A Distributed Simplex Algorithm and the Multi-Agent Assignment Problem

Mathias Bürger, Giuseppe Notarstefano, Frank Allgöwer and Francesco Bullo

***Abstract*— In this paper we propose a novel distributed algorithm to solve degenerate linear programs on asynchronous networks. Namely, we propose a distributed version of the well known simplex algorithm. We prove its convergence to the global lexicographic minimum for possibly fully degenerate problems and provide simulations supporting the conjecture that the completion time scales linearly with the diameter of the graph. The algorithm can be interpreted as a dual version of the *constraints consensus* algorithm proposed in [1] to solve abstract programs when the last is applied to linear programs. Finally, we study a multi-agent task assignment problem and show that it can be solved by means of our distributed simplex algorithm.**

I. Introduction

The increasing interest in performing complex tasks via multi-agent systems (e.g. sensor and robotic networks) has raised the interest in solving distributed optimization problems. The fundamental paradigms in distributed computation are that: (i) information, relevant for the solution of the problem, is distributed all over a network of processors with limited memory and computation capability, and (ii) the overall computation relies only on local computation and information exchange amongst neighboring processors. Optimizing linear objectives over linear constraints takes a central role in the optimization literature and thus deserves particular attention also in distributed computation. In this paper we consider a distributed version of linear programs. In particular, we consider linear programs in standard form where the number of decision variables is much larger than the number of equality constraints. Each processor in the network is assigned only the information relative to a subset of the decision variables. The objective is to agree on a global minimum of the problem, if it exists, or agree that the problem is either unbounded or infeasible. Particular attention is given to degenerate linear programs, in which more than one solution is optimal and a mutual agreement of all agents on one solution is required.

Although distributed and parallelized optimization algorithms have been studied for a long time, see e.g. [2], the multi-agent perspective has recently become subject of renewed interest in the control and optimization theory community. While several interior point algorithms were proposed to solve quadratic programs and other convex programs [3], [4], [5], [6], to the best of our knowledge a theory for distributed linear programming is missing. The idea of parallelizing and distributing the computation of the simplex algorithm has been exploited in the computer science literature for a while, see e.g. [7], [8]. An alternative contribution in this area, focusing in particular on the multi-agent requirements, is given in [1], see also [9], where *distributed abstract programs* are introduced as a general class of optimization problems including linear programs and a *constraints consensus* algorithm is proposed to solve them. Following the trail opened in [1], in this paper we propose a distributed version of the well known and widely used simplex method [10] to solve linear programming in a distributed way.

The contributions of this paper are threefold. First, we introduce a distributed version of the well known simplex algorithm to solve degenerate linear programs in asynchronous networks with time-varying directed communication topology. Each agent has a candidate basis that it exchanges with its neighbors and updates by iteratively performing *pivot* operations on a subset of columns given by its basis, its neighbors' candidate bases and its original columns it has been assigned. The new algorithm relies only on the two step procedure of exchanging information and performing the pivot operation. This can be done by each agent on its own speed, and does not require network-wide coordination by consensus-type algorithms between the update steps. This distinguishes our algorithm form other distributed simplex algorithms. Second, we characterize the main properties of the algorithm. In detail, we show that our proposed algorithm can deal with fully degenerate linear programs. This is obtained by modifying the local solver at each node so that the pivot steps implement a *lexicographic ratio test*. As a consequence of using lexicographic ordering, we prove that each candidate basis converges to the unique lexicographic minimum of the problem. The main idea behind the proposed distributed simplex algorithm is inspired by the constraints consensus algorithm proposed in [1]. In fact, in this paper we show that, if constraints consensus is implemented for linear programs in standard dual form by using a lexicographic test, then the proposed distributed simplex turns to be its dual implementation. Third and final, we show how to apply the proposed algorithm to the multi-agent assignment problem which is known to be highly primal degenerate. We show that, thanks to the special structure of the problem, if N is the number of agents and, thus, N^2 the number of decision variables, the algorithm can be efficiently implemented by storing and exchanging $\mathcal{O}(N \log_2 N)$ bytes. We provide numerical computations supporting our conjecture that the distributed simplex algorithm scales linearly with the diameter of the graph in networks with fixed topologies.

The remainder of the paper is organized as follows. Section II reviews some fundamental principles of linear programming and introduces a non-standard version of the simplex algorithm that will be used as local solver in the distributed set-up. Section III introduces the multi-agent system and the communication network. The main results of the paper are presented in Section IV, where a distributed simplex algorithm for degenerate linear programs is proposed and analyzed. The usefulness of the algorithm is illustrated in Section V, where the application to the class of multi-agent assignment problems is discussed and simulation results are provided for a time complexity analysis of the algorithm.

M. Bürger and F. Allgöwer thank the German Research Foundation (DFG) for financial support within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart and within the Priority Program 1305 "Control Theory of Digitally Networked Dynamical Systems". The research of G. Notarstefano has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 224428 (CHAT) and n. 231378 (CO3AUV) and from the Italian Minister under the national project "Sviluppo di nuovi metodi e algoritmi per lidentificazione, la stima bayesiana e il controllo adattativo e distribuito." The work by F. Bullo is partly supported by NSF Award CNS-0834446.

M. Bürger and F. Allgöwer are with the Institute for Systems Theory and Automatic Control, University of Stuttgart, Pfaffenwaldring 9, 70550 Stuttgart, Germany, {buerger, allgower}@ist.uni-stuttgart.de.

Giuseppe Notarstefano is with the Department of Engineering, University of Lecce, 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, CA 93106, USA, bullo@engineering.ucsb.edu

II. Centralized Linear Programming

Some preliminaries concerning linear programming are reviewed in this section. The goal is to lay the ground for the remainder of the paper.

A. Problem formulation and optimality conditions

Throughout this paper, we consider linear programs in the standard equality form

$$\begin{aligned} &\min\ c^T x \\ &\text{s.t. } Ax = b \quad x \geq 0, \end{aligned} \tag{1}$$

where $A \in \mathbb{R}^{d\times n}$, $b \in \mathbb{R}^d$ and $c \in \mathbb{R}^n$, are the problem data and $x \in \mathbb{R}^n$ is the vector of decision variables. In this paper, we are interested in problems where the number of decision variables n is significantly larger than the number of constraints d. Without loss of generality, it can be assumed that $\mathrm{rank}(A) = d$, since otherwise constraints are redundant and can be ignored. A problem in form (1) is called the *primal problem*. The *dual problem* of the standard linear program (1) is

$$\max\ b^T y \quad \text{s.t. } A^T y \leq c. \tag{2}$$

While the primal problem has n decision variables and d constraints, this relation is just reversed in the dual. In the following we will be working with the primal problem, but will sometimes explicitly refer to the dual formulation. We call a *column* of problem (1), a vector $\mathrm{h}_i \in \mathbb{R}^{1+d}$ defined as

$$\begin{bmatrix} c^T \\ A \end{bmatrix} = \begin{bmatrix} c_1, \ldots, c_n \\ A_1, \ldots, A_n \end{bmatrix} =: [\mathrm{h}_1, \ldots, \mathrm{h}_n], \tag{3}$$

where $c_i \in \mathbb{R}$ are the coefficients of the vector c and $A_i \in \mathbb{R}^{d\times 1}$ are the columns of A. Note that the name column refers to the primal problem formulation and that a column in the primal corresponds to an inequality constraint in the dual. The set of all columns is denoted by $\mathbb{H} = \{\mathrm{h}_i\}_{i\in\{1,\ldots,n\}}$. For any subset $\mathbb{G} \subset \mathbb{H}$, the notation c_G and A_G refers to the cost vector and the constraint matrix, respectively constructed from the columns contained in $\mathbb{G}$. The same notation x_G is used to denote the corresponding decision variables. Using this notation, a linear program (1) is fully characterized by the pair $(\mathbb{H}, b)$, that is by the set of columns and the right-hand side vector of the equality constraints. A well known notion in linear programming is the one of *basic solution*. A set of exactly d columns $\mathbb{B} \subseteq \mathbb{H}$ is called a *basis* if $\mathrm{rank}(A_B) = d$. The *fundamental theorem of linear programming* [10] states that if a linear program $(\mathbb{H}, b)$ has an optimal feasible solution, it has an optimal basic feasible solution. The optimal solution of a linear program is, in general, such that the *basic variables* $x_B \neq 0$ and the *non-basic variables* $x_N = 0$. A solution for the basic variables is determined by $x_B = A_B^{-1} b$. A basis is called a *feasible basis* if $x_B \geq 0$. Two bases $\mathbb{B}_1$ and $\mathbb{B}_2$ are said to be *adjacent*, if there exist columns $\mathrm{e} \in \mathbb{B}_2$ and $\mathrm{l} \in \mathbb{B}_1$ such that $\mathbb{B}_2 = \{\mathbb{B}_1 \cup \mathrm{e}\} \setminus \{\mathrm{l}\}$. For each feasible basis there is a value called the *primal cost*

$$z_B = c_B^T x_B. \tag{4}$$

A basis $\mathbb{B}_i$ is *optimal* if there is no other basis $\mathbb{B}_j,\ j \neq i$, such that $z_{B_i} > z_{B_j}$. In the following we will mark optimal values with a star.

Suppose a basis $\mathbb{B}$ is known, then there is immediate need for a criterion to check whether this basis gives rise to an optimal solution, that is $z_B = z^*$. Therefore, the following concept is useful. Given a basis $\mathbb{B}$ and a single non-basic column $\mathrm{h} \notin \mathbb{B}$, the *reduced cost* of the column is defined as

$$\bar{c}_h = c_h - A_h^T (A_B^{-1})^T c_B \ =:\ r^T_{\{B\cup h\}} c_{\{B\cup h\}}, \tag{5}$$

where $c_h \in \mathbb{R}$ and $A_h \in \mathbb{R}^d$ refer to the problem data related to column h. The reduced cost gives rise to the standard optimality condition in linear programming.

Theorem 2.1 (Optimality Condition, [10]): If for some given basis $\mathbb{B}$ with basic feasible solution $x_B = A_B^{-1} b$ the reduced cost $\bar{c}_h \geq 0$, for all columns $\mathrm{h} \in \mathbb{H}$, then this solution is optimal. □

B. Problem degeneracy

A difficulty in linear programming is that the optimal solution may be not unique. In this case the linear program is called *degenerate*. Commonly, two types of degeneracy are distinguished. A linear program is said to be *primal degenerate* if there is more than one basis that leads to the optimal primal solution. That is, there exist two bases $\mathbb{B}_i$ and $\mathbb{B}_j$ such that $x_{B_i} = x_{B_j} = x^*$. In a primal degenerate linear program, some of the optimal basic variables x_B are zero, and the corresponding basic columns can therefore be replaced by some non-basic columns, without changing the value z_B. A linear program is said to be *dual degenerate* if more than one primal solution is optimal. That is, there exist several bases, say $\mathbb{B}_i$ and $\mathbb{B}_j$, providing different basic solutions, $x_{B_i} \neq x_{B_j}$, while both bases admit the optimal primal cost $z_{B_i} = z_{B_j} = z^*$. A primal linear program is dual degenerate if and only if its dual problem is primal degenerate. A problem is said to be *fully non-degenerate*, if it is neither primal nor dual degenerate. Non-degeneracy is highly desirable from a computational point of view, as the following results show.

Lemma 2.2 ([11]): Every fully non-degenerate linear program has at most one optimal solution. □

Theorem 2.3 ([11]): If a linear program has an optimal solution and is fully non-degenerate, then there exists a sequence of adjacent bases from any basis $\mathbb{B}$ to the unique optimal basis $\mathbb{B}^*$. □

However, degeneracy is a very common phenomenon in linear programs and solution methods are required to deal with highly degenerate linear problems.

C. A simplex algorithm for degenerate linear programs

A well established procedure for solving linear programs is the *simplex algorithm* introduced by Dantzig [10]. An informal description of the simplex algorithm is as follows:

> *Simplex Algorithm*: Let a primal feasible basis $\mathbb{B}$ be given. While there exists an entering column $\mathrm{e} \notin \mathbb{B}$ such that $\bar{c}_e < 0$, find a leaving basic column $\mathrm{l}(\mathrm{e}) \in \mathbb{B}$ such that $(\mathbb{B} \cup \{\mathrm{e}\}) \setminus \{\mathrm{l}\}$ is again a feasible basis. Exchange the column l with e to get a new basis.

The procedure of replacing a basic column with a non-basic one is called *pivot*. For a non-degenerate linear program, the primal cost improves at each iteration, i.e. $z_{B\cup\{e\}\setminus\{l\}} < z_B$. Since only a finite number of bases exist, the algorithm converges after a finite number of steps to the optimal solution. However, non-degeneracy is often not met and basically two problems arise in the application of the simplex algorithm to degenerate linear programs: (i) cycling among multiple bases (primal degeneracy), and (ii) convergence to a non-unique minimizer (dual degeneracy). While the first problem has received great attention in the literature, the second problem has been rarely studied. However, problem (ii) turns to be critical in a multi-agent setup, where several decision makers solve the problem and, thus, have to agree on the same solution.

In order to handle degeneracy, a simplex algorithm with a unique solution is presented in the following. We are using the algorithm proposed in [12], relying on results in [11]. The algorithm relies heavily on the concept of *lexicographic ordering* of vectors.

Definition 2.4 (Lex-positivity): If $\gamma = (\gamma_1, \ldots, \gamma_r)$ is a vector, then it is said to be *lexico-positive* (or lex-positive) if $\gamma \neq 0$ and the first non-zero component of γ is positive. □

Lexico-positivity will be denoted by the symbol $\gamma \succ 0$. Given two vectors, v and u, we say that $v \succ u$ if $v - u \succ 0$ and $v \preceq 0$

if $-v \succ 0$ or $v = 0$. Given a set of vectors $\{v_1, \ldots, v_r\}$, the lexicographical minimum, denoted *lexmin*, is the element v_i, such that $v_j \succeq v_i$ for all $j \neq i$. For the same set of vectors, we use *lexsort*$\{v_1, \ldots, v_r\}$ to refer to the lexicographically sorted set of these vectors.

The concept of lex-positivity gives rise to a refinement of the notion of feasible basis.

Definition 2.5 (Lex-feasibility): A feasible basis $\mathbb{B}$ is called *lexicographically feasible* (lex-feasible) if every row of the matrix $[A_B^{-1}b,\ A_B^{-1}]$ is lex-positive. □

It is well known in the literature [10], see also [12], that the simplex method can be refined such that after a pivot iteration the new basis is again primal lex-feasible. This is done by choosing the leaving columns according to the *lexicographic ratio test*. Let $\mathbb{B}$ be a lex-feasible basis and e the entering column, then the leaving column is chosen as

$$\mathrm{l}_{\mathrm{lex}}(\mathrm{e}) = \arg \operatorname*{lexmin}_{\mathrm{j} \in \mathbb{B}} \{[A_B^{-1}b,\ A_B^{-1}]_{\bullet j}/(A_B^{-1}A_e)_{\bullet j} | \ (A_B^{-1}A_e)_{\bullet j} > 0\}, \tag{6}$$

where the subscript $\bullet j$ denotes selection of the j-th row of the matrix, respectively vector. It is well known that the lexicographic ratio test is equivalent to a perturbation making the problem primal non-degenerate [10]. Such a selection rule prevents the simplex algorithm from cycling.

In a multi-agent setting, we want the algorithm to converge to a *unique* optimal solution. This is usually not required in centralized linear programming, where all optimal solutions are equally desirable. However, distributed algorithms have to ensure that all computation nodes end up with the same solution. In order to guarantee convergence of the algorithm to a *unique* solution, the optimization criterion can be suitably modified. We change the minimization objective from the primal cost $z = c^T x$ to the lexicographically perturbed cost

$$\phi(x) = c^T x + \delta_0 x_1 + \delta_0^2 x_2 + \ldots + \delta_0^n x_n, \tag{7}$$

where x_i represents the i-th component of the vector x. For a sufficiently small constant δ_0, the optimizer x^*=arg $\min_x \phi(x)$ corresponds to the unique optimal solution of (1) which is minimal in a lexicographic sense. Now let $\delta = [\delta_0, \ldots, \delta_0^n]^T$ and write $\phi = (c^T + \delta^T)x$. A column becomes admissible with respect to the perturbed cost ϕ if $\bar{c}_e = r_{\{B\cup e\}}^T c_{\{B\cup e\}} + r_{\{B\cup e\}}^T \delta < 0$. This in turn is equivalent to requiring

$$[r_{\{B\cup e\}}^T c_{\{B\cup e\}}, r_{\{B\cup e\}}^T] \prec 0. \tag{8}$$

Remark 2.6: The perturbation used in (7) and (8) requires an ordering of the decision variables, respectively columns. We will discuss a suitable perturbation for distributed linear programming later on. □

Next, we give a pseudo code description of the lexicographically modified `Pivot` algorithm and, consistently, of the `Simplex` algorithm. The `Pivot` in this form is proposed in [12].

Algorithm 1 `Pivot` $(\mathbb{B}, \mathrm{e})$

Require: A lex-feasible basis $\mathbb{B}$, a non-basic column $\mathrm{e} \notin \mathbb{B}$
  if $[r_{\{B\cup e\}}^T c_{\{B\cup e\}},\ r_{\{B\cup e\}}^T] \prec 0$ **then**
    select the leaving column $\mathrm{l}_{\mathrm{lex}}(\mathrm{e})$ via *lex ratio test* (6)
    if $\mathrm{l}_{\mathrm{lex}}(\mathrm{e}) \neq \oslash$ **then**
      $\mathbb{B} \leftarrow (\mathbb{B} \cup \{\mathrm{e}\}) \setminus \{\mathrm{l}_{\mathrm{lex}}(\mathrm{e})\}$ *% make the pivot*
    else
      $\mathbb{B} \leftarrow$ `null` *% problem is unbounded*
    end if
  end if
  Return $\mathbb{B}$

Algorithm 2 `Simplex` $(\mathbb{H}, \mathbb{B})$

Require: A set of columns $\mathbb{H}$, a lex-feasible basis $\mathbb{B} \subseteq \mathbb{H}$
  while $\exists \mathrm{e} \in \mathbb{H}$ such that $[\mathrm{r}_{\{B\cup e\}}^T c_{\{B\cup e\}}, \mathrm{r}_{\{B\cup e\}}^T] \prec 0$ **do**
    $\mathbb{B} \leftarrow$ `Pivot` $(\mathbb{B}, \mathrm{e})$
  end while

The proposed simplex algorithm requires a lex-feasible basis for the initialization. Several procedures are known in the literature to initialize a simplex algorithm. In this paper we use the *big-M method*. We assume without loss of generality that each entry of the vector b is non-negative. Then, *artificial decision variables*, $\hat{x}_1, \ldots \hat{x}_d$, are introduced. Corresponding to the artificial decision variables, an initial basis $\mathbb{B}_M$ is defined as follows. Choose $A_{B_M} = \mathbb{I}_q$ and $c_{B_M} = M \cdot 1$, where $\mathbb{I}_q$ is the $d \times d$ identity matrix, and 1 is a d-dimensional vector of ones. The cost coefficients are all given the value $M > 0$, which is chosen larger than any cost coefficient that possibly occurs in the original problem, i.e. $M \gg \max_{i=1,\ldots,n}(c_i)$.

In the following, the columns corresponding to the initial basis are denoted by $\hat{\mathrm{h}}_i$. An initial basis defined in this way is primal lex-feasible, since every row of $[\mathbb{I}^{-1}b, \mathbb{I}^{-1}]$ is lexicographic positive.

III. Network model

Before introducing the network model we need some definitions from graph theory. Let $\mathcal{G}_c = (\{1, \ldots, N\}, E_c)$ denote a directed, static graph (digraph). The set $\{1, \ldots, N\}$ are the nodes of the graph, corresponding to unique identifiers of the agents. The set $E_c \subset \{1, \ldots, N\}^2$ denotes the set of edges connecting two nodes. The number of edges going out from (coming into) node i is called the outdegree (indegree). A digraph is said to be *strongly connected* if, for every pair of nodes $(i, j) \in \{1, \ldots, N\} \times \{1, \ldots, N\}$, there exists a path of directed edges that goes from i to j. In a directed graph, the minimum number of edges between node i and j is called the *distance* from i to j and is denoted by $\mathrm{dist}(i, j)$. The maximum $\mathrm{dist}(i, j)$ taken over all pairs (i, j) is the *diameter* of the graph $\mathcal{G}_c$ and is denoted by $\mathrm{diam}(\mathcal{G}_c)$. We allow the communication network to be time-varying and therefore to be described by a time-dependent digraph of the form $\mathcal{G}_c(t) = (V, E(t))$, $t \in \mathbb{R}_{\geq 0}$, where t represents the universal time. The set of outgoing (incoming) neighbors of node i at time t are the set of nodes to (from) which there are edges from (to) i at time t. They are denoted by $\mathcal{N}_{\mathrm{out}}(i, t)$ and $\mathcal{N}_{\mathrm{in}}(i, t)$, respectively.

A graph $\mathcal{G}_c(t)$ models the communication in the network in the sense that at time t there is an edge from node i to node j if and only if agent i transmits information to agent j at time t. In the rest of the paper we use the following assumption.

Assumption 3.1 (Periodically Strong Connectivity): There exists a positive and bounded constant T_c such that for every time instant $t \in \mathbb{R}_{\geq 0}$, the digraph $\mathcal{G}_c^{T_c}(t) := \cup_{\tau=t}^{T_c} \mathcal{G}_c(\tau)$ is strongly connected. □

The agents in the network perform a *distributed algorithm* [13] to solve the optimization problem. In what follows, the superscript $[i]$ denotes that a quantity belongs to agent i. A distributed algorithm consists of: (1) the set W, called the set of *states* $w^{[i]}$, (2) the set Σ, called the *communication alphabet* including the null element, (3) the map $\mathtt{MSG} : W \times (1, \ldots, N) \to \Sigma$, called *message function*, and (4) the map $\mathtt{STF} : W \times \Sigma^N \to W$, called the *state transition function*. In addition to the universal time t, we denote $t_k^{[i]}$ the time instants at which agent i updates its internal state. In this sense k is a counter for updates performed by an agent. Between two discrete updates, the state is constant $w^{[i]}(t) = w(t_k^{[i]})$ for all $t_k^{[i]} \leq t < t_{k+1}^{[i]}$. The evolution of the distributed algorithm is then as follows. The algorithm starts at $t = 0$ and each agent initializes its state to $w^{[i]}(0)$. Each agent performs two actions repeatedly: (i) the ith agent sends to each of its outgoing neighbors in the

communication graph a message computed as $\texttt{MSG}(w^{[i]}(t_k^{[i]}))$; (ii) whenever it receives information from its in-neighbors, it updates its state $w^{[i]}(t_{k+1}^{[i]})$ according to the state transition function. Each agent performs these two actions at its own speed and independent of the speed of the other agents. In this sense, no synchronization is required in the network. Following [2], we say that the algorithm is *partially asynchronous* since it performs asynchronously, but T_c imposes a global bound on the time allowed to pass between consecutive state updates.

IV. Distributed Linear Programming

In this section we describe our distributed simplex algorithm to solve linear programs of the form (1).

First, we present the notion of *distributed linear program*.

Definition 4.1 (Distributed linear program): A distributed linear program is a tuple $(\mathcal{G}_c, (\mathbb{H}, b), \mathcal{P})$ that consists of

(i) a time-varying communication graph $\mathcal{G}_c(t) = (\{1, \ldots, N\}, E_c(t))$;
(ii) a linear program $(\mathbb{H}, b)$;
(iii) a unique partitioning $\mathcal{P} = \{\mathbb{P}^{[i]},\ i = 1, \ldots, N\}$ of the problem columns, with $\mathbb{H} = \cup_{i=1}^{N} \mathbb{P}^{[i]}$. □

A solution of the distributed linear program is attained when all agents have computed the same basis solving $(\mathbb{H}, b)$.

Defining a distributed linear program in this way, implies the following properties. An entry of the decision vector x belongs to only one agent and each agent i supervises n_i decision variables, with $\sum_{i=1}^{N} n_i = n$. We write $x_{i\kappa}$ to refer to the κ-th decision variable supervised by agent i, with $\kappa = 1, \ldots, n_i$. We assume that the problem information is initially distributed all over the network of agents. That is, the each column h_{ik}, consisting of the coefficient $c_{i\kappa} \in \mathbb{R}$ and the vector $A_{i\kappa} \in \mathbb{R}^d$, is initially only available to the agent i. The information, that is permanently available to an agent, is defined by the partition $\mathcal{P}$, i.e. $\mathbb{P}^{[i]} = \{\mathrm{h}_{i\kappa} : \kappa = 1, \ldots, n_i\}$.

We will use the big-M method to locally initialize the optimization problem at every agent in the network. To do that, we need the following assumption.

Assumption 4.2: An upper bound M on the cost coefficients c_{ik} is known to every agent. □

A. Distributed Simplex Algorithm

The `Pivot` iteration, presented in Section II, is at the basis of the distributed algorithm presented below. Let us first informally outline the underlying idea of the algorithm, performing on a partially asynchronous network.

> *Distributed Simplex Algorithm*: Let $\mathcal{G}_c(t)$ be a time-varying communication graph. The state of every agent i is a lex-feasible basis, $w^{[i]}(t) = \mathbb{B}^{[i]}(t)$. Each agent initializes a lex-feasible basis using the big-M method. Each agent performs consecutively the following tasks:
>
> (i) it transmits irregularly, but at least after a time interval of maximal length T_c, its basis to all its out-neighbors;
> (ii) whenever it acquires a basis from one of its in-neighbors, it sorts all columns in its memory - its permanent columns $\mathbb{P}^{[i]}$, its local basis $\mathbb{B}^{[i]}$ and the columns received from its neighbors $\mathrm{h}^{[j]}$, $j \in \mathcal{N}_I$ - according to a lexicographic ordering and performs the `Simplex` ;
> (iii) it updates its local basis with the optimal basis computed in step (ii).

Next, we analyze the technical properties of the proposed algorithm. The following theorem summarizes the convergence properties of the Distributed Simplex algorithm. The proof of the algorithm is inspired to the proof of Theorem IV.4 in [1].

`Problem data:` $((\mathbb{H}, b), \mathcal{G}_c, \mathcal{P})$
`Algorithm:` Distributed Simplex
`Message alphabet:` $\Sigma = \mathbb{H}^d \cup \{\texttt{null}\}$
`Processor state:` $\mathbb{B}^{[i]} \subset \mathbb{H}$ with $\mathrm{card}(\mathbb{B}) = d$
`Initialization:` $\mathbb{B}^{[i]} := \mathbb{B}_M$

`function` $\texttt{MSG}(\mathbb{B}^{[i]}, j)$
  return all $\mathrm{h}^{[i]}$ contained in $\mathbb{B}^{[i]}$ but not in $\mathbb{B}_M$.

`function` $\texttt{STF}(\mathbb{B}^{[i]}, y)$
% executed by agent i, with $y_j := \texttt{MSG}(\mathbb{B}^{[j]}, i) = \mathbb{B}^{[j]}$
  if $y_j \neq \texttt{null}$ for all $j \in \mathcal{N}_I(i)$ **then**
    $\mathbb{H}^{tmp} \leftarrow \mathrm{lexsort}\{\mathbb{P}^{[i]} \cup \mathbb{B}^{[i]} \cup \big(\cup_{j \in \mathcal{N}_I(i)}\, y_j\big)\}$
    $\mathbb{B}^{[i]} \leftarrow \texttt{Simplex}\ (\mathbb{H}^{tmp}, \mathbb{B}^{[i]})$
  else
    $\mathbb{B}^{[i]} \leftarrow \texttt{null}$
  end if

Theorem 4.3: Consider a distributed linear program $(\mathcal{G}_c(t), (\mathbb{H}, b), \mathcal{P})$ with periodically strongly connected network $\mathcal{G}_c(t)$, $t \in \mathbb{R}_{\geq 0}$. Let the agents run the Distributed Simplex algorithm. Then there exists a finite time T_f such that

(i) if the centralized problem $(\mathbb{H}, b)$ has a finite optimal solution, the candidate bases $\mathbb{B}^{[i]}$ of all agents have converged to the same lex-optimal basis;
(ii) if the centralized problem $(\mathbb{H}, b)$ is unbounded, all agents have detected unboundedness, in the sense that all bases are the `null` symbol;
(iii) if the centralized problem $(\mathbb{H}, b)$ is infeasible, all agents can detect infeasibility, in the sense that all bases $\mathbb{B}^{[i]}$ have converged, but still contain artificial columns. □

For space constrains the proof omitted in this paper and will be provided in a forthcoming document.

Having established the convergence properties of the Distributed Simplex algorithm, we also provide a Halting Condition. We provide here, without proof, the Halting Condition proposed in [1].

Theorem 4.4 ([1]): Consider a network described by a time-independent, strongly connected digraph $\mathcal{G}_c$ implementing a Distributed Simplex algorithm. Each agent can halt the algorithm execution if the value of the basis has not changed in a time interval of length $(2\,\mathrm{diam}(\mathcal{G}_c) + 1)T_c$. □

B. Duality to Constraints Consensus Algorithm

Distributed linear programs, as they are introduced in this note, are strongly related to *distributed abstract programs* introduced in [1].

Abstract programs are a generalization of linear programs [14], usually presented in the dual form (2). They are defined by a pair (H, w), where H is a finite set with the elements called *constraints*, and $w : 2^H \to \mathcal{W}$ is a function taking values in a linearly ordered set $(\mathcal{W}, \leq)$. In distributed abstract programs, [1] , the constraints in H are distributed throughout a network of agents, similar to the column distribution considered in this note. In the Constraints Consensus algorithm agents transmit continuously constraints taken from the set H. The next proposition clarifies the relation between the Constraints Consensus algorithm, when applied to linear programs in standard dual form, and the Distributed Simplex algorithm.

Proposition 4.5: The Constraints Consensus algorithm of [1] applied to a fully non-degenerate linear program of the form (2) is dual to the Distributed Simplex algorithm. □

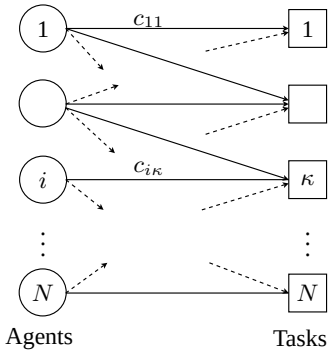

Fig. 1. Assignment Graph $\mathcal{G}_a$.

Proof: The linear program (2) is shown to be an abstract program in [1]. Each constraint of the dual problem (2) corresponds to a column of the primal problem. Since strong duality holds, for a given set of columns, respectively constraints, both algorithms compute the same primal and dual feasible solution. ■

V. The Distributed Assignment Problem

One of the most fundamental resource allocation problems (and one of the most relevant benchmark problems for linear programming) is the matching of N agents to N tasks on a one-to-one basis, while minimizing the overall cost.

A *distributed assignment problem* consists of N agents that (i) communicate over a time-varying directed communication network $\mathcal{G}_c(t)$; (ii) aim to find a one-to-one assignment to N tasks; while (iii) initially each agent i only knows the cost $c_{i\kappa}$ it takes it to perform the tasks $\kappa \in \{1, \dots, N\}$. A distributed assignment problem is solved, when all agents know the same optimal assignment.

A popular method to illustrate assignment problems is by bipartite assignment graphs $\mathcal{G}_a = \{V, W; E_a\}$, where V is the set of all agents and W is the set of tasks. Figure 1 illustrates the assignment graph.

Note that now two different graphs are connected to the problem: (i) the assignment graph $\mathcal{G}_a$, representing the agent-task assignment, and (ii) the communication graph $\mathcal{G}_c$ representing the inter agent communication structure. According to the notation introduced for the Distributed Simplex algorithm, we will use in the following Roman letters, e.g. i, to index the agents, and Greek letters, e.g. κ, to index the tasks. The edges of the assignment graph $(i, \kappa) \in E_a$ are weighted with the cost coefficients $c_{i\kappa} \in \mathbb{Z}_{>0}$.

For each agent i and task κ a binary decision variable $x_{i\kappa} \in \{0, 1\}$ is introduced, which is 1 if agent i is assigned to task κ and 0 otherwise. The assignment problem corresponds now to the optimization problem $\min_{x_{i\kappa}\in\{0,1\}} \sum_i \sum_\kappa c_{i\kappa} x_{i\kappa}$, with the constraints that a full assignment is achieved. It is well known in the literature that the convex relaxation of the previous problem gives rise to the linear optimization problem [15], [10]:

$$\min \sum_i \sum_\kappa c_{i\kappa} x_{i\kappa} \tag{9}$$

$$\text{s.t.} \sum_\kappa x_{i\kappa} = 1, \quad \forall i \in \{1, \dots, N\}, \tag{10}$$

$$\sum_i x_{i\kappa} = 1, \quad \forall \kappa \in \{1, \dots, N\}, \tag{11}$$

$$0 \leq x_{i\kappa}, \quad \forall (i, \kappa). \tag{12}$$

TABLE I
The Assignment Problem data as Distributed Linear Program.

$\mathrm{h}_{i\kappa}$:	edge of $\mathcal{G}_a$ connecting agent i and task κ
$\mathbb{H}$:	N^2 edges of $\mathcal{G}_a$
b :	vector of ones in $\mathbb{R}^{2N-1}$
$\mathbb{P}^{[i]}$:	N edges of $\mathcal{G}_a$ connecting agent i with all N tasks
$\mathcal{G}_c$:	partially asynchronous communication network among agents
$\mathbb{B}$:	spanning tree of $\mathcal{G}_a$, with $2N-1$ arcs.

This problem has always an optimal solution $x_{i\kappa} \in \{0, 1\}$, corresponding to the optimal assignment. Note that the linear program (9)-(12) has $n = N^2$ decision variables and $d = 2N$ equality constraints. The problem is fully determined by a subset of $d = 2N - 1$ equality constraints, and one constraint can be ignored. Obviously, an edge of the assignment graph $\mathcal{G}_a$ is uniquely characterized by the triplet of integers

$$\mathrm{h}_{i\kappa} := (c_{i\kappa}, i, \kappa) \tag{13}$$

which is the cost coefficient $c_{i\kappa}$ and the corresponding identifier of an agent i and a task κ. Table I shows how the assignment problem can be represented as a distributed linear program.

The decision variables $x_{i\kappa}$ in (9)-(12) determine whether the edge $h_{i\kappa}$ will be active, and therefore whether agent i will be assigned to task κ. As previously seen, the optimal solution x^* must have exactly N entries equal to 1, since this provides a full assignment. However, since a basis solution contains always $2N-1$ decision variables, the assignment problem is inherently primal degenerate. Very often (although not inherently) the assignment problem is also dual degenerate. Consider for example the extreme, but not unrealistic, situation where $c_{i\kappa} = 1$, for all i, κ. Such a multiplicity of optimal solution imposes a severe challenge in the distributed assignment problem. It is not sufficient for an agent to have locally available an optimal solution, but it is critically necessary that all agents have the same optimal solution available. The Distributed Simplex algorithm is perfectly suited to solve the distributed assignment problem. Next, we provide a bound on the number of bytes that agents need to transmit for the distributed assignment problem.

Proposition 5.1: At each communication instant, every agent transmits at most $\mathcal{O}(N \log_2 N)$ bytes.

Proof: Assume that the integers $c_{i\kappa}$ can be encoded with 2 bytes. Let the ceil operator $\lceil r \rceil$ indicate a rounding of r to the next larger integer. Then a column can be encoded with $2 + \lceil \frac{1}{4}(\log_2(N) + 1) \rceil$ bytes. Thus, at each round, at max $(2N - 1) \cdot (2 + \lceil \frac{1}{4}(\log_2(N) + 1) \rceil)$ bytes are sent out by each agent. ■

With an increasing number of agents N in the network, the number of bytes that has to be exchanged grows only with $\mathcal{O}(N \log_2 N)$. In contrast, the number of decision variables n of the centralized assignment problem (9) grows with $\mathcal{O}(N^2)$. Therefore, while the original problem grows quadratically, the local complexity of the Distributed Simplex algorithm increases only almost linearly.

We use distributed assignment problems to analyze by simulation the expected time complexity of the Distributed Simplex algorithm. For each simulation, we generate a random assignment problem by choosing the cost coefficients $c_{i\kappa}$ uniformly from the interval $[0, 20]$. This allows a randomized analysis of the expected convergence time of the Distributed Simplex algorithm.

We use the following setup: throughout all scenarios we consider $N = 40$ agents. Therefore, the centralized assignment problem has $n = 1600$ columns. For simulation purposes, we consider a

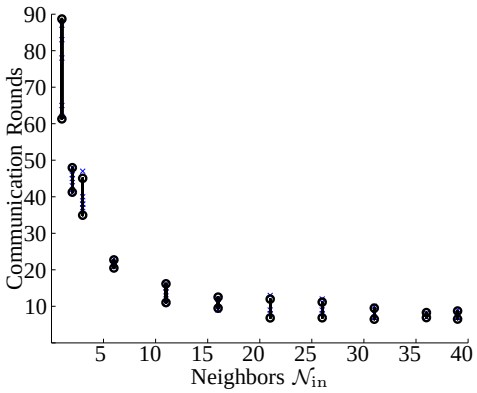

Fig. 2. Communication rounds in dependence of the number of in-neighbors on the graph. The 'x' indicate the completion time and the black bars indicate the 95% confidence intervals for the expected number of communication rounds.

synchronous communication network, where all agents transmit their information simultaneously. We call a time interval with information exchange and local computations a communication round. We assume that all agents are arranged in a *ring* structure. Each agents communicates one-directional to a specified number of agents, following him in the ring graph. We will vary the number of out-neighbors, which is in this setup equivalent to the number of in-neighbors, in order to analyze the time complexity in dependence of the graph parameters *number of in-neighbors* and *diameter* of the graph.

For a fixed number of neighbors, we simulate five random assignment problems, and compute based on this information the 95% confidence intervals for the *expected time complexity*. For computing the confidence intervals, we assume that the convergence time of the random problems is normally distributed. The simulation results in dependence of the number of in-neighbors $\mathcal{N}_{\text{in}}$ and of the graph diameter $\text{diam}(\mathcal{G}_c)$ are shown in the Figures 2 and 3.

In Figure 2, which shows the required communication rounds with respect to the in-neighbors, one can see that the time complexity first quickly decreases as the number of in-neighbors grows, but then saturates to a certain level. From Figure 3 the diameter of the communication graph $\mathcal{G}_c$ appeared to be significantly more important for the time complexity of the algorithm than the number of in-neighbors $\mathcal{N}_{\text{in}}$. However, the number of in-neighbors directly

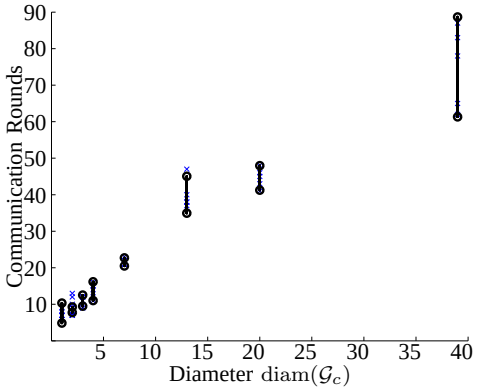

Fig. 3. Communication rounds in dependence of the diameter of the graph. The 'x' indicate the simulation results and the black bars the 95% confidence intervals for the expected communication rounds.

determines the dimension of the local subproblems. While the subproblems each agent has to solve are fairly small for few in-neighbors, they can become very large, and in the limit even the centralized problem, as the number of neighbors increases. We can conclude from the Figures 2 and 3, that, in the given setup, there is an optimal communication graph with around 15 in-neighbors, where the expected number of communication rounds is almost minimal, while the local subproblems are still of a moderate size.

VI. Conclusions

We have proposed the notion of *Distributed Linear Programs*, as an extension of linear programs to a multi-agent setup. By utilizing a non-standard version of the classical simplex algorithm, a distributed algorithm, named Distributed Simplex has been derived for solving Distributed Linear Programs. The algorithm is proven to work in asynchronous networks and poses little requirements on the communication structure. Additionally, we have derived a duality relation of the new algorithm to the recently proposed Constraints Consensus algorithm. Finally, the multi-agent assignment problem has been introduced exemplary as a relevant problem class for which the algorithm is especially well suited. The multi-agent assignment problem is also used for a randomized analysis of the expected time complexity of the proposed algorithm. The simulation results indicate, that the time complexity grows at least linearly with the diameter of the communication network.

References

[1] G. Notarstefano and F. Bullo, "Distributed abstract optimization via constraints consensus: Theory and applications," *IEEE Transactions on Automatic Control*, 2011, accepted. [Online]. Available: http://motion.me.ucsb.edu/pdf/2007z-nb.pdf

[2] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computations: Numerical Methods*. Belmont, Massachusetts: Athena Scientific, 1997.

[3] A. Nedic, A. Ozdaglar, and P. A. Parrilo, "Constrained consensus and optimization in multi-agent networks," *IEEE Transactions on Automatic Control*, vol. 55, no. 4, pp. 922–938, 2010.

[4] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[5] M. Zhu and S. Martínez, "On distributed convex optimization under inequality and equality constraints via primal-dual subgradient methods," *IEEE Transactions on Automatic Control*, 2011, to appear. Available at http://arxiv.org/abs/1001.2612v1.

[6] B. Johansson, A. Speranzon, and M. Johansson, "On decentralized negotiations of optimal consensus," *Automatica*, vol. 14, no. 4, pp. 1175–1179, 2008.

[7] J. Hall and K. McKinnon, "ASYNPLEX, an asynchronous parallel revised simplex algorithms," *Annals of Operations Research*, vol. 81, no. 24, pp. 27–50, 1998.

[8] G. Yarmish and R. Slyke, "A distributed, scalable simplex method," *Journal of Supercomputing*, vol. 49, pp. 373–381, 2009.

[9] G. Notarstefano and F. Bullo, "Network abstract linear programming with application to minimum-time formation control," in *IEEE Conference on Decision and Control*, New Orleans, LA, December 2007, pp. 927–932.

[10] G. B. Dantzig, *Linear Programming and Extensions*. Princeton University Press, 1963.

[11] K. Fukuda, H.-J. Lüthi, and M. Namiki, "The existence of a short sequence of admissible pivots to an optimal basis in LP and LCP," *International Transactions in Operational Research*, vol. 4, no. 4, pp. 273–284, 1997.

[12] C. Jones, E. Kerrigan, and J. Maciejowski, "Lexicographic perturbation for multiparametric linear programming with applications to control," *Automatica*, vol. 43, pp. 1808–1816, 2007.

[13] F. Bullo, J. Cortés, and S. Martínez, *Distributed Control of Robotic Networks*, ser. Applied Mathematics Series. Princeton University Press, 2009. [Online]. Available: http://coordinationbook.info

[14] B. Gärtner and E. Welzl, *Linear Programming-Randomization and Abstract Frameworks*, ser. Lecture Notes in Computer Science. Springer, 1996, vol. 1046, ch. Symposium on Theoretical Aspects of Computer Science, pp. 669–687.

[15] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of Operations Research*, vol. 14, no. 1-4, pp. 105–123, 1989.