CDC 2007, To appear
New Orleans, LA

Network abstract linear programming with application to minimum-time formation control

Giuseppe Notarstefano Francesco Bullo

***Abstract*— We identify a novel class of distributed optimization problems, namely a networked version of abstract linear programming. For such problems we propose distributed algorithms for networks with various connectivity and/or memory constraints. Finally, we show how various minimum-time formation control problems can be tackled through appropriate geometric examples of abstract linear programs.**

I. Introduction

This paper focuses on a class of distributed computing problems and on its applications to formation control problems for mobile robotic networks. The objective of formation control problems is to move the robots in the network to relative positions with specific properties. To do so, we study abstract linear programming, that is, a generalized version of linear programming that was introduced by Matoušek, Sharir and Welzl in [1] and extended by Gärtner in [2]. Abstract linear programming is applicable also to some geometric optimization problems, such as the minimum enclosing ball, the minimum enclosing stripe and the minimum enclosing annulus. These geometric optimization problems are relevant in the design of efficient robotic algorithms for minimum-time formation control problems.

Linear programming and its generalizations have received widespread attention in the literature. The following references are most relevant in our treatment. The earliest (deterministic) algorithm that solves a linear program in a fixed number of variables subject to n linear inequalities in time $O(n)$ is given in [3]. An efficient randomized incremental algorithm for linear programming is proposed in [1], where a linear program in d variables subject to n linear inequalities is solved in expected time $O(d^2n + e^{O(\sqrt{d \log d})})$; the expectation is taken over the internal randomizations executed by the algorithm. An elegant survey on randomized methods in linear programming is [4]. The survey [5] discusses the application of abstract linear programming to a number of geometric optimization problems. Regarding parallel computation approaches to linear programming, we only note that linear programs with n linear inequalities can be solved [6] by n parallel processors in time $O((\log \log(n))^d)$. The approach in [6] and the ones in the references therein are, however, limited to parallel random-access machines (usually denoted PRAM), where a shared memory is readable and writable to all processors. In this paper, we focus on networks described by arbitrary graphs and on robotic networks described by geometric graphs.

The literature on formation control for robotic networks has been growing recently. An early reference on distributed algorithms for the formation of geometric patterns is [7]. Regarding the rendezvous problem, that is, the problem of gathering the robots at a common location, an early reference is [8]. A control-Lyapunov function approach to formation control is discussed in [9]. An input-to-state stability approach is taken in [10].

The contributions of this paper are three-fold. First, we identify a class of distributed optimization problems that appears to be novel and of intrinsic interest. Second, we propose a novel simple algorithmic methodology to solve these problems in networks with various connectivity and/or memory constraints. Specifically, we propose three algorithms, prove their correctness and establish halting conditions. Finally, we illustrate how these distributed computation problems are relevant in minimum-time formation control problems, such as the rendezvous problem and the problems of line and circle formations. Specifically, we design some joint communication and motion coordinations schemes in which robots move towards the estimated final shape while the final shape is being computed. The proposed laws extend to a general setting the rendezvous control law proposed in [11].

The paper is organized as follows. Section II introduces abstract linear programs. Section III introduces network models. Section IV contains the definition of network abstract linear programs and the proposed distributed algorithms. Section V shows the relevance of the proposed distributed computing algorithms in the context of formation control.

Notation: We let $\mathbb{N}$, $\mathbb{N}_0$, and $\mathbb{R}_+$ denote the natural numbers, the non-negative integer numbers, and the positive real numbers, respectively. For $r \in \mathbb{R}_+$ and $p \in \mathbb{R}^d$, we let $B(p, r)$ denote the closed ball centered at p with radius r, i.e., $B(p, r) = \{q \in \mathbb{R}^d \mid \|p - q\|_2 \leq r\}$. For $f, g : \mathbb{N} \to \mathbb{R}$, we say that $f \in O(g)$ if there exist $n_0 \in \mathbb{N}$ and $k \in \mathbb{R}_+$ such that $|f(n)| \leq k|g(n)|$ for all $n \geq n_0$.

II. Abstract linear programming

In this section we present an abstract framework that captures a wide class of optimization problems including linear programming and various geometric optimization problems. These problems are known as *abstract linear programs* (or *LP-type problems*). They can be considered a generalization of linear programming in the sense that they share some important properties. A comprehensive analysis of these problems may be found for example in [5].

This material is based upon work supported in part by ARO MURI Award W911NF-05-1-0219 and ONR Award N00014-07-1-0721. This work was completed as part of G. Notarstefano's PhD program at the University of Padova.

Giuseppe Notarstefano is with the Dipartimento di Ingegneria dell'Innovazione, Università del Salento, Via per Monteroni, 73100 Lecce, Italy, giuseppe.notarstefano@unile.it

Francesco Bullo is with the Center for Control, Dynamical Systems and Computation, University of California at Santa Barbara, Santa Barbara, CA 93106, bullo@engineering.ucsb.edu

A. Abstract framework

We consider optimization problems specified by a pair (H, ω), where H is a finite set, and $\omega : 2^H \to \Omega$ is a function with values in a linearly ordered set $(\Omega, \leq)$; we assume that Ω has a minimum value $-\infty$. The elements of H are called *constraints*, and for $G \subset H$, $\omega(G)$ is called the *value* of G. Intuitively, $\omega(G)$ is the smallest value attainable by a certain objective function while satisfying the constraints of G. An optimization problem of this sort is called *abstract linear program* if the following two axioms are satisfied:

(i) *Monotonicity*: if $F \subset G \subset H$, then $\omega(F) \leq \omega(G)$;
(ii) *Locality*: if $F \subset G \subset H$ with $-\infty < \omega(F) = \omega(G)$, then, for all $h \in H$,

$$\omega(G) < \omega(G \cup \{h\}) \implies w(F) < w(F \cup \{h\}).$$

A set $B \subset H$ is *minimal* if $\omega(B) > \omega(B')$ for all proper subsets B' of B. A minimal set B with $-\infty < \omega(B)$ is a *basis*. Given $G \subset H$, a *basis of* G is a minimal subset $B \subset G$, such that $-\infty < \omega(B) = \omega(G)$. A constraint h is said to be *violated* by G, if $\omega(G) < \omega(G \cup \{h\})$.

The *solution* of an abstract linear program (H, ω) is a minimal set $B_H \subset H$ with the property that $\omega(B_H) = \omega(H)$. The *combinatorial dimension* δ of (H, ω) is the maximum cardinality of any basis. Finally, an abstract linear program is called *basis regular* if, for any basis with $\mathrm{card}(B) = \delta$ and any constraint $h \in H$, every basis of $B \cup \{h\}$ has the same cardinality of B. We now define two important primitive operations that are useful to solve abstract linear programs.

(i) *Violation test*: given a constraint h and a basis B, it tests whether h is violated by B; we denote this primitive by $\texttt{Viol}(B, h)$;
(ii) *Basis computation*: given a constraint h and a basis B, it computes a basis of $B \cup \{h\}$; we denote this primitive by $\texttt{Basis}(B, h)$.

Remark 2.1 (Examples of abstract linear programs): We present three useful geometric examples; see Figure 1.

(i) *Smallest enclosing ball:* Given n points in $\mathbb{R}^d$, compute the center and radius of the ball of smallest volume containing all the points. This problem has combinatorial dimension $d + 1$.
(ii) *Smallest enclosing stripe:* Given n points in $\mathbb{R}^2$ in generic positions, compute the center and the width of the stripe of smallest width containing all the points. This problem has combinatorial dimension 5.
(iii) *Smallest enclosing annulus:* Given n points in $\mathbb{R}^2$, compute the center and the two radiuses of the annulus of smallest area containing all the points. This problem has combinatorial dimension 4.

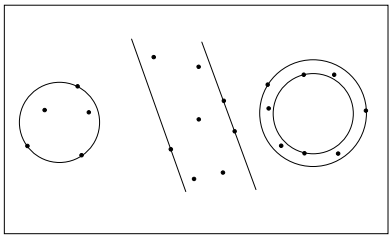

Fig. 1. Smallest enclosing ball, stripe and annulus

More examples are discussed in [1], [2], [4], [5]. □

B. Randomized sub-exponential algorithm

A randomized algorithm for solving abstract linear programs has been proposed in [1]. Such algorithm has linear expected running time in terms of the number of constraints, whenever the combinatorial dimension δ is fixed, and subexponential in δ. The algorithm, called $\texttt{SUBEX\_lp}$, has a recursive structure and is based on the two primitives introduced above, i.e., the violation test and the basis computation primitives. For simplicity, we assume here that such primitives may be implemented in constant time, independent of the number of constraints. Given a set of constraints G and a candidate basis $C \subset G$, the algorithm is as follows.

function $\texttt{SUBEX\_lp}(G, C)$
  if $G = C$, **then** return C
  else
    choose a random $h \in G \setminus C$
    $B := \texttt{SUBEX\_lp}(G \setminus \{h\}, C)$
    if $\texttt{Viol}(B, h)$, i.e., h is violated by B,
      return $\texttt{SUBEX\_lp}(G, \texttt{Basis}(B, h))$
    else return B
    end if
  end if

For the abstract linear program (H, ω), the routine is invoked with $\texttt{SUBEX\_lp}(H, B)$, given any initial candidate basis B.

In [1] the expected completion time for the $\texttt{SUBEX\_lp}$ algorithm in conjunction with Clarkson's algorithms was shown to be in $O(d^2 n + e^{O(\sqrt{d \log d})})$ for basis regular abstract linear programs. In [4] the result was extended to problems that are not basis regular.

III. Network models

Following [12], we define a synchronous network system as a "collection of *computing elements* located at nodes of a directed network graph." These computing elements are sometimes called *processors*.

A. Digraphs and connectivity

We let $I = \{1, \ldots, n\}$ and let $\mathcal{G} = (I, E)$ denote a directed graph, where I is the set of nodes and $E \subset I \times I$ is the set of edges. For each node i of $\mathcal{G}$, the number of edges going out from (coming into) node i is called *out-degree* (*in-degree*) and is denoted $\texttt{outdeg}[i]$ ($\texttt{indeg}[i]$). The set of outgoing (incoming) neighbors of node i are the set of nodes to (from) which there are edges from (to) i. They are denoted $\mathcal{N}_O(i)$ and $\mathcal{N}_I(i)$, respectively. A directed graph is called *strongly connected* if, for every pair of nodes $(i, j) \in I \times I$, there exists a path of directed edges that goes from i to j. In a strongly connected digraph, the minimum number of edges between node i and j is called the *distance from* i *to* j and is denoted $\mathrm{dist}(i, j)$. The maximum $\mathrm{dist}(i, j)$ taken over all pairs (i, j) is the *diameter* and is denoted $\mathrm{diam}(\mathcal{G})$. Finally, we consider time-dependent directed graphs of the form $t \mapsto \mathcal{G}(t) = (I, E(t))$. The time-dependent directed graph $\mathcal{G}$ is *jointly strongly connected* if, for every $t \in \mathbb{N}_0$,

$$\cup_{\tau=t}^{+\infty} \mathcal{G}(\tau) \text{ is strongly connected.}$$

Moreover, the time-dependent directed graph $\mathcal{G}$ is *uniformly strongly connected* if, there exists $S > 0$ s.t. for every $t \in \mathbb{N}_0$

$$\cup_{\tau=t}^{t+S} \mathcal{G}(\tau) \text{ is strongly connected.}$$

B. Synchronous networks and distributed algorithms

Strictly speaking, a *synchronous network* is a directed graph $\mathcal{G} = (I, E_{\text{cmm}})$ where the set $I = \{1, \dots, n\}$ is the set of *identifiers* of the computing elements, and the time-dependent map $E_{\text{cmm}} : \mathbb{N}_0 \to 2^{I \times I}$ is the *communication edge map* with the following property: an edge (i, j) belongs to $E_{\text{cmm}}(t)$ if and only if processor i can communicate to processor j at time t.

Definition 3.1 (Distributed algorithm): Let $\mathcal{G} = (I, E_{\text{cmm}})$ be a synchronous network. A distributed algorithm consists of the sets

- W, set of "logical" states $w^{[i]}$, for all $i \in I$;
- $W_0 \subset W$, subset of allowable initial values;
- M, message alphabet, including the `null` symbol;

and the maps

- msg : $W \times I \to M$, message-generation function;
- stf : $W \times M^n \to W$, state-transition function. □

Execution of the network begins with all processors in their start states and all channels empty. Then the processors repeatedly perform the following two actions. First, the ith processor sends to each of its outgoing neighbors in the communication graph a message (possibly the `null` message) computed by applying the message-generation function to the current value of $w^{[i]}$. After a negligible period of time, the ith processor computes the new value of its logical variables $w^{[i]}$ by applying the state-transition function to the current value of $w^{[i]}$, and to the incoming messages (present in each communication edge). The combination of the two actions is called a *communication round* or simply a round.

In this execution scheme we have assumed that each processor executes all the calculations in one round. If it is not possible to upper bound the execution-time of the algorithm, we may consider a slightly different network model that allows the state-transition function to be executed in multiple rounds. When this happens, the message is generated by using the logical state at the previous round.

The last aspect to consider is the *algorithm halting*, that is a situation such that the network (and therefore each processor) is in a idle mode. Such status can be used to indicate the achievement of a prescribed task. Formally we say that a distributed algorithm is in *halting status* if the logical state is a fixed point for the state-transition function (that becomes a self-loop) and no message (or equivalently the `null` message) is generated at each node.

C. Robotic networks and control and communication laws

We report here the notion of robotic network adopting a simplified version of the formal model introduced in [13]. A *robotic network* is a tuple $(I, \mathcal{A}, E_{\text{cmm}})$, where $I = \{1, \dots, n\}$ is the set of identifiers; $\mathcal{A} = \{A^{[i]}\}_{i \in I} = \{(X, U, X_0, f)\}_{i \in I}$ is called the *set of physical agents* and is a set of control systems consisting of a differentiable manifold X (state space), a compact subset U of $\mathbb{R}^m$ (input space), a subset X_0 of X (set of allowable initial states) and a (sufficiently smooth) map $f : X \times U \to X$ describing the dynamics of ith agent; $E_{\text{cmm}} : X^n \to 2^{I \times I}$ is the *communication edge map*. Here the communication edges depend upon the position of the robots, as opposed to time.

The robotic network evolves according to a discrete-time communication and motion model.

Definition 3.2 (Control and communication law): Let $\mathcal{S}$ be a robotic network. A *control and communication law* $\mathcal{CC}$ for $\mathcal{S}$ consists of the sets M (message alphabet), W (set of logical states) and $W_0 \subseteq W$ (allowable initial values), as defined in Definition 3.1, and of the maps:

(i) msg : $X \times W \times I \to M$, called *message-generation function*;
(ii) stf : $W \times M^n \to W$, called *state-transition function*;
(iii) ctl : $X \times W \times M^n \to U$, called *control function*. □

Roughly speaking, this definition has the following meaning: for all $i \in I$, to the ith physical agent corresponds a processor, labeled i, that performs the following actions. First, at each communication round the ith processor sends to each of its outgoing neighbors in the communication graph a message (possibly the `null` message) computed by applying the message-generation function to the current values of $x^{[i]}$ and $w^{[i]}$. After a negligible period of time, the ith processor resets the value of its logical state $w^{[i]}$ by applying the state-transition function to the current value of $w^{[i]}$, and to the messages received at time t. Between communication instants, the motion of the ith agent is determined by applying the control function to the current value of $x^{[i]}$, and the current value of $w^{[i]}$. This idea is formalized as follows.

Definition 3.3 (Evolution of a robotic network): Let $\mathcal{S}$ be a robotic network and $\mathcal{CC}$ be a control and communication law for $\mathcal{S}$. The *evolution* of $(\mathcal{S}, \mathcal{CC})$ from initial conditions $x_0^{[i]} \in X_0$ and $w_0^{[i]} \in W_0$, $i \in I$, is the set of curves $x^{[i]} : \mathbb{N}_0 \to X$ and $w^{[i]} : \mathbb{N}_0 \to W$, $i \in I$, satisfying

$$x^{[i]}(t+1) = f\big(x^{[i]}(t), \text{ctl}(x^{[i]}(t), w^{[i]}(t+1), y^{[i]}(t))\big),$$

where, for $i \in I$,

$$w^{[i]}(t+1) = \text{stf}(w^{[i]}(t), y^{[i]}(t)),$$

with the conventions that $x^{[i]}(0) = x_0^{[i]}$ and $w^{[i]}(0) = w_0^{[i]}$. Here, the function $y^{[i]} : \mathbb{N}_0 \to M^n$ (describing the messages received by agent i) has components

$$y_j^{[i]}(t) = \begin{cases} \text{msg}(x^{[j]}(t), w^{[j]}(t), i), & \text{if } (i,j) \in E_{\text{cmm}}(x(t)), \\ \texttt{null}, & \text{otherwise.} \end{cases}$$

□

IV. Network abstract linear programming

In this section we define a *network abstract linear program* and propose novel distributed algorithms to solve it.

A. Problem statement

Informally we can say that a *network abstract linear program* consists of three main elements: a network, an abstract linear program and a mapping that associates to each constraint of the abstract linear program a node of the network. A more formal definition is the following.

Definition 4.1: A network abstract linear program (NALP) is a tuple $(\mathcal{G}, (H, \omega), \mathcal{B})$ consisting of

(i) $\mathcal{G} = (I, E_{\text{cmm}})$, a communication digraph;
(ii) (H, ω), an abstract linear program;
(iii) $\mathcal{B} : H \to I$, a surjective map called *constraint distribution map*. □

The *solution* of the network abstract linear program is attained when all processors in the network have computed a solution to the abstract linear program.

Remark 4.2: Our definition allows for various versions of network abstract linear programs. Regarding the constraint distribution map, the most natural case to consider is when the constraint distribution map is bijective. In this case one constraint is assigned to each node. More complex distribution laws are also interesting depending on the computation power and memory of the processors in the network. In what follows, we assume $\mathcal{B}$ to be bijective. □

B. Distributed algorithms

Next we define three distributed algorithms that solve network abstract linear programs. First, we describe a synchronous version that is well suited for time-dependent networks whose nodes have bounded computation time and memory, but also bounded in-degree or equivalently arbitrary in-degree, but also arbitrary computation time and memory. Then we describe two variations that take into account the problem of dealing with arbitrary in-degree versus short computation time and small memory. The second version of the algorithm is suited for time-dependent networks that have arbitrary in-degree and bounded computation time, but are allowed to store arbitrarily large amount of information, in the sense that the number of stored messages may depend on the number of nodes of the network. The third algorithm considers the case of time-independent networks with arbitrary in-degree and bounded computation time and memory.

In the algorithms we consider a uniform network $\mathcal{S}$ with communication digraph $\mathcal{G} = (I, E_{\text{cmm}})$ and a network abstract linear program $(\mathcal{G}, (H, \omega), \mathcal{B})$. We assume $\mathcal{B}$ to be bijective, that is, the set of constraints H has dimension n, $H = \{h_1, \cdots, h_n\}$. The combinatorial dimension is δ.

Here is an informal description of what we shall refer to as the *FloodBasis* algorithm:

> *[Informal description]* Each processor has a logical state of $\delta + 1$ variables taking values in H. The first δ components represent the current value of the basis to compute, while the last element is the constraint assigned to that node. At the start round the processor initializes every component of the basis to its constraint, then, at each communication round, performs the following tasks: (i) it acquires from its neighbors (a message consisting of) their current basis; (ii) it executes the `SUBEX_lp` algorithm over the constraint set given by the collection of its and its neighbors' basis and its constraint (that it maintains in memory), thus computing a new basis; (iii) it updates its logical state and message using the new basis obtained in (ii).

In the second scenario we work with a time-dependent network with no bounds on the in-degree of the nodes and on the memory size. In this setting the execution of the `SUBEX_lp` may exceed the communication round length. In order to deal with this problem, we slightly change the network model as described in Section III, so that each processor may execute the state transition function "asynchronously", in the sense that the time-length of the execution may take multiple rounds. If that happens, the message generation function in each intermediate round is called using the logical state of the previous round. Here is an informal description of what we shall refer to as the *FloodBasisMultiRound* algorithm:

> *[Informal description]* Each processor has the same message alphabet and logical state as in *FloodBasis* and also the same state initialization. At each communication round it performs the following tasks: i) it acquires the messages from its in-neighbors; ii) if the execution of the `SUBEX_lp` at the previous round was over it starts a new instance, otherwise it keeps executing the one in progress; iii) if the execution of the `SUBEX_lp` ends it updates the logical state and runs the message-generation function with the new state, otherwise it generates the same message as in the previous round.

In the third scenario we work with a time-independent network with no bounds on the in-degree of the nodes. We suppose that each processor has limited memory capacity, so that it can store at most D messages. The memory is dimensioned so to guarantee that the `SUBEX_lp` is always solvable during two communication rounds. The memory constraint is solved by processing only part of the incoming messages at each round and cycling in a suitable way in order to process all the messages in multiple rounds.

Here is an informal description of what we shall refer to as the *FloodBasisCycling* algorithm:

> *[Informal description]* The first $\delta + 1$ components of the logical state are the same as in *FloodBasis* and are initialized in the same way. A further component is added. It is simply a counter variable that keeps trace of the current round. At each communication round each processor performs the following tasks: (i) it acquires from its neighbors (a message consisting of) their current basis; (ii) it chooses D messages according to a scheduled protocol, e.g. it labels its in-neighboring edges with natural numbers from 1 up to $\texttt{indeg}^{[i]}$ and cycles over them in increasing order; (iii) it executes the `SUBEX_lp` algorithm over the constraint set given by the collection of the D messages plus its basis and its constraint (that it maintains in memory), thus computing a new basis; (iv) it updates its logical state and message using the new basis obtained in (iii).

Remark 4.3: For the algorithm to converge it is important that each agent keeps in memory its constraint and thus implements the `SUBEX_lp` on the bases received from its neighbors together with its constraint. This requirement is important because of the following reason: no element of a basis B for a set $G \subset H$ needs to be an element in the basis of $G \cup \{h\}$ for any $h \in H \setminus G$. □

We are now ready to prove the algorithms' correctness.

Proposition 4.4 (Correctness of FloodBasis*):* Let $\mathcal{S}$ be a synchronous time-dependent network with communication

digraph $\mathcal{G} = (I, E_{\text{cmm}})$ and let $(\mathcal{G}, (H, \omega), \mathcal{B})$ be a network abstract linear program. If $\mathcal{G}$ is jointly strongly connected, then the *FloodBasis* algorithm solves $(\mathcal{G}, (H, \omega), \mathcal{B})$, that is, in a finite number of rounds each node acquires a copy of the solution of (H, ω), i.e., the basis B of H.

Proof: In order to prove correctness of the algorithm, observe, first of all, that each law at every node converges in a finite number of steps. In fact, using axioms from abstract linear programming and finiteness of H, each sequence $\omega(B^{[i]}(t))$, $t \in \mathbb{N}_0$, is monotone nondecreasing, upper bounded and can assume a finite number of values. Then we proceed by contradiction to prove that all the laws converge to the same $\omega(B)$ and that it is exactly $\omega(B) = \omega(H)$. Suppose that for $t > t_0 > 0$ all the nodes have converged to their limit basis and that there exist at least two nodes, call them i and j, such that $\omega(B^{[i]}(t)) = \omega(B^{[i]}) \neq \omega(B^{[j]}) = \omega(B^{[j]}(t))$, for all $t \geq t_0$. For $t = t_0 + 1$, for every $k_1 \in \mathcal{N}_O(i)$, $B^{[i]}$ does not violate $B^{[k_1]}$, otherwise they would compute a new basis thus violating the assumption that they have converged. Using the same argument at $t = t_0 + 2$, for every $k_2 \in \mathcal{N}_O(k_1)$, $B^{[k_1]}$ does not violate $B^{[k_2]}$. Notice that this does not imply that $B^{[i]}$ does not violate $B^{[k_2]}$, but it implies that $\omega(B^{[i]}) \leq \omega(B^{[k_2]})$. Iterating this argument we can show that for every $S > 0$, every k connected to i in the graph $\cup_{t=t_0}^{t_0+S} \mathcal{G}(t)$ must have a basis $B^{[k]}$ such that $\omega(B^{[i]}) \leq \omega(B^{[k]})$. However, using the joint connectivity assumption, there exists $S_0 > 0$ such that $\cup_{t=t_0}^{t_0+S_0} \mathcal{G}(t)$ is strongly connected and therefore i is connected to j, thus showing that $\omega(B^{[i]}) \leq \omega(B^{[j]})$. Repeating the same argument by starting from node j we obtain that $\omega(B^{[j]}) \leq \omega(B^{[i]})$, that implies $\omega(B^{[i]}) = \omega(B^{[j]})$, thus giving the contradiction. Now, the basis at each node satisfies, by construction, the constraints of that node. Since the basis is the same for each node, it satisfies all the constraints, then $\omega(B) = \omega(H)$. ■

Remark 4.5: Correctness of the other two versions of the *FloodBasis* algorithm may be established along the same lines. For example, it is immediate to establish that the basis at each node reaches a constant value in finite time. It is easy to show that this constant value is the solution of the abstract linear program for the *FloodBasisMultiRound* algorithm. For the *FloodBasisCycling* algorithm we note that the procedure used to process the incoming data is equivalent to considering a time-dependent graph whose edges change with that law. □

Proposition 4.6 (Halting condition): Consider a network $\mathcal{S}$ with time-independent, strongly connected digraph $\mathcal{G}$ where the *FloodBasis* algorithm is running. Each processor can halt the algorithm execution if the value of its basis has not changed after $2\,\text{diam}(\mathcal{G}) + 1$ communication rounds.

Proof: First, notice that, for all $t \in \mathbb{N}_0$ and for every $(i, j) \in E_{\text{cmm}}$,

$$\omega(B^{[i]}(t)) \leq \omega(B^{[j]}(t+1)). \tag{1}$$

This holds by simply noting that $B^{[j]}(t+1)$ is not violated by $B^{[i]}(t)$ by construction of the *FloodBasis* algorithm. Assume that node i satisfies $B^{[i]}(t) = B$ for all $t \in \{t_0, \ldots, t_0 + 2\,\text{diam}(\mathcal{G})\}$, and pick any other node j. Without loss of generality assume that $t_0 = 0$. Because of equation (1), if $k_1 \in \mathcal{N}_O(i)$, then $\omega(B^{[k_1]}(1)) \geq \omega(B)$ and, recursively, if $k_2 \in \mathcal{N}_O(k_1)$, then $\omega(B^{[k_2]}(2)) \geq \omega(B^{[k_1]}(1)) \geq \omega(B)$. Iterating this argument $\text{dist}(i, j)$ times, the node j satisfies $\omega(B^{[j]}(\text{dist}(i, j))) \geq \omega(B)$. Now, consider the out-neighbors of node j. For every $k_3 \in \mathcal{N}_O(j)$, it must hold that $\omega(B^{[k_3]}(\text{dist}(i, j) + 1)) \geq \omega(B^{[j]}(t))$. Iterating this argument $\text{dist}(j, i)$ times, the node i satisfies $\omega(B^{[i]}(\text{dist}(i, j) + \text{dist}(j, i))) \geq \omega(B^{[j]}(\text{dist}(i, j)))$. In summary, because $\text{dist}(i, j) + \text{dist}(j, i) \leq 2\,\text{diam}(\mathcal{G})$, we know that $B^{[i]}(\text{dist}(i, j) + \text{dist}(j, i)) = B$ and, in turn, that

$$\omega(B) \geq \omega(B^{[j]}(\text{dist}(i, j))) \geq \omega(B).$$

This shows that, if basis i does not change for a duration $2\,\text{diam}(\mathcal{G}) + 1$, then it will never change afterwards because all bases $B^{[j]}$, for $j \in \{1, \ldots, n\}$, have cost equal to $\omega(B)$ at least as early as time equal to $\text{diam}(\mathcal{G}) + 1$. Therefore, node i can safely stop after a $2\,\text{diam}(\mathcal{G}) + 1$ duration. ■

V. Formation control for robotic networks

In this section we introduce the problem of minimum time formation for a robotic network. We focus on the formation control problem for a point formation (rendezvous), a line formation and a circle formation. A control and communication law based on the distributed algorithm of the previous section is proposed as an approximate solution.

We consider the following robotic network. Each agent i occupies a location $p^{[i]} \in \mathbb{R}^d$, $d \in \mathbb{N}$, and moves according to the first order discrete-time integrator $p^{[i]}(t+1) = p^{[i]}(t) + u^{[i]}(t)$. The communication edge map is the one arising according to the *disk graph*, E_{disk}; note that the disk graph is undirected. Each control $u^{[i]}$ takes values in the bounded subset of $\mathbb{R}^d$ $B(0, r_{\text{ctr}})$, that is, $\|u^{[i]}\|_2 \leq r_{\text{ctr}}$.

A. Formation tasks

In the literature many definitions of formation have been given and studied. Here we consider a somehow specific situation. We let $F_0 \subset \mathbb{R}^d$ be a "nominal" subset of the state space and let $\alpha \mapsto F_\alpha$, $\alpha \in \mathbb{R}^m$, be a mapping that provides a parametrization of F_0. For example F_0 could be the x axis in the plane and F_α the set of lines translated and rotated in the plane. We ask the network to reach the configuration where all the agents' states belong to the same subset F_α (for some α). The *formation task* is defined as

$$\mathcal{T}_{\text{form}}(x) = \begin{cases} \texttt{true}, & \text{if } \exists \alpha \text{ s.t. } x^{[i]} \in F_\alpha, x^{[j]} \in F_\alpha \\ & \qquad \text{for all } (i, j) \in E_{\text{cmm}}(x), \\ \texttt{false}, & \text{otherwise.} \end{cases}$$

Note that, if the digraph is not connected, then the formation task is achieved if agents belonging to each disconnected component belong to the same set F_α, for possibly different values of α.

In the following we are interested in formation to a point (rendezvous), a line and a circle for $d = 2$. It is easy to reformulate these tasks in terms of appropriate sets F_0 and F_α. For the *rendezvous* task, $\mathcal{T}_{\text{rndzvs}}$, F_0 is, e.g., the origin of the reference frame and each F_α is a point in the plane. For the *line-formation* task, $\mathcal{T}_{\text{lform}}$, F_0 is, e.g., the x axis and each F_α is a line in the plane. Finally, for the *circle-formation* task, $\mathcal{T}_{\text{cform}}$, F_0 may be taken as the circle of unit radius centered at the origin, and each F_α is a circle (with different center and radius) in the plane.

B. Minimum-time formation control

Having defined the formation tasks for the robotic network, we ask whether such tasks can be accomplished in minimum time. The problem may be formalized as follows.

$$\begin{aligned} \min \quad & T \\ \text{subj. to:} \quad & \mathcal{T}_{\text{form}}(x(t)) = \texttt{true} \text{ for all } t \geq T, \\ & t \mapsto x(t) \text{ is an evolution of the robotic network.} \end{aligned}$$

An important property of the minimum time formation to a point and a line is that the centralized version of these problems is equivalent to finding the smallest enclosing ball and stripe enclosing the n agents. Recall from Remark 2.1 that these problems are found to be abstract linear programs. The property is formalized in the following proposition.

Lemma 5.1: Let P be the set of points in $\mathbb{R}^2$, L the set of lines in $\mathbb{R}^2$ and C the set of circles in $\mathbb{R}^2$. Given the set of points $P_n = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$, consider the problems:

$$\min_{p \in P} \max_{p_j \in P_n} \|p_j - p\|, \qquad \min_{l \in L} \max_{p_j \in P_n} \operatorname{dist}(p_j, l),$$

where, given a set S and a point p, $\operatorname{dist}(p, S)$ is the distance of p from S, that is $\operatorname{dist}(p, S) = \min_{s \in S} \|p - s\|$. These are equivalent to the problems of finding the smallest enclosing ball and stripe of the point set, which are abstract linear programs. For Problem (ii) we need the further assumption that the point set is in generic position in the sense that, given any two groups of three distinct points p_1, p_2, p_3 and p_4, p_5, p_6 (where points in distinct groups may coincide), the distance between the point p_1 and the line through the points p_2 and p_3 is different from the distance between the point p_4 and the line through the points p_5 and p_6. □

Remark 5.2 (Circle-formation): The centralized version of the minimum time circle-formation problem is equivalent to the problem of finding the minimum-width annulus containing the point-set. This problem is very complex and cannot be tackled via abstract linear programming. For arbitrary data sets, the minimum-width annulus has arbitrarily large minimum radius and, even worse, it is possible that the minimum-width annulus is a stripe. Typically, in such configurations, all points are contained only in a small fraction of the annulus. This is not the solution we envision when we consider placing robots in circle. A related, simpler problem is the smallest-area annulus. The cost function in this problem, that is known to be an abstract linear program, penalizes both the difference of the radiuses of the annulus (width of the annulus) and their sum. Typical solutions to smallest-area annulus are annuli where the points are more uniformly distributed; circle-formation problems will be analyzed in detail in a forthcoming paper. □

C. Move-toward-estimate control and communication law

We have shown that the centralized solution of the minimum time formation to point and line may be found by solving an abstract linear program. Next, we use the distributed algorithms introduced in Section IV to design a control and communication law that approximates the centralized solution of the minimum time formation problem.

The *move-toward-estimate* control and communication law may be summarized as follows. On the basis of their initial positions, the agents run the *FloodBasis* algorithm for the problem of interest (smallest enclosing ball or stripe). A possibility could be to wait for the algorithm to end, then move (at maximum speed) toward the optimal set. We propose a slightly different strategy. While the algorithm is running, each agent starts moving toward the set corresponding to its own current estimate of the solution. Everyone does it while maintaining connectivity with its current neighbors. In order to speed up the process and to guarantee convergence, connectivity is no longer enforced once the algorithm reaches the halting condition (meaning that the network abstract linear program has been solved).

Proposition 5.3 (Move-toward-estimate correctness): On the network $\mathcal{S}$ with communication edge map E_{disk} and bound on the ith control input $u^{[i]} \in B(0, r_{\text{ctr}})$, the move-toward-estimate control and communication laws achieve the task $\mathcal{T}_{\text{rndzvs}}$ and $\mathcal{T}_{\text{lform}}$ respectively. For the line-formation task we need the further assumption that the point set of initial conditions is in generic position.

Proof: By the connectivity arguments done before and by Proposition 4.4 we know that there exists $\overline{T} \in \mathbb{N}_0$ such that for $t = \overline{T}$ the network is connected and all the agents have solved the network abstract linear program. Since this instant all the agents can move toward the target set (point or line) at maximum speed without enforcing connectivity constraints anymore. Thus, they achieve the task. ■

In [11] the reader can find a simulation of the control and communication law for the rendezvous problem in the plane.

REFERENCES

[1] J. Matousek, M. Sharir, and E. Welzl, "A subexponential bound for linear programming," *Algorithmica*, vol. 16, no. 4/5, pp. 498–516, 1996.

[2] B. Gärtner, "A subexponential algorithm for abstract optimization problems," *SIAM J Computing*, vol. 24, no. 5, pp. 1018–1035, 1995.

[3] N. Megiddo, "Linear programming in linear time when the dimension is fixed," *J. Assoc. Computing Mach.*, vol. 31, no. 1, pp. 114–127, 1984.

[4] B. Gärtner and E. Welzl, "Linear programming - randomization and abstract frameworks," in *Symposium on Theoretical Aspects of Computer Science*, ser. Lect. Notes Comp. Science, vol. 1046, 1996, pp. 669–687.

[5] P. K. Agarwal and S. Sen, "Randomized algorithms for geometric optimization problems," in *Handbook of Randomization*, P. Pardalos, S. Rajasekaran, J. Reif, and J. Rolim, Eds. Dordrecht, The Netherlands: Kluwer, 2001.

[6] M. Ajtai and N. Megiddo, "A deterministic $\operatorname{poly}(\log \log n)$-time n-processor algorithm for linear programming in fixed dimension," *SIAM J Computing*, vol. 25, no. 6, pp. 1171–1195, 1996.

[7] I. Suzuki and M. Yamashita, "Distributed anonymous mobile robots: Formation of geometric patterns," *SIAM J Computing*, vol. 28, no. 4, pp. 1347–1363, 1999.

[8] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita, "Distributed memoryless point convergence algorithm for mobile robots with limited visibility," *IEEE Trans Robotics & Automation*, vol. 15, no. 5, pp. 818–828, 1999.

[9] M. Egerstedt and X. Hu, "Formation constrained multi-agent control," *IEEE Trans Robotics & Automation*, vol. 17, no. 6, pp. 947–951, 2001.

[10] H. G. Tanner, G. J. Pappas, and V. Kumar, "Leader-to-formation stability," *IEEE Trans Robotics & Automation*, vol. 20, no. 3, pp. 443–455, 2004.

[11] G. Notarstefano and F. Bullo, "Distributed consensus on enclosing shapes and minimum time rendezvous," in *Proc CDC*, San Diego, CA, Dec. 2006, pp. 4295–4300.

[12] N. A. Lynch, *Distributed Algorithms*. San Mateo, CA: Morgan Kaufmann Publishers, 1997.

[13] S. Martínez, F. Bullo, J. Cortés, and E. Frazzoli, "On synchronous robotic networks – Part I: Models, tasks, and complexity. Part II: Time complexity of rendezvous and deployment algorithms," *IEEE Trans Automatic Ctrl*, 2007, to appear.