BGP Convergence and Optimal Routing

Handouts: A guide to reading research papers.

Lecture Outline:

- The BGP convergence problem.
- Multicommodity flow problem formulations.
- Minimum delay routing.

1 The BGP convergence problem

Our presentation is based on the work of Griffin and Wilfong [GW99]. We are given an undirected graph $G = (V, E)$, and for each node in V a set of rules that assigns preferences to routes. Since the route calculations for different destinations do not interact, we restrict our attention to a fixed destination. Let the set of rules for node v be specified by a permitted set $\mathcal{P}_v$ and a ranking function λ_v.

One can model the BGP route exchange algorithm and the asynchronous nature of the network by the notion of the *activation of a node*: When a set A of nodes is activated, each node in A recomputes its most preferred route based on routes obtained from neighbors, $\mathcal{P}_v$, and λ_v.

At any instant, we can capture the state of the system by a tuple, which consists of the preferred routes for each node. We say that a state is *final* if any activation step leads back to the same state. In the initial state, only the destination has a route to itself; all other routes are empty.

There are several basic questions one can ask within the above stable paths problem framework.

- *Solvability:* Is there a sequence of activations that leads the initial state to a final state?
- *Reachability:* Given two nodes u and v, does there exist a final state in which u has a route to v.
- *k-robust:* Given a solvable system, will the system remain solvable under any possible failure of k links?

1.1 Examples of covergence and divergence

The instances of SPP can take several guises. In the following figures, we consider three examples. In Figure 1, we have a system that is solvable; that is, there exists a sequence of activations which lead the system to a stable state. Furthermore, it can be shown that there is a unique stable state. Figure 2 depicts a system that is unsolvable; every sequence of activations leads to an unstable state.

Figure 3 depicts a system that is also solvable but consists of multiple stable states, each reachable through a different sequence of activations. It is also interesting to note that if we consider the synchronous activation sequence in which both 1 and 2 are activated simultaneously in each step, we find that the system does not converge.

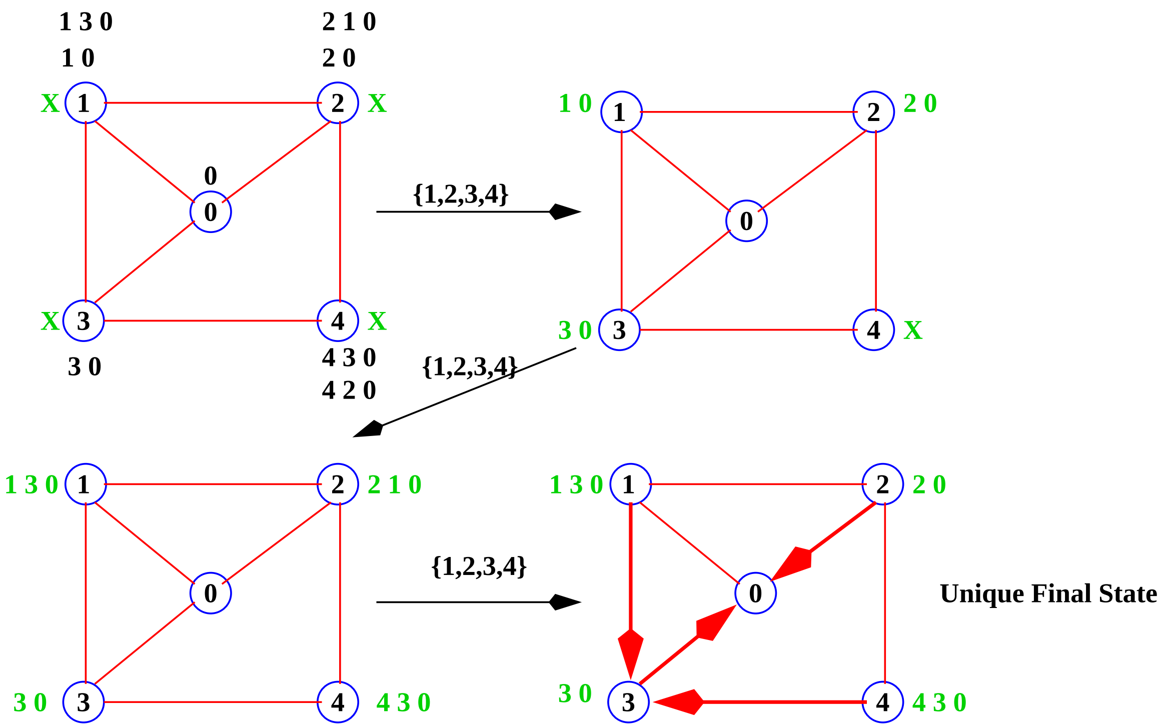

Figure 1: A solvable BGP system.

1.2 NP-completeness basics

Here we review some basics of polynomial-time computability and NP-completeness. For a more thorough discussion, we refer the reader to Chapter 34 of [CLR90] and [GJ79]. A canonical notion of a tractable problem is that of a problem that can be solved in polynomial time. We now review the classes P and NP. Consider the class of decision problems, that is, problems that require a yes or no answer. A decision problem Π is in P, if there exists an algorithm, which given any instance I of Π determines in time polynomial in the size of I whether I has a Y answer. A decision problem Π is in NP, if for every yes-instance I of Π there exists a *certificate* that can be verified in polynomial time.

A problem Π *p-time reduces* to Π' if there exists a poly-time computable function $f : I(\Pi) \mapsto I(\Pi')$ such that instance I of Π is a yes-instance of Π iff $f(I)$ is a yes-instance of Π'.

A problem Π is in NPC if (i) Π is in NP; and (ii) for every Π' in NP, Π' p-time reduces to Π. The implications of the NP-completeness of a problem are the following. If Π is solvable in polynomial time, then every problem in NP is solvable in polynomial time; in other words, $P = NP$. So Π is among the hardest problems in NP. All current evidence suggests that $P \neq NP$; so it is considered highly unlikely that a problem in NPC can be solved in polynomial time.

How do we prove that a problem is NP-complete? First, we prove that it is in NP. Second, we show that there exists an NP-complete problem Π' that p-time reduces to Π. For the second step, we can reduce a *known* NP-complete problem Π' to Π.

1.3 NP-completeness of SPP solvability

We first show that SPP solvability is in NP. This is easy, since one can quickly check whether a given state is stable. One can simply activate all nodes simultaneously. If any route changes,

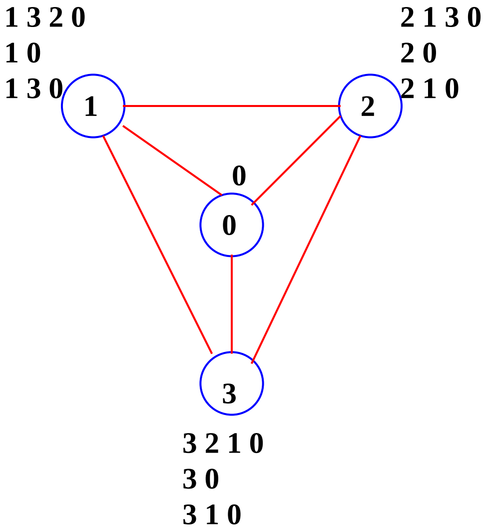

Figure 2: An unsolvable solvable BGP system.

then the state is not a final state; otherwise, the state is a final state. The total time taken is proportional to the $O(V + E + P)$, where P is the total number of policy rules.

For the second step of the NP-completeness proof, we reduce SAT, a well-known NP-complete problem, to SPP solvability. In SAT, we are given a Boolean formula F over n variables using $\neg$, $\vee$ and asked to determine whether there is a Boolean assignment that satisfies F. We assume that F is given in conjunctive normal form, that is a conjunction of disjunctive clauses.

The reduction from SAT to SPP solvability is depicted in Figure 5. Suppose we are given Boolean formula $C_1 \bigwedge C_2 \bigwedge \cdots \bigwedge C_m$ with n variables $x_1, \ldots, x_n$. Then, we construct the network for the SPP system as in Figure 5. Fix the destination node to be w. We now specify the policies at each node. For any i, node x_i (resp., $\neg x_i$) accepts any route to w that does not contain both x_{i-1} and $\neg x_{i-1}$. Furthermore, x_i (resp., $\neg x_i$) assigns highest preference to a route going through $\neg x_i$ (resp., x_i). For any i, node C_i only accepts routes that go through the literals in C_i. Finally, each of nodes 1, 2, and 3 have preferences among routes that go through 0 in the same way as in Figure 2. However, each of the 3 nodes have highest preference for the direct route through C_m.

We now show that F is satisfiable iff the above SPP system is solvable. Suppose F is satisfiable, and let L denote the set of literals that are set to true. Sort all the literals according to their index, in increasing order. (Note that no two literals in L have the same index since we cannot have x_i and $\neg x_i$ to be true simultaneously.) Now consider the sequence of activations $w, \ell_1, \neg \ell_1, \ell_2, \neg \ell_2$, and so on until $\ell_n, \neg \ell_n$, and then $C_1, C_2, \ldots, C_m$, and then $\{0, 1, 2, 3\}$. It is easy to see that we will obtain the following stable routing tree for the system. There will be a "spine" that goes from C_m through C_1 through all the literals in L. The negation of each literal ℓ in L will have a directed edge to the literal ℓ. Each of $\{1, 2, 3\}$ will have an edge to C_m and the node 0 will have an edge to w.

We now show the reverse direction. Suppose there is a sequence of activations that takes the system to a stable state σ. Then, by the nature of the subgraph consisting of $\{0, 1, 2, 3\}$, at least one of the nodes has a route to w through C_m (this is because the example in Figure 2 is unsolvable). This implies there is a route from C_m to w that satisfies the policies of all of the clauses. Furthermore, due to the preference policies of the literals, exactly one of literals x_i and $\neg x_i$ could be along this path. We set all of the literals along the path to be true. Owing to the

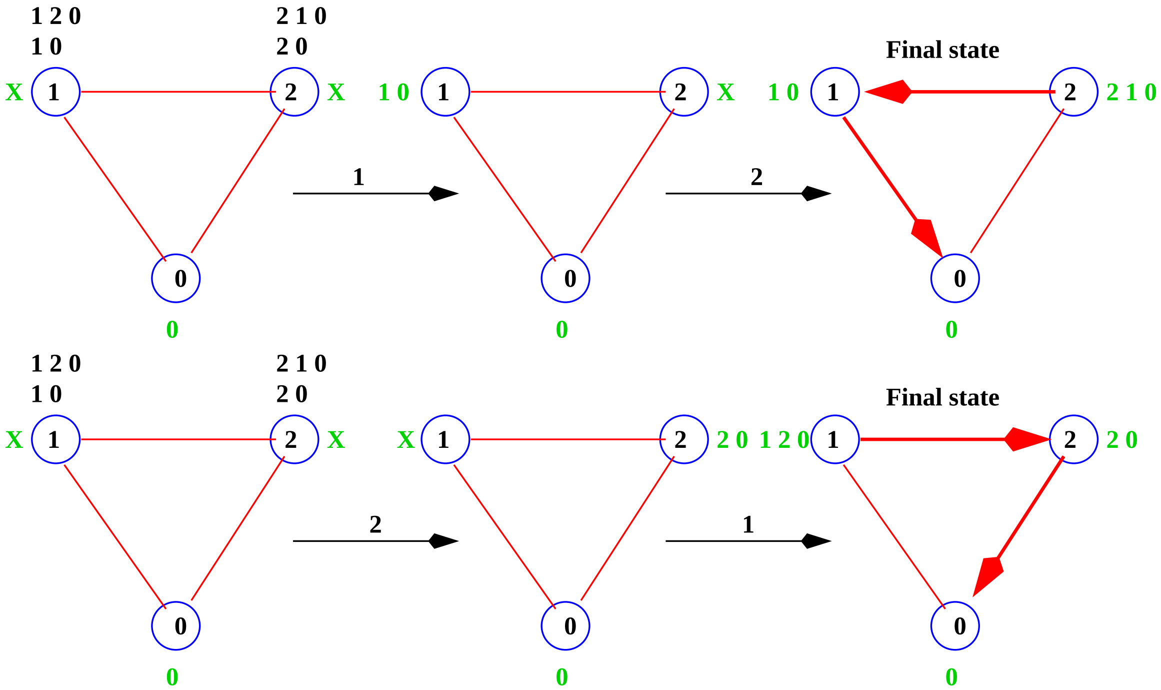

Figure 3: A solvable BGP system with multiple stable states.

policies of the clause nodes, all of the clauses are satisfied; so F is satisfied. This completes the reduction and the proof of NP-completeness.

2 Multicommodity flow formulations

We now switch gear to intradomain routing. Most all of the intradomain routing protocols implement shortest-paths routing. The shortest-paths problem assumes fixed link weights. Link weights typically represent queueing delays at nodes, which depend on the rate of traffic flowing through them, and hence changes with time. As noted in Lecture 1, the distance-vector protocols do not respond quickly to changes in link state while link state protocols may not scale when networks become larger. Another drawback of shortest-paths routing that each source-destination path is selected independent of other paths. This in turn results in changes in link weights, which may cause oscillations.

Optimal routing attempts to determine routes and flows along the routes so as to optimize some appropriate global objective. The objectives and constraints can be formulated as a *multicommodity flow* problem. In a multicommodity flow problem, we are given a network $G = (V, E)$ with capacity $C(i, j)$ for each directed edge (i, j). Let $r_i(j)$ be the traffic rate from source i to destination j. While routing traffic, we obtain a *flow*; that is, traffic splits among multiple paths. We need to find paths between source-destination pairs and need to determine how much traffic to route along each path. Differing constraints and objectives define different problems.

Two kinds of constraints that are standard in multicommodity flow problems are the capacity and flow conservation constraints. Let $f_{ik}(j)$ denote the amount of traffic destined for j being sent on link (i, k). The capacity constraint is given by the condition

$$\sum_j f_{ik}(j) \leq C(i, k).$$

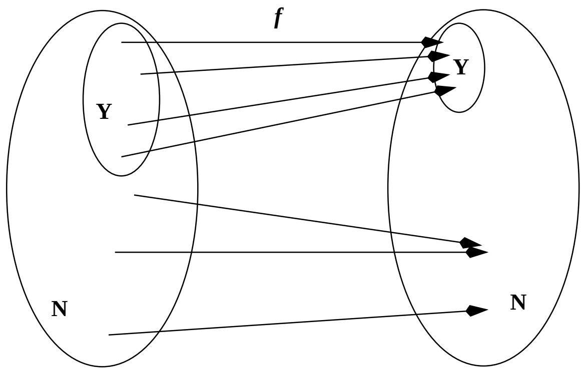

Figure 4: Reduction from one decision problem to another.

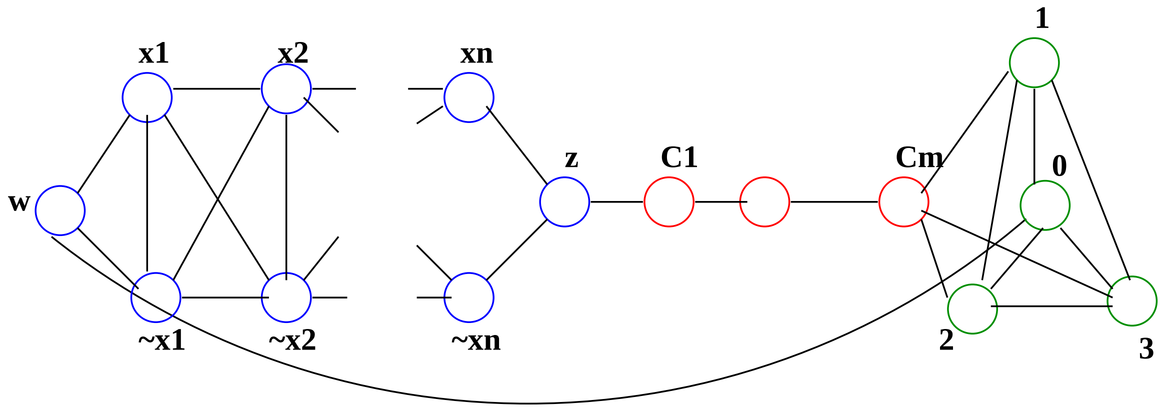

Figure 5: SPP instance corresponding to a given Boolean formula.

The flow conservation constraint says that the amount of traffic for j coming into node i equals the amount of traffic for j leaving node i. Formally, we have

$$r_i(j) + \sum_k f_{ki}(j) = \sum_k f_{ik}(j).$$

Maximum multicommodity flow. A natural objective is to maximize the sum of the traffic rates that can be delivered in the network. We can think of this formulation as that of maximizing total throughput. Thus, the objective function is to

$$\text{maximize } \sum_{i,j} r_i(j),$$

subject to capacity and flow conservation constraints.

Note that we have expressed the optimization problem as a *linear program*. In principle, we can solve the problem "efficiently" using known optimization techniques.

Maximum concurrent flow problem. A "fairer" objective would be to maximize the *fraction of demand* routed for each commodity. Assume that $r_i(j)$ is given as a demand for the commodity that needs to be shipped from i to j. Our goal is to maximize the fraction of this demand that we can route. Thus formally, we need to maximize fraction μ subject to the capacity and flow conservation constraints. The latter may be rewritten as

$$\mu r_i(j) + \sum_k f_{ki}(j) = \sum_k f_{ik}(j).$$

We can again solve the above problem using linear programming algorithms. These algorithms tend to be complex and are usually not amendable to a distributed implementation. In the next class, we will study an elegant local-control algorithm for this problem.

Minimum delay routing problem (MDRP). An objective particularly well-suited for routing is that of minimizing average delay. As before, suppose we are given a capacitated network $G = (V, E)$ and pairwise traffic rates $r_i(j)$ (traffic rate from i to j). We need to route the traffic along paths between sources and destinations such that average delay is minimized, subject to flow conservation and capacity constraints.

Typically, the average delay on a link is an increasing (convex) function of the amount of traffic on the link. Suppose, the average delay per packet going on link (i, k) is is given by $D_{ik}(f_{ik})$. Since delay is additive, our objective can be formally stated as

$$\text{minimize } D_T = \sum_{(i,k)} D_{ik}(f_{ik}).$$

3 A distributed algorithm for MDRP

Unfortunately, the objective function for MDRP is not linear; so the problem is very complicated in general. We now take a brief look at a distributed algorithm due to Gallager [Gal77].

Let $t_i(j)$ denote the total flow (traffic) entering node i that is destined for j; thus,

$$t_i(j) = r_i(j) + \sum_k f_{ki}(j).$$

Let $\phi_{ik}(j)$ denote the fraction of $t_i(j)$ that is sent out along link (i, k).

$$t_k(j) = r_k(j) + \sum_i t_i(j)\phi_{ik}(j).$$

We can restate the problem in terms of ϕ as follows. We are given the rates $r_i(j)$, the capacity, and flow conservation constraints. We need to find the fractions $\phi_{ik}(j)$ that minimize the average delay. We have the following conditions on ϕ:

- $\phi_{ik}(j)$ is 0, if (i, k) is not an edge.
- $\sum_k \phi_{ik}(j) = 1$, for all j.

The variable set ϕ can be thought of as a *routing variable* and can be realized by the individual nodes.

Conditions for optimality. Using elementary calculus, we can show that a sufficient condition for optimality is as follows: For each pair of nodes i, j, there exists constant λ_{ij} such that

$$\frac{\partial D}{\partial \phi_{ik}(j)} \begin{cases} = & \lambda_{ij}, \text{ if } \phi_{ik}(j) > 0 \\ \geq & \lambda_{ij}, \text{ if } \phi_{ik}(j) = 0. \end{cases}$$

The quantity $\partial D_T / \partial \phi_{ik}(j)$ can be written as

$$t_i(j)[D'_{ik}(f_{ik}) + \frac{\partial D_T}{\partial r_k(j)}],$$

where $D'_{ik}(f_{ik})$ is called the *marginal delay* and $\frac{\partial D_T}{\partial r_k(j)}$ is the *marginal distance* of k from j.

Gallager's algorithm for MDRP. Gallager's algorithm consists of three parts: (i) a protocol to calculate the marginal distances, (ii) a separate mechanism to estimate marginal delays, and (iii) a rule for updating the $\phi_{ik}(j)$. Initially, the $\phi_{ik}(j)$ can be set on the basis of some shortest-path computation; that is, they are set to 1 for the edges along the shortest paths, and 0 otherwise.

The following rule is proposed for updating $\phi_{ik}(j)$:

- If $\frac{\partial D_T}{\partial r_k(j)}$ is large, then reduce $\phi_{ik}(j)$ by ε.
- Move excess fraction over to $\phi_{im}(j)$ that minimizes $\partial D_T/\partial \phi_{ik}(j)$.

Gallager proves that for ε suitably small, the above iterative algorithm converges to the minimum delay routing solution. The choice of ε, it turns out, is quite crucial in the rate of convergence of the algorithm. A small ε may take a large time to converge, while a large ε may cause the system to diverge or oscillate around the minimum.

References

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[Gal77] R. G. Gallager. A minimum delay routing algorithm using distributed computation. *IEEE Transactions on Communications*, 25:73–85, 1977.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[GW99] Timothy Griffin and Gordon T. Wilfong. An analysis of BGP convergence properties. In *Proceedings of ACM SIGCOMM*, pages 277–288, 1999.