbf{x})$ of function $f(\mathbf{x})$, meaning that a tree structure will approximate $f(\mathbf{x})$. This operation is performed by recursively partitioning a 2-dimensional unit square region into 4 subregions. Now, if $\mathbf{x} \in \mathbb{R}^N$, a generalized version of a Quadtree can be utilized, that recursively partitions a N-dimensional unit hypercube region into 2^N subregions. The same principle applies even to 1-D, by partitioning the space into 2 subregions each time.

The process initiates from the root node r of the tree to be constructed, which obtains the value of the center point within the N-dimensional unit hypercube region S_r, where $N = dim(\mathbf{x}_m)$. Then, the root node is partitioned into 2^N child nodes c, corresponding to subregions of the unit hypercube, and they obtain the value of their respective center point $\mathbf{x}_m$; and recursively perform the same operation.

A threshold value ϵ_q dictates the finalization of this operation. Any given node p with center point (of its region S_p) $\mathbf{x}_p$ will stop further partitioning, if the condition $|f(\mathbf{x}_p) - f(\mathbf{x}_c)| \leq \epsilon_q$ is met for the center point ($\mathbf{x}_c$), which in turn is associated with the partitioned subregion S_c of a child node c. This criterion essentially provides us the dynamic aspect of the algorithm, determining both the position of the discrete points and the number of them according to the granularity of the approximating function $f()$.[1] The process outlined above is used for image compression with Quadtrees [Shusterman and Feder, 1994]. Figure 2 exhibits a simple example of a Quadtree approximation for a given function in a 2-dimensional surface. In this Figure, (a) showcases the actual function and a projection of the Quadtree approximation that shows the discrete points selected along with the regions, while (b) visualizes the tree depth of each region and the respective points. Finally, (c) indicates how the Quadtree is actually formed, and can be easily traversed.

Algorithm 1 lists the entire procedure [Shusterman and Feder, 1994], as outlined above. Notice that additional variables are included, namely $minD, maxD$ corresponding to the minimum and maximum tree depth respectively, i.e., the resulting Quadtree function's leaf nodes are bounded within a predefined range of depths. In more detail, the leaf nodes' depth indicates the size of the subregion, and thus the resolution of the approximated function, meaning that as the depth

[1] As we discuss in Sec. 3, this translates to the computation of the messages in Max-Sum, using this dynamic discretization procedure.

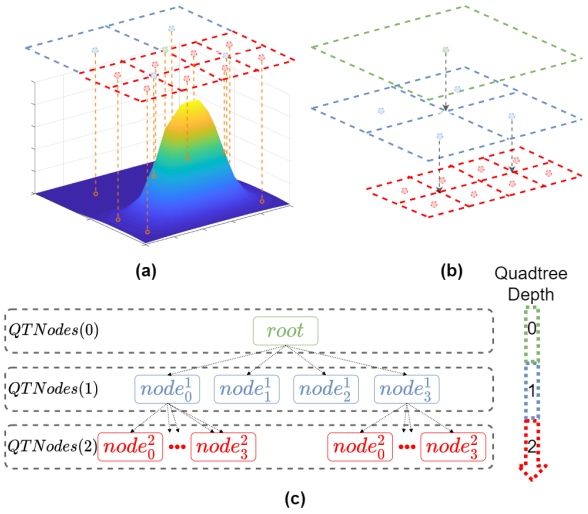

Figure 2: Example of Quadtree Function Approximation.

Algorithm 1 Construct a Quadtree Function

function QT($f(), reg, curD, minD, maxD, \epsilon_q$)
  $\mathbf{x}_p \leftarrow getCenterPointOfRegion(reg)$
  $curNode \leftarrow newNode(curD, reg, f(\mathbf{x}_p))$
  $QTNodes(curD) \leftarrow QTNodes(curD) \cup curNode$
  if $curD \geq maxD$ **then**
    return $curNode$
  end if
  for each $chReg \in partition(reg)$ **do**
    $\mathbf{x}_c \leftarrow getCenterPointOfRegion(chReg)$
    $conv \leftarrow |f(\mathbf{x}_p) - f(\mathbf{x}_c)| \leq \epsilon_q$
    if $(conv \wedge (curD \geq minD))$ **then**
      continue
    end if
    $chNode \leftarrow$QT($f(), chReg, curD + 1, \ldots$)
    $curNode \rightarrow$ADDCHILD($chNode$)
  end for
  return $curNode$
end function

of the tree increases, the function's resolution becomes better. However, this constitutes a trade-off with respect to time and memory needed for this operation. Hence the reason we wish to have control over the minimum and maximum depth of the tree, guaranteeing a minimum resolution of the approximating function, and upper bounding the time and memory required for the approximation procedure.

Provided with a Quadtree representation $f^{(Qt)}(\mathbf{x})$, obtaining the value of any point $\mathbf{x}$ within the available region is equivalent to performing a search operation on a B-tree, where we start from the root node, and traverse the tree based on the corresponding subregions that contain $\mathbf{x}$. Essentially, we obtain the value $f(\mathbf{x}_c)$ of the closest center point $\mathbf{x}_c$ from the available tree nodes (not necessarily a leaf node).

Thus, given a function $f(\mathbf{x})$, we can use Algorithm 1 to construct an approximate function $f^{(Qt)}(\mathbf{x})$, and then we can obtain an approximation of $y = f^{(Qt)}(\mathbf{x})$ for every $\mathbf{x}$.

3 Max-Sum with Quadtrees

Messages in Max-Sum, i.e., the $q_{i\to j}(x_i), r_{j\to i}(x_i)$ functions, are 1-D, since they are associated with a single scalar variable x_i (which is discrete) [Farinelli *et al.*, 2008]. However, in most multiagent settings of interest, agents have more than one variables under their control—and in many cases the variables are continuous. For instance, consider the application domain of interest in our work here, that is,lane-free autonomous driving, where vehicles need to coordinate their movement. In such an environment, each vehicle (agent) needs to control two continuous valued variables, one for the longitudinal (gas/break) movement, and one for the lateral (steering wheel) axis. As such, the message functions can entail more information via having a vector $\mathbf{x}_i$ as input.

So, to utilize Max-Sum in continuous domains requires us to represent a continuous function of multiple variables; we do not restrict to pairwise factors [Stranders *et al.*, 2009], thus the messages can have a high dimensionality. Thus, the use of Quadtrees for calculating these messages adds much desired flexibility regarding the dimensionality of variables and messages. We should mention here that a trade-off emerges from this flexibility, since having N-dimensional variables adds to the complexity of performing the (addition and marginal maximization) operations needed in Max-Sum. However, this trade-off is relevant to the problem at hand, therefore having such flexibility is not in any way limiting. To this end, given a Factor Graph representation of agents, we now model each agent i controlling a set of variables $\mathbf{x}_i \subseteq \mathbf{x}$, and thus all functions corresponding to the messages are now sent between factors j and agents i with dimension $X_i = |\mathbf{x}_i|$.

Moreover, we need to define appropriately how the *addition* and the *marginal maximization* operations [Stranders *et al.*, 2009] are carried out given the Quadtrees representation and the vector input of the messages. To begin, let us reformulate the Max-Sum equations [Farinelli *et al.*, 2008] using a *vector* input $\mathbf{x}_i$ for all messages. Consider a factor $F_j(\mathbf{s}_j)$ that depends on $|M_j|$ variables, and one of the two core Max-Sum equations—the one corresponding to the messages sent from a factor j to a connected agent i:

$$r^{(Qt)}_{j\to i}(\mathbf{x}_i) \quad = \quad \max_{\mathbf{s}_j \setminus \mathbf{x_i}} \Big[F_j(\mathbf{s}_j) + \sum_{k \in N_j \setminus i} q^{(Qt)}_{k\to j}(\mathbf{x}_k) \Big] \quad (3)$$

where (Qt) indicates that these functions have the form of a Quadtree representation of a real multivariate function $f : \mathbb{R}^N \to \mathbb{R}$.

Now, the whole process initiates from factor $F_j(\mathbf{s}_j)$, which is a real multivariate function $F_j : \mathbb{R}^n \to \mathbb{R}$, with n being the size of vector $\mathbf{s}_j$. The union of vectors $\{\mathbf{x}_k : k \in N_j \setminus i\}$ is in fact the vector $\mathbf{s}_j \setminus \mathbf{x}_i$, i.e., $\cup_{k \in N_j \setminus i} \mathbf{x}_k = \mathbf{s}_j \setminus \mathbf{x}_i$, meaning that a function that depends on $\mathbf{s}_j$ entails all variables within this union. Therefore, the following summation process:

$$Q^{(Qt)}_{sum}(\mathbf{s}_j) \quad = \quad \sum_{k \in N_j \setminus i} q^{(Qt)}_{k\to j}(\mathbf{x}_k) \quad (4)$$

also approximates (with a Quadtree) a real multivariate function $Q_{sum} : \mathbb{R}^n \to \mathbb{R}$. Consequently, the operation of *addition* needs to be defined for Quadtrees in order to calculate Eq. 4, along with the following:

$$R^{(Qt)}_{j\to i}(\mathbf{s}_j) \quad = \quad F_j(\mathbf{s}_j) + Q^{(Qt)}_{sum}(\mathbf{s}_j) \quad (5)$$

where $R^{(Qt)}_{j\to i}(\mathbf{s}_j)$ is again a Quadtree, and depends on the same vector (set of variables) $\mathbf{s}_j$ as the two addend functions.

The final step in Eq. 3 is to maximize $R^{(Qt)}_{j\to i}(\mathbf{s}_j)$ w.r.t. variables $\mathbf{x}_i$, thus the operation of a marginal maximization needs to be defined:

$$r^{(Qt)}_{j\to i}(\mathbf{x}_i) \quad = \quad \max_{\mathbf{s}_j \setminus \mathbf{x_i}} \Big[R^{(Qt)}_{j\to i}(\mathbf{s}_j) \Big] \quad (6)$$

which yields the final result for the message function to be exchanged $r^{(Qt)}_{j\to i}(\mathbf{x}_i)$, in the form of a Quadtree. Likewise, for messages sent from an agent i to a connected factor j, we have from [Farinelli *et al.*, 2008]:

$$q^{(Qt)}_{i\to j}(\mathbf{x}_i) \quad = \quad a_{ij} + \sum_{k \in M_i \setminus j} r^{(Qt)}_{k\to i}(\mathbf{x}_i) \quad (7)$$

Here, we need to perform a summation of:

$$R^{(Qt)}_{sum}(\mathbf{x}_i) \quad = \quad \sum_{k \in M_i \setminus j} r^{(Qt)}_{k\to i}(\mathbf{x}_i) \quad (8)$$

involving many variable representations of $r^{(Qt)}_{k\to i}$, but they all depend on the same variables $\mathbf{x}_i$. So, the summation here results also in a two variable representation $R^{(Qt)}_{sum}(\mathbf{x}_i)$, and we obtain the final result $q^{(Qt)}_{i\to j}(\mathbf{x}_i)$ by adding a normalization constant a_{ij} to it.

Evidently, in the case where $\mathbf{x}_i$ is actually in a continuous domain, and not discretized, this summation process is not quite clear, as we do not have a specific set of values for $\mathbf{x}_i$, but rather a range of values. Therefore, this issue depends on the representation of real functions that is chosen, hence this choice dictates the calculation of a_{ij}. We show how to estimate a_{ij} in Sec. 3.3.

Example 1 (Illustration on a simple Factor Graph)**.** *We use the Factor Graph representation in Fig. 3 to showcase the use of Quadtrees within Max-Sum. We have agents* $\{1, 2\}$*, and a factor* F_1 *connected only to agent* 1*, while* F_2 *is connected to both agents. We initialize all messages to* 0*, therefore the first calculation of all* $q^{(Qt)}_{i\to j}(\mathbf{x}_i)$ *messages (Eq. 7) is again* 0*, which is depicted as a Quadtree covering the whole region, with a single point with* 0 *value at its center. Then,* $r^{(Qt)}_{j\to i}(\mathbf{x}_i)$ *messages are calculated (Eq. 3), and a respective Quadtree is constructed according to connected factor* j *of each message. Notice the* dynamic *discretization process, with the point selection (and size) decided in an online fashion.*

Moving on to the second iteration, $q^{(Qt)}_{i\to j}(\mathbf{x}_i)$ *messages are updated, essentially propagating the received messages of the corresponding agent* i*, meaning that the resulting Quadtrees in fact directly propagate messages, i.e.,* $q^{(Qt)}_{1\to 2}(\mathbf{x}_1)$ *at the second iteration propagates* $r^{(Qt)}_{1\to 1}(\mathbf{x}_1)$*, and similarly* $q^{(Qt)}_{1\to 1}(\mathbf{x}_1)$ *propagates* $r^{(Qt)}_{2\to 1}(\mathbf{x}_1)$*. Accordingly,* $q^{(Qt)}_{2\to 2}(\mathbf{x}_2)$ *remains zero,*

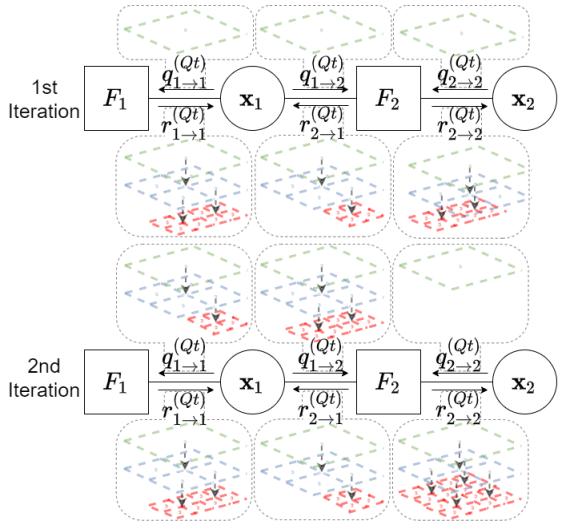

Figure 3: Message passing operations within Max-Sum-Quad.

since no other factor is connected to agent 2*, thus no other messages are there to be propagated.*

Finally, we update $r^{(Qt)}_{j\to i}(\mathbf{x}_i)$ *messages. Now,* $r^{(Qt)}_{1\to1}(\mathbf{x}_1)$ *message remains the same since* F_1 *is not connected to other agents, and consequently there is no other incoming message to be propagated. Similarly for* $r^{(Qt)}_{2\to1}(\mathbf{x}_1)$*. While* F_2 *is connected also to* x_2*,* $q^{(Qt)}_{2\to2}(\mathbf{x}_2) = 0$ *across all region, therefore this will not affect the resulting Quadtree for this message. However, this is not the case for* $r^{(Qt)}_{2\to2}(\mathbf{x}_2)$*, since message* $q^{(Qt)}_{1\to2}(\mathbf{x}_1)$ *is now updated, and consequently it will result in a Quadtree with a potentially different set of points to account for the different granularity of the approximated message, stemming from the operations needed to calculate it.*

3.1 Addition Operation for Quadtree Functions

Consider two Quadtree representations $f^{(Qt)}_1(\mathbf{x})$ and $f^{(Qt)}_2(\mathbf{x})$. We are interested to construct a Quadtree representation $g^{(Qt)}(\mathbf{x})$ that approximates: $g(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x})$. The simplest way to do that is to start constructing the tree of $g^{(Qt)}(\mathbf{x})$, as one would do for a real function (see Algorithm 1), but instead of using a real function $f(\mathbf{x})$ to obtain values for each node, one can use the Quadtree search operation (see Sec. 2.3) to obtain the values of $v_1 = f^{(Qt)}_1(\mathbf{x})$, $v_2 = f^{(Qt)}_2(\mathbf{x})$ and add these to obtain the value $v_{node} = v_1 + v_2$ of each node. Therefore, we utilize Algorithm 1, and set as input function: $f(\mathbf{x}) = f^{(Qt)}_1(\mathbf{x}) + f^{(Qt)}_2(\mathbf{x})$. By so doing, the resulting Quadtree would indeed approximate the addition operation between two Quadtree representations. In a similar manner, we can perform addition between a quadtree representation and a real function (a process that is imposed by Eq. 5).

3.2 Marginal Maximization for Quadtrees

The Marginal Maximization operation is vital in the process of Max-Sum. Let us revisit equation Eq. 6 that imposes the need of this operation. As mentioned, $R^{(Qt)}_{j\to i}(\mathbf{s}_j)$, depends on all variables of factor j. Yet, for the maximization procedure, variables $\mathbf{x}_i$ are bounded each time, so we need to maximize for all the remaining ones, i.e., we need the value: $\max_{\mathbf{s}_j \setminus \mathbf{x}_i} \left[R^{(Qt)}_{j\to i}(\{\mathbf{s}_j \setminus \mathbf{x}_i\}|\mathbf{x}_i) \right]$ given the provided bounded values of $\mathbf{x}_i$. Thus, to attain a Quadtree function of $r^{(Qt)}_{j\to i}(\mathbf{x}_i)$ (see Eq. 6), we again construct it via Alg. 1, and set:

$$f(\mathbf{x}_i) \quad = \quad \max_{\mathbf{s}_j \setminus \mathbf{x}_i} \left[R^{(Qt)}_{j\to i}\Big(\{\mathbf{s}_j \setminus \mathbf{x}_i\}|\mathbf{x}_i\Big) \right] \tag{9}$$

where for every function call required within the algorithm (i.e., given a set of values for $\mathbf{x}_i$ corresponding to a discrete point and a tree node), we construct in turn a Quadtree for variables $\mathbf{s}_j \setminus \mathbf{x}_i$ to obtain the max value of the term $R^{(Qt)}_{j\to i}\Big(\{\mathbf{s}_j \setminus \mathbf{x}_i\}|\mathbf{x}_i\Big)$. While constructing the tree, we keep track of the node with the maximum value observed, therefore we have direct access to the requested value as soon as the tree construction process is terminated.

3.3 Normalization Constant Estimation

We have a Quadtree $q^{(Qt)}_{i\to j}(\mathbf{x}_i)$ that approximates the real function $q_{i\to j} : \mathbb{R}^n \to \mathbb{R}$. A normalization constant is important when the Factor Graph contains cycles (as in our domain of interest). As [Farinelli *et al.*, 2008] suggests, a_{ij} is set so as to satisfy $\sum_{\mathbf{x}_i} q^{(Qt)}_{i\to j}(\mathbf{x}_i) = 0$.

Starting from Eq. 7, and using a Quadtree approximation, we define an auxiliary function for convenience:

$$q^{(Qt)}_{i\to j,n}(\mathbf{x}_i) \quad = \quad \sum_{k\in M_i \setminus j} r^{(Qt)}_{k\to i}(\mathbf{x}_i) \tag{10}$$

where $q^{(Qt)}_{i\to j,n}(\mathbf{x}_i)$ is simply $q^{(Qt)}_{i\to j}(\mathbf{x}_i)$ but without the normalization constant. Therefore, the normalization constant should satisfy:

$$\sum_{\mathbf{x}_i} q^{(Qt)}_{i\to j}(\mathbf{x}_i) \quad = \quad 0 \tag{11}$$

where:

$$\sum_{\mathbf{x}_i} [a_{ij} + q^{(Qt)}_{i\to j,n}(\mathbf{x}_i)] \quad = \quad 0 \tag{12}$$

In the standard/discrete Max-Sum algorithm we would normally sum over all the possible combinations and solve for a_{ij}, but this is not straightforward in our case. If we attempted an analytical solution, we would probably replace the sum with an integral [Kschischang *et al.*, 2001]. Here, we should properly define this process for a Quadtree function. We remind the reader that we have a tree structure that approximates the actual function for a unit hybercube space, and each leaf node of the tree represents the value of the function for a specific subregion. Consequently, this summation could be approximated by summing over all leaf node values and multiply them with the corresponding area.

Thus, we define the set of all leaf nodes L_q for a Quadtree function $q^{(Qt)}_{i\to j}(\mathbf{x}_i)$, and the two functions $s(l), v(l)$ that return the surface area and the function value of a leaf node l respectively.

Therefore, we define:

$$\sum_{\mathbf{x}_i} q_{i\to j}^{(Qt)}(\mathbf{x}_i) \triangleq \sum_{\forall l\in L_q} s(l)\cdot v(l) \quad (13)$$

where $\sum_{\forall l\in L_q} s(l) = 1$, since the total surface is a unit hypercube. Thus, in the context of Max-Sum, we can employ the Quadtree of $q_{i\to j,n}^{(Qt)}(\mathbf{x}_i)$, without normalization, and then, using the definition above we can obtain a value for a_{ij}. In the equation below, L_q is the set of all leaf nodes for $q_{i\to j,n}^{(Qt)}(\mathbf{x}_i)$.

$$\begin{aligned}
\sum_{x_{i,1},x_{i,2}} [a_{ij} + q_{i\to j,n}^{(Qt)}(x_{i,1},x_{i,2})] &= 0\\
\sum_{\forall l\in L_q} s(l)[a_{ij} + v(l)] &= 0\\
a_{ij}\sum_{\forall l\in L_q} s(l) + \sum_{\forall l\in L_q} s(l)v(l) &= 0\\
a_{ij} + \sum_{\forall l\in L_q} s(l)v(l) &= 0
\end{aligned}$$

which yields:

$$a_{ij} = -\sum_{\forall l\in L_q} s(l)v(l) \quad (14)$$

After the Quadtree function is constructed for $q_{i\to j,n}^{(Qt)}(\mathbf{x}_i)$, then the normalization constant is calculated and incorporated accordingly.

3.4 Variable Configuration

Finally, after the optimization process is carried out, each agent i needs to select the configuration of $\mathbf{x}_i$ that maximizes:

$$\mathbf{x}_i^* = \arg\max_{\mathbf{x}_i} \sum_{j\in M_i} r_{j\to i}^{(Qt)}(\mathbf{x}_i) \quad (15)$$

We obtain the sum of $\sum_{j\in M_i} r_{j\to i}^{(Qt)}(\mathbf{x}_i)$ by simply constructing a Quadtree function, meaning that we gain direct access to the maximization node, and consequently to the subregion that has the maximum value observed. Finally, we assign the center point of the appointed region to $\mathbf{x}_i^*$.

4 Experimental Evaluation

In this section we present our lane-free traffic application domain, and our experimental evaluation results.[2]

We consider a highway populated with many vehicles entering from an origin point, where each vehicle possesses a desired speed parameter v_d, chosen at random within a specified speed region $[v_{d,low}, v_{d,high}]$. The induced speed deviations among nearby vehicles, along with heavier traffic in a lane-free environment, constitute a complex multi-agent environment appropriate for the application of our method.

[2]The source code for Max-Sum-Quad implementation can be found in: https://bit.ly/3MLRVLk

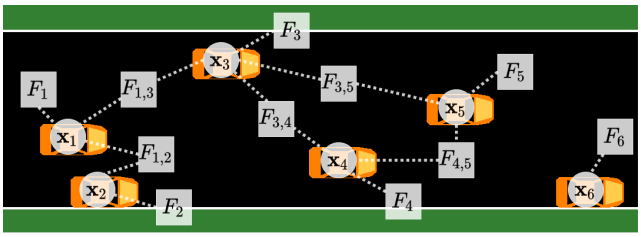

Figure 4: An instance of a Factor Graph on the lane-free traffic environment.

Here we follow the methodology of [Troullinos *et al.*, 2021a], since, to the best of our knowledge, this is the only work tackling Continuous DCOPs in the lane-free traffic domain—albeit using discretization. Specifically, we model each vehicle as an agent that controls two acceleration parameters, one longitudinal and one lateral. [Troullinos *et al.*, 2021a] define a set of 5 actions per agent, discretizing the joint action space of the two acceleration, i.e., converting the actual 2-D action space into a vector of 5 different combinations for agents. While this coarse-grained discretization may be adequate up to a certain point, heavier traffic (induced either by larger traffic flows, or larger vehicles) may require a finer discretization, i.e., better resolution with respect to the available actions of the agents. In this case, there is the issue of properly pre-defining the points in the 2-dimensional control space, as vehicles may face various traffic conditions, and therefore a predefined set of points could be potentially large, and adjusted through a meticulous experimental process. Max-Sum-Quad lifts the need of pre-selecting this set of points.

4.1 Formulation of the Problem

For a given instance of the traffic environment, we define a Factor Graph representation of vehicles, as illustrated in Fig. 4. The graph contains two types of factors. The first factor type $F_s(a_x, a_y)$ is connected with a single vehicle (the two variables of each vehicle) and accounts for the vehicle's desired speed, while the second one $F_p(a_{x,1}, a_{y,1}, a_{x,2}, a_{y,2})$ is pairwise, in the sense that it connects two vehicles at a time, i.e., 4 variables. This type of factor aims to properly adjust the vehicles' movement so as to avoid collisions. The factors' parameters reflect the vehicles' control variables, i.e., the longitudinal and lateral acceleration a_x, a_y respectively (in m/s^2). The form of the first type of factor is:

$$F_s(a_x, a_y) = w_{s,1}\cdot r_x(v_x', v_d) + w_{s,2}\cdot r_y(v_y') \quad (16)$$

where:

$$r_x(v_x, v_d) = \begin{cases} \frac{v_x}{v_d}, & (v_x < v_d)\\ \frac{2\cdot v_d - v_x}{v_d}, & else \end{cases} \quad (17)$$

and:

$$r_y(v_y) = \begin{cases} \frac{v_y}{v_{y,max}}, & (v_y < 0)\\ \frac{-v_y}{v_{y,max}}, & else \end{cases} \quad (18)$$

Essentially, F_s is responsible for 'motivating' the vehicle to follow its desired speed v_d (due to r_x), and also to have a lateral speed close to 0 (due to r_y). Notation $'$ in Eq.16 denotes

a_c	$v_{length} + 0.8m$	ϵ_q	0.05
b_c	$v_{width} + 0.8m$	$v_{length,1}$	$3.2m$
τ_x, τ_y	$0.3s, 0.45s$	$v_{width,1}$	$1.6m$
p_x, p_y, p_t	$2, 4, 2$	$v_{length,2}$	$3.9m$
$w_{s,1}$	1	$v_{width,2}$	$1.8m$
$w_{s,2}$	0.1	$road_{length}$	$2km$
$minD$	1	$road_{width}$	$10.2m$
$maxD$	2	sim. time	$1hr$
$a_{x,max}$ (QT)	3.5 $\frac{m}{s^2}$	time-interval	$0.25s$
$a_{x,min}$ (QT)	-3.5 $\frac{m}{s^2}$	v_d range	$[25, 35]\frac{m}{s}$
$a_{y,max}$ (QT)	1.5 $\frac{m}{s^2}$	v_{init}	$25\frac{m}{s}$

Table 1: Factor Graph, Quadtree & simulation parameters

that these variables have the value as a result of the input variables, e.g., v'_x is the longitudinal speed in the next time-step based on a_x. Moving on to the pairwise factor F_p:

$$F_p(a_{x,1}, a_{y,1}, a_{x,2}, a_{y,2}) = E_c(dx', dy') + E_b(dx', dy', dv'_x, dv'_y) \quad (19)$$

where it depends on the accelerations of two connected vehicles $(1, 2)$, therefore it has 4 input parameters. The form of E_c and E_b is adopted from [Troullinos *et al.*, 2021a], but we included some minor adjustments relevant to E_b, which perform better specifically in denser traffic. Both E_c and E_b are ellipsoid fields, with the form:

$$E(dx, dy) = \frac{m}{\left(\left(\frac{|dx|}{a}\right)^{p_x} + \left(\frac{|dy|}{b}\right)^{p_y} + 1\right)^{p_t}} \quad (20)$$

where parameters a, b dictate the ellipses' stretch in each dimension. Specifically for E_b field, that captures a broader region, we provide with some small adjustments regarding the relevant parameters a_b, b_b dictating the ellipses' reach:

$$a_b = \frac{(dv'_x)^2}{2 \cdot |u_x^{d,max}|} + \tau_x \cdot \frac{v_{x,1} + v_{x,2}}{2} + a_c \quad (21)$$

$$b_b = \frac{(dv'_y)^2}{2 \cdot |u_y^{d,max}|} + \tau_y \cdot (|v_{y,1}| + |v_{y,2}|) + b_c \quad (22)$$

where the speeds variables $v_{x,1}, v_{x,2}, v_{y,1}, v_{y,2}$ for the terms relevant to τ_x, τ_y are not the resulting ones based on the acceleration input, but rather the current ones, hence the lack of notation ($'$). This choice appears to diminish instabilities caused in heavier traffic scenarios. We refer the interested reader to [Troullinos *et al.*, 2021a] for more details regarding the ellipsoid functions used.

4.2 Implementation Details

As mentioned, we consider a unit hypercube space for Quadtrees, meaning that all control variables should reside within the range $[0, 1]$. Therefore, we perform a normalization, so that 0 corresponds to the minimum acceleration ($a_{x \vee y,min}$) and 1 to the maximum value ($a_{x \vee y,max}$), for both acceleration axes. We have $a_{y,min} = -a_{y,max}$ in all cases, since it is sensible that the available values for acceleration towards the left ($a_y > 0$) or right ($a_y < 0$) should be symmetric. Note that due to the Quadtree function approximation, the control values that can be obtained always correspond to the center point of a maximizing subregion of the whole space. Essentially, this means that the values $a_{x,max}, a_{x,min}, a_{y,max}$ are not attainable, and the actual maximum and minimum control values are determined by the maximum tree depth ($maxD$). For example, in the longitudinal axis, we have $a_x : [a_{x,min}, a_{x,max}] = [-3.5, 3.5]$, and a maximum tree depth of $maxD = 2$. Consequently, the length for each subregion at depth 2 is $\frac{a_{x,max} - a_{x,min}}{2^{maxD}} = 1.75$, and thus the actual minimum and maximum values are within the range $a_x : [-3.5 + \frac{1.75}{2}, 3.5 - \frac{1.75}{2}] = [-2.625, 2.625]$, since the corresponding center points lie in the middle of each subregion. The same principle applies for the lateral axis as well.

An important aspect is the fact that the traffic environment is dynamic, hence we need to revise the control variables in every instance, meaning that the Factor Graph is updated in every time-step, and then Max-Sum(-Quad) performs one iteration so that each agent can obtain a new solution. In this manner, the computed Quadtree message functions are retained after each time-step (provided that the correlated agents preserve their existing connection), so that we do not start computing a new solution from scratch every time.

Parameter choices regarding the traffic environment, tuning of the algorithm, the factors, and Quadtrees can be found in Table 1.

4.3 Experiments and Results

A primary objective in our experiments is to demonstrate the efficiency of our approach for intensified traffic conditions, contrasting its performance with that of [Troullinos *et al.*, 2021a], and to showcase the dynamic discretization of Quadtrees experimentally. Experimental evaluation is conducted using the simulation tool of [Troullinos *et al.*, 2021b], which is an extension of the SUMO microscopic simulation platform [Lopez *et al.*, 2018] for lane-free traffic environments. To this end, we set the factors' safety parameters so that vehicles employ a much more aggressive driving style (smaller reaction times and safety distances); and by exam-

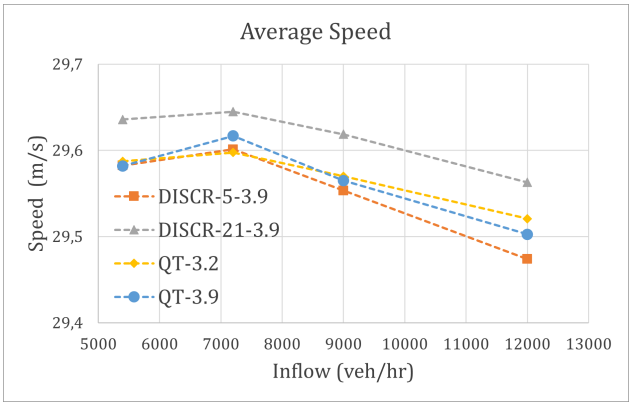

Figure 5: Average speed of vehicles for each case.

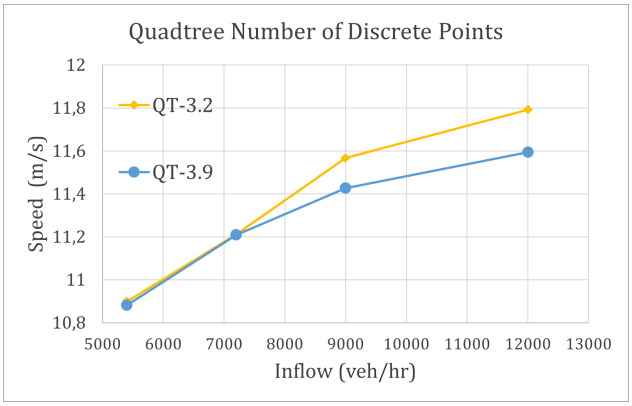

Figure 6: Average number of points selected by Quadtrees.

ining two different configurations of vehicle dimensions (Table 1) as they contribute to traffic density in lane-free traffic.

Experimental results are provided in Figs. 5, 6, where we compare each method and vehicle configuration, for various inflow rates $[5400 - 12000]\frac{veh}{hr}$ at the origin point of the highway. In total, we compare four different cases, as evident in the relevant figures, where QT-3.2 & QT-3.9 correspond to the Max-Sum-Quad approach, with the relevant parameters tuning provided in Table 1. The numbers "3.2" & "3.9" simply indicate the associated vehicles' length dimensions. Moreover, DISCR-5-3.9 & DISCR-21-3.9 adopt the standard Max-Sum algorithm, with a discretization of the 2-dimensional acceleration space of 5 and 21 different points respectively. DISCR-5-3.9 corresponds to the equivalent of a fully partitioned Quadtree with maximum depth of 1, i.e., 1 point at the center and the 4 center points of the respective subregions, and DISCR-21-3.9 is the discretization of a fully partitioned Quadtree at depth of 2, containing all points of DISCR-5-3.9, along with 16 more points at tree depth of 2.

As evident in Fig. 5, the induced aggressive behaviour of vehicles, stemming from the parameter choices regarding safety distances and reaction time, substantially improves the performance of all examined scenarios with respect to the formulation of [Troullinos *et al.*, 2021a] with safer parameter choices, where average speeds are ranging from $28.61\frac{m}{s}$ to $27.64\frac{m}{s}$ (and only up to $9000\frac{veh}{hr}$). It is noticeable also that the size of the vehicles affects performance, especially as the inflow increases. Now, in order to properly assess the performance of Max-Sum-Quad, we focus on more intense conditions, with larger vehicle dimensions. DISCR-5-3.9, due to its coarse discretization, is not collision-free at higher inflow rates (we observe 3 collision occurrences at $9000\frac{veh}{hr}$), meaning that the selection of 5 points is not adequate. DISCR-21-3.9 manages to improve upon DISCR-5-3.9 due to the finer discretization, and without any observed collisions. However, our approach manages to provide an intermediate solution, but now with a "dynamic discretization" procedure, one that *does not* require the user to discretize the domain beforehand. With Max-Sum-Quad, we did not observe *any collisions* in our experiments.

In Fig. 6 we showcase the associated average number of points selected by Quadtrees in this domain, and we observe that in the most demanding scenario, the constructed Quadtrees on average result in 11.6 points—i.e., Max-Sum-Quad achieves similar performance with 56% of the standard Max-Sum Algorithm with 21 points. This also showcases that the dynamic discretization process automatically adapts to more intense traffic conditions, since we have a noticeable increase on the size of Quadtree nodes on average (cf. Fig. 6). Evidently, smaller vehicles require more refined partitioning on average, in more intense traffic conditions. This result is non-intuitive, in the sense that the scenario with larger vehicles is more demanding in terms of collision avoidance. However, this might be an indication that the resolution level of Quadtrees is affected by the fact that for the same inflow rate, the vehicles operate with slightly higher speed (on average).

Summing up, the imposed challenging traffic scenarios, populated as they are with vehicles driving aggressively, necessitate a more refined selection of available actions to handle the more demanding cases that now frequently arise—e.g., when many vehicles are nearby and some attempt overtaking. Our results show that our approach offers vehicles a dynamic selection of available actions, affected by messages in the form of Quadtrees. Evidently, Max-Sum-Quad adapts to the problem at hand without requiring a manual discretization process by the user, and it also adjusts automatically the level of discretization for each message in an online fashion.

5 Conclusions and Future Work

In this work, we presented an extension of the well-known Max-Sum algorithm for Continuous DCOPs, with application in the novel lane-free autonomous driving domain. Our results confirm that embedding Quadtrees in Max-Sum, render the latter appropriate for demanding Continuous DCOPs.

In future work, we plan to explore different domains of application, such as Random Graphs which are commonly used in related work (see Sec. 2.2), and compare the efficiency of the proposed method with other approaches in the literature. Another direction is to establish theoretical guarantees for Max-Sum-Quad and consider alternative function approximation techniques.

Acknowledgements

The research leading to these results has received funding from the European Research Council under the European Union's Horizon 2020 Research and Innovation programme/ ERC Grant Agreement n. [833915], project TrafficFluid.

References

[Choudhury *et al.*, 2020] M. Choudhury, S. Mahmud, and M. M. Khan. A particle swarm based algorithm for functional distributed constraint optimization problems. *Proc. of the AAAI Conference on Artificial Intelligence*, 34:7111–7118, 2020.

[Farinelli *et al.*, 2008] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 639–646, 2008.

[Finkel and Bentley, 1974] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[Fransman *et al.*, 2019] J. Fransman, J. Sijs, H. Dol, E. Theunissen, and B. De Schutter. Bayesian-dpop for continuous distributed constraint optimization problems. In *Proc. of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, page 1961–1963, 2019.

[Guestrin *et al.*, 2002] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, 2002.

[Har-Peled, 2011] S. Har-Peled. *Geometric approximation algorithms*. American Mathematical Soc., 2011.

[Hoang *et al.*, 2020] K. D. Hoang, W. Yeoh, M. Yokoo, and Z. Rabinovich. New algorithms for continuous distributed constraint optimization problems. In *Proc. of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, page 502–510, 2020.

[Kok and Vlassis, 2006] J. R. Kok and N. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7(65):1789–1828, 2006.

[Kschischang *et al.*, 2001] F. R. Kschischang, B. J. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. on information theory*, 47(2):498–519, 2001.

[Kuyer *et al.*, 2008] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Machine Learning and Knowledge Discovery in Databases*, pages 656–671, 2008.

[Lopez *et al.*, 2018] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *21st IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[Malekzadeh *et al.*, 2021] M. Malekzadeh, I. Papamichail, M. Papageorgiou, and K. Bogenberger. Optimal internal boundary control of lane-free automated vehicle traffic. *Transportation Research Part C: Emerging Technologies*, 126:103060, 2021.

[Modi *et al.*, 2003] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proc. of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, page 161–168, 2003.

[Murphy *et al.*, 1999] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proc. of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, page 467–475, San Francisco, CA, USA, 1999.

[Papageorgiou *et al.*, 2021] M. Papageorgiou, K. Mountakis, I. Karafyllis, I. Papamichail, and Y. Wang. Lane-free artificial-fluid concept for vehicular traffic. *Proc. of the IEEE*, 109(2):114–121, 2021.

[Pendleton *et al.*, 2017] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1), 2017.

[Sarker *et al.*, 2021] A. Sarker, M. Choudhury, and M. M. Khan. A local search based approach to solve continuous dcops. In *Proc. of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, page 1127–1135, 2021.

[Shusterman and Feder, 1994] E. Shusterman and M. Feder. Image compression via improved quadtree decomposition algorithms. *IEEE Trans. on Image Processing*, 3(2):207–215, 1994.

[Stranders *et al.*, 2009] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised coordination of continuously valued control parameters using the max-sum algorithm. In *Proc. of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) - Volume 1*, page 601–608, 2009.

[Troullinos *et al.*, 2021a] D. Troullinos, G. Chalkiadakis, I. Papamichail, and M. Papageorgiou. Collaborative multiagent decision making for lane-free autonomous driving. In *Proc. of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1335–1343, 2021.

[Troullinos *et al.*, 2021b] Dimitrios Troullinos, Georgios Chalkiadakis, Diamantis Manolis, Ioannis Papamichail, and Markos Papageorgiou. Lane-free microscopic simulation for connected and automated vehicles. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3292–3299, 2021.

[van der Pol and Oliehoek, 2016] E. van der Pol and F. A. Oliehoek. Coordinated deep reinforcement learners for traffic light control. In *NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems*, 2016.

[Voice *et al.*, 2010] T. Voice, R. Stranders, A. Rogers, and N. R. Jennings. A hybrid continuous max-sum algorithm for decentralised coordination. In *Proc. of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, page 61–66, 2010.

[Yanumula *et al.*, 2021] Venkata K. Yanumula, P. Typaldos, D. Troullinos, M. Malekzadeh, I. Papamichail, and M. Papageorgiou. Optimal path planning for connected and automated vehicles in lane-free traffic. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3545–3552, 2021.

[Yu *et al.*, 2020] C. Yu, X. Wang, X. Xu, M. Zhang, H. Ge, J. Ren, L. Sun, B. Chen, and G. Tan. Distributed multiagent coordinated learning for autonomous driving in highways based on dynamic coordination graphs. *IEEE Trans. on Intelligent Transportation Systems*, 21(2):735–748, 2020.