s-t Paths Using the Min-Sum Algorithm

Nicholas Ruozzi[†]
Computer Science
Yale University
New Haven, CT 06520-8285, USA
Nicholas.Ruozzi@yale.edu

Sekhar Tatikonda
Electrical Engineering
Yale University
New Haven, CT 06520-8285, USA
Sekhar.Tatikonda@yale.edu

***Abstract*— Solving the distributed shortest path problem has important applications in the theory of distributed systems, most notably routing. In this paper, we provide and prove the convergence of a min-sum algorithm to compute the shortest path between two nodes in a graph with positive edge weights. Unlike the standard distributed shortest path algorithms, the rate of convergence depends on the weight of the minimal path and not necessarily the number of nodes in the network.**

I. Introduction

The use of the max-product and min-sum algorithms to solve combinatorial optimization problems has several advantages over the typical algorithms that are used to solve these problems:

1) Message passing algorithms can be easily converted into distributed algorithms.
2) These algorithms are easy to describe and implement.

The recent work on max-product and linear programming of [1], [2], and [3] has explored the connection between integer programs for the max-weight matching and the max-weight independent set problems and the convergence of max-product for these problems. These papers suggest that starting with any integer program, one can formulate a max-product or min-sum message passing scheme on a factor graph where the variables are the variables of the integer program and the factors correspond to the constraints of the integer program. Further, they demonstrate that the convergence of max-product depends on the solutions to the relaxation of the integer program.

The previous results suggest suprising connections between max-product and linear programs. However, these problems may not be sufficient to develop a general understanding of when message passing algorithms can be used to solve combinatorial optimization problems:

1) Max-weight independent set is known to be NP-complete which suggests that max-product is not likely to yield an efficient solution.
2) The constraints of the integer programs for these problems are all binary which simplifies max-product.

There are known easy classes of integer programming problems. In this paper, we explore the connections between integer programming and the min-sum algorithm by examining the behavior of min-sum on a simple optimization problem from one such easy class of integer programs: given a directed graph $G = (V, E)$, vertices s and $t \in V$, and weights $w_e > 0$ for each $e \in E$, *the shortest s-t path problem* is to find the path of minimum weight in G starting at s and ending at t. If no such path exists the shortest s-t path is infinite. This problem is related to routing and has important applications in distributed systems.

A. Total Unimodularity

***Definition** 1.1:* A matrix A is totally unimodular if every square sub-matrix has determinant 0, 1, or -1. Note that this implies that the entries of A are 0, 1, or -1.

***Theorem** 1.2:* Let A be a totally unimodular $m \times n$ matrix and b an integral n vector. The polyhedron $P = \{x | Ax \leq b\}$ is integral.

The theorem implies that if an integer programming problem $Ax \leq b$ with A a totally unimodular matrix and b an integral vector b has a unique solution then the linear programming problem obtained by relaxing the requirement that the variables in the integer program can only take integer values also has a unique solution.

For an instance of the weighted matching problem and the weighted independent set problems, the corresponding integer program is not necessarily totally unimodular. In these cases, a unique integral solution to the IP does not guarantee a unique solution to the linear relaxation. This complicates the proofs of convergence as evidenced in [1] and [3]. Therefore, with the hope of producing a general theory, totally unimodular matrices seem an appropriate starting point. More information on total unimodularity and linear programming can be found in [4].

In this paper, we will show that if the shortest s-t path problem has a unique solution then the min-sum algorithm always converges to the correct solution in a number of steps that depends on the weight of the shortest path and the weight of the second shortest path. This paper is organized as follows: in Section II we formulate the shortest path problem for both integer programming and the min-sum algorithm and in Section III we show that min-sum converges if the LP relaxation has a unique solution.

II. Shortest s-t Paths

The shortest s-t path problem can be formulated as an integer program:

[†] Supported by NSF grant 0534052

minimize:

$$\sum_{e \in E} w_e X_e$$

subject to:

$$\sum_{(u,v)\in E} X_{(u,v)} = \sum_{(v,u)\in E} X_{(v,u)} \text{ for } v \in V - \{s,t\} \quad (1)$$

$$\sum_{(s,u)\in E} X_{(s,u)} = 1 + \sum_{(u,s)\in E} X_{(u,s)} \quad (2)$$

$$\sum_{(u,t)\in E} X_{(u,t)} = 1 + \sum_{(t,u)\in E} X_{(t,u)} \quad (3)$$

$$X_e \in \{0,1\} \text{ for each } e \in E \quad (4)$$

This integer program can be relaxed into a linear program by changing the last constraint from $X_e \in \{0,1\}$ to $X_e \in [0,1]$. The matrix for this linear program is the combination of two copies of the $V \times E$ incidence matrix for G which is totally unimodular [4]. Hence, if there is a unique shortest path from s to t in G then the linear program has a unique optimal solution (by the previous theorem).

A. Shortest s-t Paths via Min-Sum

The above problem can be formulated as a problem on a factor graph and then solved using the min-sum algorithm. Here we will have variables X_e for each $e \in E$ and factors ψ_v for each $v \in V$. The factor ψ_v is a function depending on all edges incident to v which we will denote ∂v. ψ_v is an indicator fucntion for the constraints in the integer program at vertex v. Specifically, define ψ_v for $v \neq s,t$ as follows:

$$\psi_v(X_{\partial v}) = \begin{cases} 1 & \text{if equation (1) is satisfied at } v \\ 0 & \text{otherwise} \end{cases}$$

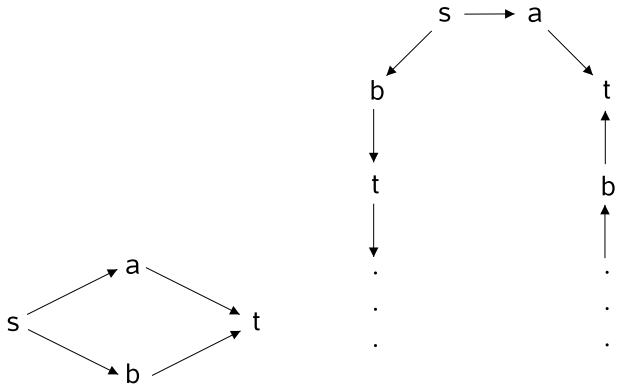
Similarly, ψ_s and ψ_t indicate whether or not (2) and (3) respectively are satisfied. The factor graph then has a vertex for each of the factors and variables with an edge joining a variable and a factor if the factor depends on that variable.

For each edge $e \in E$ define a self-potential $\phi_e(X_e) = e^{w_e X_e}$. We then define

$$f(X_E) = \sum_{e\in E} \log \phi_e(X_e) - \sum_{v \in V} \log \psi_v(X_{\partial v})$$

Given an assignment of each X_e to some value in $\{0,1\}$, $f(X_E)$ is equal to the $\sum_{e\in E} w_e X_e$ if the nonzero X_e's define a directed edge disjoint path from s to t in G and infinity otherwise. Therefore, minima of f correspond to minimal s-t paths. If no such path exists then f is infinity for all choices of X_E.

We can now use the min-sum message passing procedure in an attempt to minimize the objective function f. The message passing procedure is iterative in that at each stage a message $m_{(u,v)\to v}$ is sent from each variable $X_{(u,v)}$ to the factors ψ_v and ψ_u and a message $m_{v\to e}$ is sent from each factor ψ_v to each variable X_e such that e is incident to v in G. At any point, a variable $X_{(u,v)}$ can compute an estimate of whether or not it is in the minimum s-t path by using the messages that it received from ψ_u and ψ_v in the last time step. For the shortest path problem, the min-sum algroithm looks as follows:



(a) The graph G

(b) A few levels of the computation tree rooted at (s,a)

Fig. 1. A simple graph and computation tree.

Min-Sum Algorithm:

1) Initialize all messages to 0.
2) For e incident to v,

$$m^n_{v\to e}(x) = \min_{x_{\partial v}: x_e = x} -\log \psi_v(x_{\partial v}) + \sum_{e': e' \in \partial v - \{e\}} m^{n-1}_{e'\to v}(x_{e'})$$

3) For e incident to v and u,

$$m^n_{e\to v}(x) = \log \phi_e(x) + m^{n-1}_{u\to e}(x)$$

4) Compute the beliefs at step n:

$$b^n_e(x) = \phi_e(x) + \sum_{v\in\partial e} m^n_{v\to e}(x)$$

5) Estimate membership of $e = (u,v)$ in the min path as

$$\overline{x_e}^n = \begin{cases} 1 & \text{if } b^n_e(1) < b^n_e(0) \\ 0 & \text{if } b^n_e(0) < b^n_e(1) \\ ? & \text{otherwise} \end{cases}$$

We say that the message passing procedure has converged at step n if there is no time step $n' > n$ such that $\overline{x_e}^n \neq \overline{x_e}^{n'}$.

III. Convergence of Min-Sum

To prove convergence of the min-sum algorithm we will make extensive use of the notion of a computation tree. The computation tree rooted at a node y in the factor graph is constructed by starting at y and adding all neighbors of y as children of y in the tree. The next level of the tree is then generated by taking a leaf of the tree and adding all of the leaf's neighbors that are not its parent.

This process is the repeated for the new leaf nodes and so on. Notice that this computation tree has nodes for both edges and vertices in the original graph. We will, for ease of

illustration replace the X_e nodes with a directed edge joining e's two endpoints. We will denote this new tree starting with edge e and repeating the process for $2n$ steps as $T_e(n)$.

The resulting minimization problem on the computation tree is different from the original s-t path problem. There are now multiple copies of s and t making the problem on the computation tree a multi-source/multi-sink shortest path problem. Contrast this with the cases of max-weight matching and max-weight independent set where the problem on the original graph and the problem on the computation tree are exactly the same. Specifically, a feasible solution on the computation tree is a set of edges such that every copy of s in the tree has a path to a copy of t or a boundary node (allowed by the message initialization), every copy of t in the tree has a path from a copy of s or some boundary node, and every vertex has at least as many out edges as in edges. A feasible solution is minimal if no feasible solution has a smaller weight.

The computation tree models the message passing structure of the min-sum algorithm: minimal solutions on $T_e(n)$ correspond to the beliefs of the root obtained by running the min-sum algorithm for $2n$ steps.

Define $w(S) = \sum_{e \in S} w_e$ for S a set of edges. Let $w_{min} = \min_{e \in E} w_e$ and ϵ be the difference in weight of the second best s-t path in G and the optimal path in G then we have the following theorem:

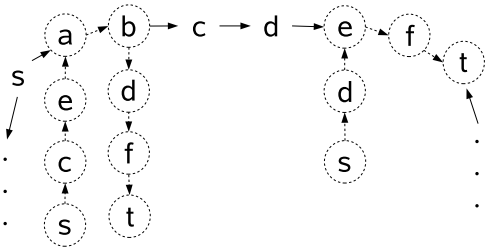
***Theorem** 3.1:* If P^* is the unique minimum s-t path in G then an edge $e \in E$ is in P^* iff every minimal solution on $T_e(n)$ contains the root for $n > \frac{w(P^*)^2}{\epsilon w_{min}} + \frac{w(P^*)}{w_{min}}$.

Proof: ($\Rightarrow$) Suppose by way of contradiction that $e = (u, v)$ is in the min s-t path P^* on G but that there is some minimal solution M on the computation tree rooted at e at time $2n$ that does not contain the root.

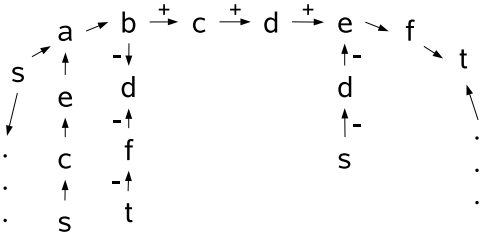
This proof, similar to that of [1], builds an alternating set of paths that can be swapped to improve the optimality of the solution. Because the constraints are not binary in our case, the construction is slightly more complicated.

Construct two subgraphs $M_{sub-opt}$ and M_{opt} of $T_e(n)$ as follows:

1) Let P be a copy of the minimum s-t path that uses the root edge $e = (u, v)$. Starting at v, follow P forward until doing so would require traversing an edge in M. Similarly, starting at u follow the edges in P backwards until doing so would require traversing an edge in M. Add this sub-path of P to M_{opt}.
2) By construction, for each sub-path P in M_{opt} not originating at a leaf, there must be at least one path in M that either (possibly both for the edge added in step 1)
   a) leaves the head of P and terminates in a copy of t or in a leaf of $T_e(n)$. If no such path is in $M_{sub-opt}$, choose such a path P' in M and follow it until t or the boundary. Add this sub-path to $M_{sub-opt}$.
   b) enters the tail of P and originates in a copy of s or in a leaf of $T_e(n)$. If no such path is in $M_{sub-opt}$, choose such a path P' in M and follow it backwards until s or the boundary. Add this sub-path to $M_{sub-opt}$.
3) By construction, for each sub-path P in $M_{sub-opt}$ not touching the boundary, there must be a copy of P^* that either
   a) leaves the head of P and terminates in a copy of t or in a leaf of $T_e(n)$. If this path is not in M_{opt}, follow it until t or when traversing an edge requires traversing an edge in M. Add this sub-path to M_{opt}.
   b) enters the tail of P and originates in a copy of s or in a leaf of $T_e(n)$. If this path is not in M_{opt}, follow it backwards until s or when traversing an edge requires traversing an edge in M. Add this sub-path to M_{opt}
4) Repeat steps 2 through 3 until it is no longer possible to add any more paths.



(a) Solution on the computation tree (dashed edges) rooted at (c, d) with optimal path $s - a - b - c - d - e - f - t$



(b) M_{opt} (+) edges and $M_{sub-opt}$ (-) edges

Fig. 2. Illustration of the construction.

This process builds a set of paths starting at the root that alternates between subpaths of copies of P^* and subpaths of the solution M. Figure 2 illustrates the construction for a graph G whose unique minimum s-t path is $(s, a), (a, b), (b, c), (c, d), (d, e), (e, f), (f, t)$.

Let $M^* = (M - M_{sub-opt}) \cup M_{opt}$. Notice that M^* is a feasible solution on $T_e(n)$ since the in degree and out degree of every node in M is preserved and, by the construction, sub-paths in M_{opt} satisfy the constraints at their heads and tails when added to M.

Let k be the number of disjoint sub-paths in M_{opt} and k' be the number of disjoint sub-paths in $M_{sub-opt}$. In the worst case, $k = k' + 1$ due to the alternating construction. As all other possible outcomes can be reduced to this situation, we only illustrate the proof in this instance. There are two cases:

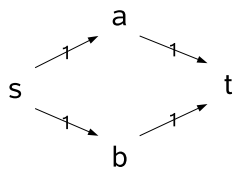


Fig. 3. Example of non-uniqueness.

Case 1: $k > \frac{w(P^*)}{\epsilon} + 1$
In this case there are $k-1$ disjoint sub-paths in $M_{sub-opt}$ each of which cannot have a weight closer than ϵ to the weight of the sub-path in M_{opt} that enters its tail. We have $(k-1)\epsilon > w(P^*)$, so

$$\begin{aligned} w(M^*) &= w(M) - w(M_{sub-opt}) + w(M_{opt}) \\ &< w(M) - (k-1)\epsilon + w(P^*) \\ &< w(M) \end{aligned}$$

Case 2: $k \leq \frac{w(P^*)}{\epsilon} + 1$

$$\begin{aligned} w(M^*) &= w(M) - w(M_{sub-opt}) + w(M_{opt}) \\ &\leq w(M) - (2x - k|P^*|)w_{min} + w(M_{opt}) \\ &\leq w(M) - (2x - \frac{kw(P^*)}{w_{min}})w_{min} + kw(P^*) \\ &\leq w(M) - 2xw_{min} + 2kw(P^*) \\ &< w(M) \end{aligned}$$

In either case, $w(M^*) < w(M)$ contradicting the minimality of M.

($\Leftarrow$) For the opposite direction, suppose by way of contradiction that every minimal solution on $T_e(n)$ contains the root edge e but that e is not in P^*. We can then construct $M_{sub-opt}$ and M_{opt} in an alternating fashion similar to the above. As the details of the proof are nearly identical, we omit them here. ■

A. Non-Uniqueness

If the shortest path is not unique, the min-sum algorithm may not produce the correct answer. For example, consider Figure 3. The s-t path is not unique, and by symmetry, the computation tree rooted at (s, a) is isomorphic to the computation tree rooted at (s, b). As a result, their beliefs at any fixed time will be the same.

IV. Conclusion and Future Work

We have demonstrated that the shortest s-t path problem can be solved by a min-sum procedure derived from an integer program. If the integer program has a unique solution, then the estimates produced by the min-sum algorithm can be used to find the min s-t path in G after $\frac{w(P^*)^2}{\epsilon w_{min}} + \frac{w(P^*)}{w_{min}}$ iterations of the algorithm. However, the above time bound may or may not be tight. A different or more careful proof may be able to reduce the time bound further.

Understanding the shortest path problem is an important first step in understanding how the max-product and min-sum algorithms can be used to solve totally unimodular linear programs. This problem highlights some complexities that are not present in the max-weight matching and max-weight independent set problems; most notably, the optimization problem on the computation tree is not the same as the original problem (recall that there were multiple s's and multiple t's on the computation tree). The result here suggests that similar techniques may have some success when applied to more general totally unimodular linear optimization problems.

References

[1] S. Sanghavi, D. Malioutov, and A. Willsky, "Linear programming analysis of loopy belief propagation for weighted matching," in *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. Cambridge, MA: MIT Press, 2008, pp. 1273–1280.

[2] M. Bayati, D. Shah, and M. Sharma, "Max-product for maximum weight matching: Convergence, correctness, and lp duality," in *Information Theory, IEEE Transactions on*, 2008, pp. 1241–1251.

[3] S. Sanghavi and D. Shah, "Tightness of lp via max-product belief propagation," 2005. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0508097

[4] A. Schrijver, *Theory of Linear and Integer Programming*. John Wiley & Sons Ltd., 1987.