Prepared by F.L. Lewis
Updated: Saturday, October 22, 2016

5. Algorithms for Parallel Computations on Graphs

The purpose of the consensus algorithms on graphs we have discussed is to find control protocols that result in all nodes reaching the same state values while communication is restricted to local neighbor protocols based on the graph structure. In this section we show that the consensus algorithms are closely related to parallel processing algorithms. Parallel algorithms are important in solving high-dimensional problems with a large number of unknowns, and have been developed in numerical analysis and in computer science for applications to parallel processing computer architectures. They have objectives other than producing consensus.

We discuss in this section parallel algorithms for solution of linear equations, for finding the first left eigenvector of a stochastic matrix, and for solving for voltages in large-scale electrical networks.

Equation Section 5

Consensus Algorithms

Given a graph $G=(V,E)$ with $V=\{v_1,\cdots,v_N\}$ a set of N nodes, each having a state x_i, we have discussed decision protocols of the form

$$u_i=\sum_{j\in N_i}a_{ij}(x_j-x_i) \tag{5.1}$$

This is known as a *local voting protocol* since the control input of each node depends on the difference between its state and all its neighbors. The communication is restricted to be along the edges of the directed communication graph having nonzero edge weights a_{ij}.

We used this protocol in continuous-time systems

$$\dot{x}_i=u_i \tag{5.2}$$

with $x_i,u_i\in R$. Then,

$$\dot{x}_i=\sum_{j\in N_i}a_{ij}(x_j-x_i) \tag{5.3}$$

Here, the global dynamics in terms of vectors $x=[x_1\cdots x_N]^T\in R^N$, $u=[u_1\cdots u_N]^T\in R^N$ is given by

$$\dot{x}=-Lx \tag{5.4}$$

with L the graph Laplacian matrix.

We also studied the discrete-time dynamics

$$x_i(k+1)=x_i(k)+\frac{1}{1+d_i}u_i(k) \tag{5.5}$$

with $x_i,u_i\in R$, where protocol (5.1) normalized by dividing by d_i, the in-degree of node *i*. The closed-loop system becomes

$$x_i(k+1) = x_i(k) + \frac{1}{1+d_i}\sum_{j\in N_i} a_{ij}(x_j(k) - x_i(k)) = \frac{1}{1+d_i}\left(x_i(k) + \sum_{j\in N_i} a_{ij}x_j(k)\right) \tag{5.6}$$

The global dynamics is

$$x(k+1) = x(k) - (I+D)^{-1}Lx(k) = (I+D)^{-1}(I+A)x(k) \equiv Fx(k) \tag{5.7}$$

where with $D = diag\{d_i\}$ and F is the normalized discrete-time graph matrix.

It was shown that, if the graph has a spanning tree, in these scenarios all nodes reach a consensus value so that $x_i(t) \to c, \forall i$ for some consensus value c.

Parallel Processing Solutions for Systems of Linear Equations

This material is from [Bertsekas and Tsitsiklis 1997]. Consider the system of linear equations

$$Ax = b \tag{5.8}$$

with matrix $A = [a_{ij}] \in R^{N\times N}$ and vector $b = [b_i] \in R^N$ given and $x = [x_1 \cdots x_N]^T \in R^N$ a vector of unknowns. Assume that A is invertible so there exists a unique solution x. The obvious way to solve this equation is to invert matrix A. However, if N is large and A is ill-conditioned, this can cause problems.

Write the system of equations as

$$\sum_{j=1}^{N} a_{ij}x_j = b_i, \quad i = 1, \cdots, N \tag{5.9}$$

Assume that diagonal elements $a_{ii} \neq 0$ and write

$$x_i = -\frac{1}{a_{ii}}\left[\left(\sum_{j\neq i} a_{ij}x_j\right) - b_i\right] \tag{5.10}$$

This form suggests many methods for parallel solution of (5.8).

Jacobi Algorithm

Start with some initial guess vector $x(0) \in R^N$ and perform the Jacobi iteration at nodes i

$$x_i(k+1) = -\frac{1}{a_{ii}}\left[\left(\sum_{j\neq i} a_{ij}x_j(k)\right) - b_i\right] \tag{5.11}$$

If this sequence converges, then clearly (5.10) holds.

The Jacobi algorithm is very close to consensus algorithm (5.6), except that in the latter one averages in also the previous state $x_i(k)$ and sets $b_i = 0, \forall i$. The consensus algorithm seeks to solve the set of equations

$$x_i = \frac{1}{1+d_i}\left(x_i + \sum_{j\in N_i} a_{ij}x_j\right) \tag{5.12}$$

or, written globally

$$(I+D)x = (I+A)x \tag{5.13}$$

It has been seen that this special set of linear equations guarantees that consensus is reached.

Note that to compute the Jacobi update (5.11), each node needs to know the value of $x_j(k)$ for all nodes j for which $a_{ij} \neq 0$. Thus, row i of matrix A implies the equivalent of a local neighborhood set for node i. If matrix A is sparse, the Jacobi algorithm is far more efficient than brute force inversion of matrix A.

Gauss-Seidel Algorithm

Start with some initial guess vector $x(0) \in R^N$ perform the iteration

$$x_i(k+1) = -\frac{1}{a_{ii}}\left[\left(\sum_{j<i} a_{ij}x_j(k+1) + \sum_{j>i} a_{ij}x_j(k)\right) - b_i\right] \tag{5.14}$$

If this sequence converges, then (5.10) holds. In the Gauss Siedel algorithm, one uses more up-to-date information than in the Jacobi algorithm, which always uses old estimates from iteration k for the update at node i. Gauss-Seidel considers matrix A as the sum of an upper triangular matrix and a lower triangular matrix. In terms of this decomposition, the algorithm is causal and the information needed for the update information of node i at iteration $k+1$ has already been computed prior to its update.

The Jacobi and Gauss-Seidel methods are often referred to as *relaxation methods*. In relaxation algorithms, each node i starts with an initial guess at the solution and slowly relaxes it towards the true solution by taking onto account information computed by its neighbors, reducing the errors as it does so.

Over-Relaxation Algorithms

A variety of related methods offer improved convergence or faster convergence. The Jacobi over-relaxation method performs the iterations

$$x_i(k+1) = (1-\alpha)x_i(k) - \frac{\alpha}{a_{ii}}\left[\left(\sum_{j\neq i} a_{ij}x_j(k)\right) - b_i\right] \tag{5.15}$$

where α is known as a relaxation parameter. Usually, $0 < \alpha \leq 1$. This algorithm does not perform the complete update (5.11) but only moves towards it by an amount that also takes into account the previous value of $x_i(k)$. A method known as Successive Over-Relaxation is similarly defined in terms of Gauss Seidel iterations.

Convergence of Parallel Algorithms

We have seen that the continuous-time protocol (5.3), (5.4) converges if and only if the graph has a spanning tree, for then the Laplacian L has a simple eigenvalue at $\lambda_1 = 0$ with all other eigenvalues having positive real-parts. Since L has all row sums equal to zero, a straightforward analysis shows that consensus is reached. A similar study shows that the discrete-time protocol (5.6) delivers consensus under the same conditions.

Let $\Delta = diag\{a_{ii}\}$ and write $A = \Delta + B$. Then the Jacobi algorithm solves at each step

$$x(k+1) = -\Delta^{-1}Bx(k) + \Delta^{-1}b \tag{5.16}$$

Let A be invertible. Then, this algorithm converges if and only if $\Delta^{-1}B$ has all eigenvalues inside the unit circle. Moreover, then it converges to the solution of (5.8) for all initial values $x(0)$. Under the condition, (5.16) defines a contraction map.

This convergence criterion is not often useful because the eigenvalues of $\Delta^{-1}B$ are difficult to determine if N is large. Let matrix A be strictly diagonally dominant, that is

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \quad \forall i \tag{5.17}$$

Then, $\Delta^{-1}B$ has all absolute row sums less than one and all diagonal elements equal to zero. According to the Gerschgorin criterion, all its eigenvalues are within a union of discs having centers at zero and radii less than 1. Therefore, all eigenvalues of $\Delta^{-1}B$ are strictly inside the unit circle.

We have shown that if matrix A is strictly diagonally dominant, then the Jacobi iterations converge to the solution to (5.8).

Parallel Solution of Partial Differential Equations

Consider Poisson's equation in two dimensions

$$\Delta V(x,y) = \nabla^2 V(x,y) = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) V(x,y) = f(x,y), \tag{5.18}$$

where $\Delta = \nabla^2$ is the Laplacian operator and ∇ the gradient. Function $f(x,y)$ is a forcing function, often specified on the boundary of a region. Discretizing the equation on a uniform mesh as shown in Figure 5-1 with grid size h in the *(x,y)* plane one writes in terms of the forward difference

$$\frac{\partial V(x,y)}{\partial x} = \frac{1}{h}\big(V(x+h,y) - V(x,y)\big) + O(h),$$

$$\frac{\partial^2 V(x,y)}{\partial x^2} = \frac{1}{h^2}\big(V(x+h,y) + V(x-h,y) - 2V(x,y)\big) + O(h^2),$$

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) V(x,y) = \frac{1}{h^2}\big(V(x+h,y) + V(x-h,y) + V(x,y+h) + V(x,y-h) - 4V(x,y)\big) \approx f(x,y).$$

Indexing the *(x,y)* positions with an index i and denoting $X_i = (x_i, y_i)$ one has approximately to order h^2

$$\frac{1}{h^2}\big(V(X_{i,R}) + V(X_{i,L}) + V(X_{i,u}) + V(X_{i,D}) - 4V(X_i)\big) = f(X_i), \tag{5.19}$$

for states not on the boundary and a similar equation for boundary states. $X_{i,R}$ denotes the *(x,y)* location of the state to the right of X_i, and similarly for states to the left, up, and down relative to X_i.

This is a set of N simultaneous equations in the form of (5.8). The Jacobi algorithm performs the iterations

$$V_{k+1}(X_i) = \tfrac{1}{4}V_k(X_{i,U}) + \tfrac{1}{4}V_k(X_{i,R}) + \tfrac{1}{4}V_k(X_{i,D}) + \tfrac{1}{4}V_k(X_{i,L}) - \tfrac{h^2}{4} f(X_i). \tag{5.20}$$

These updates may be done for all nodes or states simultaneously. Then, this is the *relaxation method* for numerical solution of Poisson's equation. The relaxation algorithm converges, but may do so slowly. Variants have been developed to speed it up.

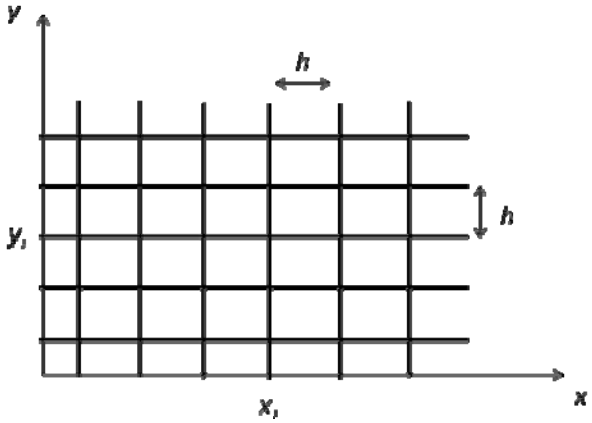


Figure 5-1. Shows sampling Poisson's equation in the *(x,y)* plane with a uniform mesh of size *h*.

■

Parallel Computation of First Left Eigenvectors

This material is from [Bertsekas and Tsitsiklis 1997]. Consider a nonnegative row-stochastic matrix $F = [f_{ij}]$. F has an eigenvector at $\lambda_1 = 1$ with associated right eigenvector $v_1 = \underline{1}$, the vector of 1's, because $F\underline{1} = \underline{1}$. Any first left eigenvector of F for $\lambda_1 = 1$ satisfies $w_1^T F = w_1^T$. If F is the discrete-time graph matrix in (5.7), we have seen the importance of w_1 in determining the consensus value reached if the graph has a spanning tree. Let us now see how to estimate the first left eigenvector of a stochastic matrix using a parallel processing algorithm.

Define the elements of the first left eigenvector as $w_1 = \begin{bmatrix} p_1 & p_2 & \cdots & p_N \end{bmatrix}^T$ and consider the parallel algorithm at each node i

$$p_i(k+1) = \sum_{j=1}^{N} f_{ji} p_j(k), \quad i = 1, \cdots, N \tag{5.21}$$

This can be written as

$$w^T(k+1) = w^T(k)F \tag{5.22}$$

with $w(k) = \begin{bmatrix} p_1(k) & p_2(k) & \cdots & p_N(k) \end{bmatrix}^T$.

To study convergence, we have the next background result.

Lemma 1. [Ren and Beard 2008]. Given a graph $G = (V, E)$ with N nodes, let $F = [f_{ij}] \in R^{N \times N}$ be nonnegative and row stochastic, and satisfy $f_{ij} > 0 \Leftrightarrow e_{ji} \in E$. Then $F\underline{1} = \underline{1}$, that is $\lambda_1 = 1$ is an eigenvalue of F with right eigenvector equal to $\underline{1}$. Moreover, $\lambda_1 = 1$ is simple, that is, it has

multiplicity equal to 1, if and only if graph G has a spanning tree. Furthermore, if $f_{ii} > 0, \forall i$, all other eigenvalues of F have magnitude less than 1. ■

Now we have the next convergence result.

Lemma 2. Let $F = [f_{ij}] \in R^{N \times N}$ be nonnegative and row stochastic with $f_{ii} > 0, \forall i$ and the associated graph have a spanning tree. Then, for all initial conditions $p_i(0),\ i = 1, \cdots, N$ the algorithm (5.21) converges to a first left eigenvector $w_1 = \begin{bmatrix} p_1 & p_2 & \cdots & p_N \end{bmatrix}^T$ of F. Moreover, if the initial conditions satisfy $\sum_i p_i(0) = 1$, then this left eigenvector is normalized so that $w_1^T \underline{1} = 1$.

Proof: Under the hypothesis one has $\lambda_1 = 1$ and all other eigenvalues inside the unit circle so that the Jordan form J yields

$$F = MJM^{-1} = \begin{bmatrix} \underline{1} & v_2 & \cdots \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & stable \end{bmatrix} \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \end{bmatrix} \tag{5.23}$$

where $w_1^T \underline{1} = 1$. Thus,

$$F^k = MJM^{-1} = \begin{bmatrix} \underline{1} & v_2 & \cdots \end{bmatrix} \begin{bmatrix} 1^k & 0 \\ 0 & stable \end{bmatrix} \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \end{bmatrix} \tag{5.24}$$

Therefore,

$$F^k = \underline{1} w_1^T + decaying\ terms \to \underline{1} w_1^T \tag{5.25}$$

and F is SIA or ergodic.

The solution to (5.22) is $w^T(k) = w^T(0) F^k$ so that

$$\lim_{k \to \infty} w^T(k) = \lim_{k \to \infty} w^T(0) F^k = \sum_i p_i(0) w_1^T \tag{5.26}$$

which is a left eigenvector of F for $\lambda_1 = 1$. ■

If the graph is strongly connected, then it has a spanning tree and the algorithm converges.

In the parallel algorithm (5.21), the summation is over a column of F. Therefore, each node i requires information from its out-neighbors, that is from those nodes j such that there is an edge going from node i to node j. Also required is for each node i to know the elements f_{ji} in column j, that is the weights of the outgoing edges. By contrast, the consensus protocols, such as (5.1) require node i to have information about the state of its in-neighbors, along with the weighting elements a_{ij} in row j of graph adjacency matrix A, that is the weights of the incoming edges.

Parallel Solution of Large-Scale Electrical Networks

Consider a large-scale electrical network of interconnected resistors with voltage sources connected to some of the nodes. It is desired to find the induced voltage at every node in the network using local parallel processing protocols where computations are carried out at each node using only information about its neighbors. This can be accomplished by modeling the network as a graph and using the local protocol methods studied in this chapter.

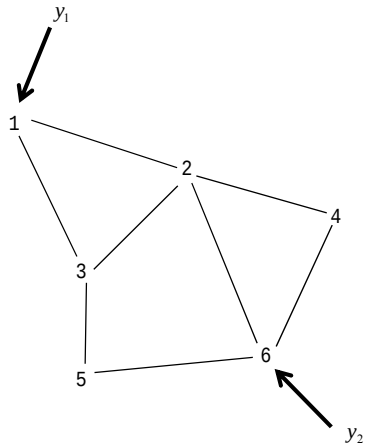


Figure 2. Undirected graph with two voltage sources.

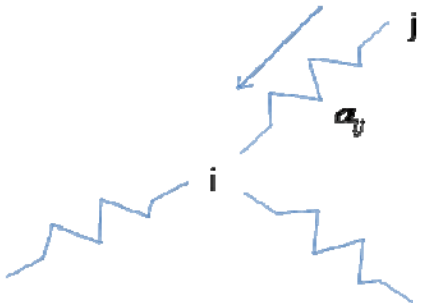


Figure 3. Electrical network: a graph whose edge weights are conductances.

Consider an undirected graph $G=(V,E)$ with $V=\{v_1,\cdots,v_N\}$ a set of N nodes, each having a state x_i defined as the voltage at that node. Let there be K root or source nodes $y_k, k=1,\cdots,K$, each with a directed edge into one node in the graph as shown in Figure 2. These nodes are the root nodes of K spanning trees whose union spans all the nodes in the graph.

Examine Figure 3, where current flows into node i due to voltage differences (x_j-x_i), with $j\in N_i$ its neighbors that have edges in common with node i. Define the edge weight as the conductance $a_{ij}>0$ on the path between node j and node i. Then according to Kirchhoff's current law one has

$$\dot{x}_i=\sum_{j\in N_i} a_{ij}(x_j-x_i)+b_{ik}(y_k-x_i) \qquad (5.27)$$

where the right-hand side is the sum of the currents flowing into node i. It has been considered that a unit capacitor is connected from each node to ground and the element flow equation $C\dot{v}=I$ is used to relate net current I, voltage v, and capacitance $C=1$ at each node. In this equation, some nodes have external voltage sources $y_k, k=1,\cdots,K$ delivering current directly to them through links with conductances $b_{ik}>0$.

Equation (5.27) is interpreted as a local update law for node i in terms of its own voltage and that of its neighbors. It has the form of the consensus protocol (5.3) plus some input terms into those nodes connected to external voltage sources. A similar consensus protocol with only one source has been called consensus with a leader [Jadbabaie et al. 2003], or pinning control [Li et al. 2004]. We shall study this protocol extensively in later chapters.

To write the global dynamics, define the global state $x = [x_1 \cdots x_N]^T \in R^N$, and the vector of voltage sources $y = [y_1 \cdots y_K]^T \in R^K$. Define the matrix $b = [b_{ik}] \in R^{N \times K}$ and the diagonal matrices

$$B_k = \begin{bmatrix} b_{1k} & & & \\ & b_{2k} & & \\ & & \ddots & \\ & & & b_{Nk} \end{bmatrix} \tag{5.28}$$

Then, according to (5.27) one writes

$$\dot{x}_i = \sum_{j \in N_i} a_{ij} x_j - d_i x_i - b_{ik} x_i + b_{ik} y_{ki} \tag{5.29}$$

or

$$\dot{x} = -(D + \sum_k B_k - A)x + \sum_k B_k \underline{1} y_k \tag{5.30}$$

$$\dot{x} = -(L + \sum_k B_k)x + \sum_k B_k \underline{1} y_k \tag{5.31}$$

with $D = diag\{d_i\}$ the diagonal matrix of in-degrees and $L = D - A$ the graph Laplacian matrix. This can also be written as

$$\dot{x} = -(L + \sum_k B_k)x + by \tag{5.32}$$

If this system is stable, then at steady-state one has

$$0 = -(L + \sum_k B_k)x + by \tag{5.33}$$

which is Kirchhoff's current law- the sum of currents into each node is equal to zero. This implies that

$$x = (L + \sum_k B_k)^{-1} by \tag{5.34}$$

if the matrix is invertible. These are the equations of a voltage divider where the elements of vector *x* are the voltages at the nodes. It is not required to use these equations, however, because the individual node protocols (5.27) converge to the node voltages under certain conditions.

The next results detail the convergence properties of the parallel protocols (5.27).

Lemma 1. Let the graph of *L* be connected and at least one gain $b_{ik} > 0$. Then matrix $(L + \sum_k B_k)$ has all eigenvalues with real parts greater than zero.

Proof: Appendix 1a. ■

This result shows that if the network is connected, and at least one node has an external voltage source, then $-(L + \sum_k B_k)$ is asymptotically stable so that (5.32) converges to a steady-state value, and invertible, so that (5.34) holds.

Lemma 2. Let the graph of *L* be connected with only one voltage source y_1. Then the voltages

of all nodes reach a consensus voltage equal to y_1.

Proof: Consider (5.30) and note that $L\underline{1} = 0$. Then

$$\dot{x} = -(L + B_1)x + B_1\underline{1}y_k = -(L + B_1)x + (L + B_1)\underline{1}y_k = -(L + B_1)(x - \underline{1}y_k)$$

According to Lemma 1, one has at steady-state

$$0 = -(L + B_1)(x - \underline{1}y_k)$$

Now, $(L + B_1)$ nonsingular implies that $x - \underline{1}y_k$, or $x_i = y_1, \forall i$. ■

Exercise 1. Kirchhoff's current law on directed graphs.

If the graph is undirected, then according to the protocol (5.27) at steady-state Kirchhoff's current law is satisfied- the sum of currents into each node is equal to zero.

Algorithm (5.27) can also be performed if the graph is directed as shown in Fig. 4. On directed graphs, according to (5.27) the sum of currents into each node is zero at steady state. Nothing can be directly inferred about the currents going out of the nodes. Show that this implies that at steady state the sum of all currents at each node is equal to zero, that is, Kirchhoff's current law holds at each node.

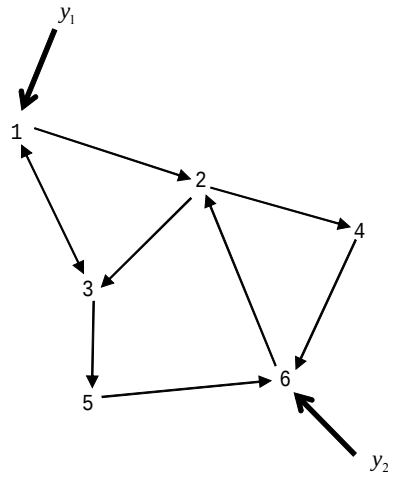


Figure 4. Directed graph with two voltage sources.

Bellman-Ford Algorithm

The Bellman-Ford Algorithm computes the minimum distance from all nodes in a graph to a root or source node. The shortest path from each node to the root can also be computed. This is all accomplished in a distributed fashion using only information from neighbor nodes. The Bellman-Ford Algorithm is performed causally forward in time. This is in contrast to dynamic programming, which solves the same problem by proceeding backwards in time.

The Bellman-Ford algorithm is given by

$$x_i(k+1) = min\left(x_i(k), \underset{j \in N_i}{min}\left(a_{ij} + x_j(k)\right)\right) \tag{5.35}$$

where $x_i(k)$ is a scalar state stored at each node. To find the path with the minimum number of edges from each node *i* to the root, all edge weights a_{ij} are set equal to 1 (if there is a directed

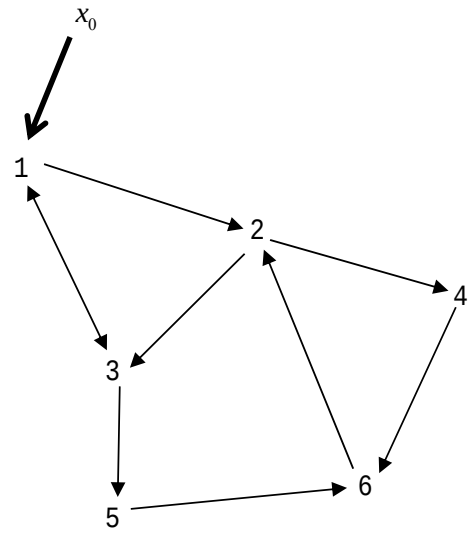


Figure 4. Bellman-Ford Algorithm finds the shortest distance from all nodes to a root node y_1.

path from node *j* to node *i*). All states of all nodes are initialized at infinity (i.e. large values) except for that of the root node, which is initialized at zero. The root node follows the protocol

$$x_0(k+1) = x_0(k) \tag{5.36}$$

The shortest path information sweeps through the network as time index *k* increases. At time *k=0*, the root node has a state value of *0* and all other nodes' states are large. At time *k=1*, the states of the direct out-neighbors of the root are set to 1 and the states of the remaining nodes do not change. At time *k=2*, the states of the out-neighbors of the root do not change, the states of the second-order out-neighbors are set to 2, and the states of the remaining nodes do not change. This continues until the states of all nodes store the minimum distance to the root node. After that, the states do not change further.

The shortest path from each node to the root can also be computed by adding one additional state $z_i(k)$ at each node. This state keeps track of which neighbor node was last used to update the shortest distance state $x_i(k)$

$$z_i(k+1) = \arg\min_{j \in N_i} \left(x_i(k), \min_{j \in N_i} \left(a_{ij} + x_j(k) \right) \right), \quad \text{if } \min_{j \in N_i} \left(a_{ij} + x_j(k) \right) < x_i(k) \tag{5.37}$$

Using this information stored at each node *i*, one can traverse through the graph along the shortest path to the root node by interrogating each node along the way about the direction to follow to move the shortest distance to the root.

The Bellman-Ford Algorithm stores two states at each node: the minimum distance $x_i(k)$ to the root and the direction $z_i(k)$ to follow to move along the shortest path. This is a

form of Reinforcement Learning known as Q-Learning. In Q-learning, the optimum value of a performance index is stored at each node, as well as the action to follow to attain that optimum value. This information is stored in the graph in the form of a lookup table.

The Bellman-Ford Algorithm can be used to find minimum-cost paths by assigning weights a_{ij} corresponding to the cost required to move along edge *(j,i).* The cost can correspond to time, fuel, geographic distance, etc. required to move along the edge. The algorithm works if there are cycles in the graph, but not if there are cycles with negative weights.

References

D. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Athena Scientific, Belmont, MA, 1997.

A. Jadbabaie, J. Lin, and S. Morse. 2003. "Coordination of Groups of Mobile Autonomous Agents Using Nearest Neighbor Rules," *IEEE Trans. Automatic Control* 48, no. 6: 988-1001.

X. Li, X. Wang, and G. Chen, "Pinning a complex dynamical network to its equilibrium," *IEEE Trans. Circuits Syst. I, Reg. Papers,* vol. 51, no. 10, pp. 2074–2087, Oct. 2004.

W. Ren and R.W. Beard, *Distributed Consensus in Multi-Vehicle Cooperative Control,* Springer-Verlag, London, 2008.