A Practical Markov Chain Monte Carlo Approach to Decision Problems

Timothy Huang, Yuriy Nevmyvaka
Department of Mathematics and Computer Science
Middlebury College
Middlebury, VT 05753
huang@middlebury.edu, nevmyvak@middlebury.edu

Abstract

Decision and optimization problems involving graphs arise in many areas of artificial intelligence, including probabilistic networks, robot navigation, and network design. Many such problems are NP-complete; this has necessitated the development of approximation methods, most of which are very complex and highly problem-specific. We propose a straightforward, practical approach to algorithm design based on Markov Chain Monte Carlo (MCMC), a statistical simulation scheme for efficiently sampling from a large (possibly exponential) set, such as the set of feasible solutions to a combinatorial task. This facilitates the development of simple, efficient, and general solutions to whole classes of decision problems. We provide detailed examples showing how this approach can be used for spanning tree problems such as Degree-Constrained Spanning Tree, Maximum Leaf Spanning Tree, and K^{th} Best Spanning Tree.

Introduction

Decision and optimization problems involving graphs arise in many areas of artificial intelligence, including probabilistic networks, robot navigation, and network design. Many such problems are NP-complete; this has necessitated the development of approximation methods, most of which are very complex and highly problem-specific. We propose a straightforward, practical approach to algorithm design based on Markov Chain Monte Carlo (MCMC), a statistical simulation scheme for efficiently sampling from a large (possibly exponential) set, such as the set of feasible solutions to a combinatorial task. This facilitates the development of simple, efficient, and general solutions to whole classes of decision problems. We provide detailed examples showing how this approach can be used for spanning tree problems such as Degree-Constrained Spanning Tree, Maximum Leaf Spanning Tree, and K^{th} Best Spanning Tree.

Our paper is organized as follows: First, we describe the basic minimum spanning tree problem and discuss several decision problems derived from it, some of which are NP-complete. Second, we provide an overview of the MCMC method. Third, we show how the MCMC method can be used to construct approximation algorithms for the aforementioned spanning tree problems in particular and for graph optimization and decision problems in general. Finally, we conclude with a discussion of our contributions and of future work.

Spanning Tree Problems

We begin by describing the minimum spanning tree problem. Minimum spanning tree problems are among the most important in network topology design. Practical applications might involve minimizing the amount of wiring used to connect n computers or minimizing the number of connections to any one computer in a network. Formally, we are given a graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. Each edge $(u,v) \in E$ has some weight $w(u,v)$. The minimum spanning tree problem is to find some subset T of E that connects all the vertices and whose total weight is minimized. Many algorithms have been developed to find minimum spanning trees efficiently. The most popular include Kruskal's and Prim's algorithms [Cormen, Leiserson, and Rivest 1990].

A number of decision and optimization problems can be derived from the basic minimum spanning tree problem by imposing additional constraints on the spanning tree. For example, in the Degree Constrained Spanning Tree (DCST) problem, the task is to determine whether there is a spanning tree for some input graph G in which no vertex has degree larger than K. This problem is NP-complete for any fixed $K \geq 2$. Note that with K=2, the Degree Constrained *Minimum* Spanning Tree problem reduces to the Traveling Salesperson Problem (TSP), a classic NP-complete problem. DCST arises in the area of network design: to ensure the reliability of a network, we may wish to limit the number of other machines affected should any one machine go down. Many approximate solutions have been suggested for DCST. Among these are approaches based on genetic algorithms [Chou, Premkumar, and Chu 1999] and parallel algorithms [Mao, Deo, and Lang 1998].

A related problem is the Maximum Leaf Spanning Tree (MLST) problem. The task here is to determine whether there exists a spanning tree for an input graph G in which K or more vertices have degree 1. Like DCST, this problem is NP-complete and has applications in network design and circuit layout. We may, for example, wish to design a network topology that minimizes the number of intermediate nodes and maximizes the number of client nodes at leaves. Yet another related problem is the K^{th} Best Spanning Tree (KBST) problem. The task here is to determine whether there are K distinct spanning trees for an input graph G where each tree has total weight B or less.

Markov Chain Monte Carlo

Following the treatment in [Jerrum and Sinclair 1996], we provide an overview of the Markov Chain Monte Carlo (MCMC) method. The MCMC method makes it possible to efficiently sample from a large combinatorial set according to a desired probability distribution and can be used for function optimization.

Suppose that Ω is a large, finite combinatorial set and that f: $\Omega \rightarrow \mathbf{R}^+$ is a function defined on Ω. We may wish to find a solution $x \in \Omega$ such that $f(x)$ is maximal. The MCMC method requires that we define an undirected, connected graph W on all possible solutions to the problem and a set of one or more *moves*, i.e., relatively simple operations that transform one element of Ω to another. Each vertex of W represents a member of Ω, i.e., one possible solution, and each edge represents a move. W is called a *neighborhood structure* because the neighbor vertices of any vertex x correspond to the set of solutions reachable via a single move from the solution represented by x.

How a neighborhood structure is defined depends on the particular problem. For example, to find the maximum cut of a graph G, we might define W such that each vertex represents a partition of G into two subgraphs. The edges of any vertex in W would connect to vertices corresponding to partitions that could be obtained by moving an element in G from one subgraph to the other. By contrast, to find the maximum number of matchings in an undirected graph G, we might define W such that each vertex denotes a set of matchings in G, and each edge connects a set to another set obtained by removing, adding, or replacing a single matching.

By representing each vertex of W with a set of features and each edge as a change to one feature, we can simulate a Markov chain through the space of possible solutions. In particular, we represent each vertex $X \in W$ as a vector of k random variables, i.e., $X = \{ X_1,..., X_i,..., X_k\}$, where k is the number of parameters needed to describe elements in G. An edge between X and another vertex $X' = \{X_1,..., X'_i,..., X_k\}$ indicates that X and X' differ by one parameter value X_i $(1 \leq i \leq k)$. A random walk on the vertices of W thus represents a sequence of solutions in which each solution differs from the previous solution by at most one feature (solutions can repeat). This sequence is a Markov chain because at each step in the sequence, the choice of the next solution depends only on the current solution and not on any previous ones.

In short, the MCMC approach to optimization and decision problems involves simulating the Markov chain for some number of steps T, beginning with an arbitrary initial solution, and then either outputting the best solution seen so far or outputting whether a solution has been found. Due to the Markov property, any algorithm based on generating a randomized sequence of solutions doesn't need to maintain a data structure for the entire graph W. In most cases, this would be impossible, given the large size of Ω.

We introduce bias into the random walk so that it favors better solutions. This usually means always accepting transitions of the Markov chain to states with higher f value but occasionally accepting transitions to states with lower f value. In particular, suppose *degree(x)* denotes the degree of vertex x in W, and suppose $D(W)$ denotes an upper bound on the maximum degree. The transition from a current vertex x to the next vertex is specified as follows:

I. with probability 0.5 let $y = x$; otherwise,

II. select y according to the distribution:

$$Pr(y) = \begin{cases} 1/D(W) & \text{if } y \text{ is a neighbor of } x; \\ 1 - degree(x)/D(W) & \text{if } y = x; \\ 0 & \text{otherwise} \end{cases}$$

III. with probability $min\{1, \alpha^{f(y)-f(x)}\}$, transition to y; otherwise, stay at x, i.e., repeat the state represented by x.

Note that $\alpha \geq 1$; this ensures that transitions to neighbors with higher f values are always accepted and that transitions to neighbors with lower f value are rejected with probability $(1 - \alpha^{f(y)-f(x)}) < 1$. The last inequality holds since $f(y) - f(x) < 0$ when the state denoted by y is a less desirable state; thus, $0 < \alpha^{f(y)-f(x)} < 1$ and $(1 - \alpha^{f(y)-f(x)}) < 1$.

To show that the resulting Markov chain converges to a stationary distribution, we note the following properties: First, since W is connected by definition, the Markov chain is irreducible, i.e., any state can eventually be reached from any other state. Second, since all self-loop probabilities are required to be non-zero, the chain is aperiodic and hence ergodic, i.e., there are no fixed cycles through which states will alternate. The probability distribution of this chain can then be defined as

$$\pi_\alpha = \alpha^{f(x)}/Z(\alpha), \text{ for all } x \in \Omega, \qquad (1)$$

where $Z(\alpha)$ is a normalizing constant that ensures π_α is a probability distribution. Finally, we note that the chain is also reversible, i.e., it satisfies the detailed balance condition:

$$\pi_\alpha(x)\, P(x, y) = \pi_\alpha(y)\, P(y, x), \text{ for all } x, y \in \Omega.$$

These conditions guarantee that the Markov chain converges to the stationary distribution π_α. Detailed definitions of each requirement and proofs of convergence can be found in [Tierney 1996] and [Taylor and Karlin 1984]. A Markov chain of this form is known as a *Metropolis process* [Metropolis, et.al. 1953].

The parameter α influences the rate at which the algorithm finds better solutions and the ability of the algorithm to move beyond local maxima in the solution space. Lower values of α smooth out the distribution π_α and help keep the chain from getting stuck in local maxima; higher values of α make the distribution π_α more peaked around optimal solutions and help find better

solutions more quickly. At the two extremes, we have an unbiased random walk on the graph W when $\alpha = 1$ and a greedy search when $\alpha = \infty$. Hence, some intermediate value for α is usually preferred. Jerrum and Sinclair provide a detailed analysis of the *Metropolis algorithm at* α in [Jerrum and Sinclair 1996]. Varying α while the process is simulated results in a *simulated annealing* algorithm [Kirkpatrick, Gelatt, and Vecchi 1983].

MCMC Applied to Spanning Tree Problems

Having discussed minimum spanning tree problems and the MCMC method, we now show how the method can be applied in a general, straightforward way to developing approximation algorithms for a whole class of decision problems. In many situations, this may be more practical than using a complex, specialized solution that doesn't adapt readily to related problems. As an example, we show how the same neighborhood structure can be used for DCST, MLST, and KBST. Only the heuristic evaluation function and the α parameter are different in each problem.

Selecting a neighborhood structure

The first step in applying MCMC to a minimum spanning tree problem on some input graph G is defining an appropriate graph representation W of the space Ω. It would be problematic to restrict this to the set of all possible solutions, i.e., to the set of all spanning trees. The difficulty lies in finding simple moves that transform one spanning tree to another. Without these, we aren't able to define a suitable neighborhood structure for W. Furthermore, finding an arbitrary initial solution in a space of spanning trees may be nontrivial for complex graphs.

To address these concerns, we define Ω to include all possible subgraphs of a graph G that are connected trees. Vertices in W represent subgraphs of G, and edges in W connect vertices corresponding to subgraphs that differ by one additional or one fewer edge. This allows us to define a connected neighborhood structure that includes all spanning trees of the graph G. Choosing an initial state for the Markov chain becomes trivial; it can be any single vertex from the input graph G. From this vertex, we can grow a solution for any minimum spanning tree problem by incrementally adding additional edges.

With modifications to the set of vertices and the set of allowable edges, the graph representation W can be adapted to describe the neighborhood structure for many other graph decision and optimization tasks, not only spanning tree problems. Once the graph W is defined, it can be applied to many problems in that class by using different maximization functions and α parameters. We provide examples of this process in our solutions to the MLST and KBST problems.

Degree Constrained Spanning Tree

We now describe how we applied the MCMC method to DCST, where the task is to determine whether there is a spanning tree for input graph G in which no vertex has degree larger than some value K. We will address the following tasks: constructing the Markov chain, designing an evaluation function, and choosing an appropriate value for the α parameter.

Our algorithm and edge update rules adapt the model of [Jerrum and Sinclair 1996] to degree constrained spanning trees. Here, E denotes the set of edges in W and each instance of *random()* denotes a new random number between 0 and 1.

Step 1. Define neighborhood. Choose initial state $x \in W$.
Step 2. If *random()* < 0.5, stay at x; else,
If *random()* < (1–(degree(x))/(|E|)), stay at x; else
Randomly choose a neighbor y.
If the $f(y) > f(x)$,
move to y.
Else if *random()* < $\min\{1,\alpha^{f(y)-f(x)}\}$, move to y
Else stay at x.
Step 3.: Repeat step 2 for some number of steps T or until a state corresponding to a minimum spanning tree of degree K or less is found.
Step 4. If tree found, output **yes**; else output **don't know**.

The initial state of the Markov chain is a vertex in W corresponding to a random vertex of the input graph G. Its neighbors in W represent subgraphs of G that consist of a single edge in G, one vertex of which is the initial vertex. At each step in the chain, the subgraph represented by the current vertex transitions to another subgraph made by adding an edge to or deleting an edge from the current subgraph, subject to the constraint that the graph remains a connected tree.

Recomputing the neighborhood of a vertex at each transition is inefficient. Instead, we maintain a list L of edges that can be added to the current subgraph, and we update the list at each transition. The following rules summarize how L is updated when an edge (v_1, v_2) is added to the subgraph (where v_1 already belongs to the subgraph and v_2 is a new vertex) or when edge (v_1, v_2) is deleted from the subgraph (where v_1 remains part of the subgraph but v_2 does not):

After adding a new edge (v_1, v_2): Add to L those edges in G the have one vertex v_2 and the other vertex not in the current subgraph. Remove from L those edges that have one vertex v_2 and the other in the current subgraph (to prevent cycle creation). Remove from L those edges that have one vertex v_1 and the other vertex belonging to another edge in the subgraph (to prevent splitting the subgraph in two)

After removing an edge (v_1, v_2): Add to L those edges of G that have one vertex v_1 and the other vertex not in the current subgraph. Add to L those edges with one vertex v_2 and the other vertex belonging to the current subgraph (these edges would have created loops before). Remove from L those edges with one vertex v_2 and the other vertex not in the current subgraph (these edges are no longer reachable).

We now propose a heuristic evaluation function to direct the Markov chain toward solution states. The aim is to favor subgraphs with as many edges as possible but with no vertices having degree greater than K. Suppose that the Markov chain algorithm proposes adding an edge so that the resulting subgraph has a maximum vertex degree of M. If this subgraph has increased maximum vertex degree and $M > K$, we discourage the transition by accepting it with probability α^{K-M}. The greater the difference between M and K, the less likely we are to accept the transition. If $M < K$, we always accept the transition since it brings the subgraph closer to a spanning tree. If the algorithm proposes removing an edge, we accept the transition with probability $\alpha^{-1/(|E|)}$, where $|E|$ is the number of edges in the current subgraph. The greater the percentage by which the subgraph size is reduced, the less likely we are to accept the transition. In this way, we discourage transitions away from a spanning tree. The only time we prefer a reduction to the size of the tree is when it decreases the maximum degree of a tree whose maximum degree already exceeds K; in those cases, we always accept the transition. We believe this evaluation function is intuitively compelling, and preliminary experimental results have shown it to work efficiently in practice.

We now address the task of choosing a parameter α. For some problems, a higher value works fine, whereas for other problems it leads to difficulties. The Markov chain generated by our algorithm starts from a single vertex and incrementally adds vertices on most transitions. Even when it needs to remove edges in order to make additional progress toward a spanning tree, it rarely needs to backtrack many steps in sequence. Hence, in practice the value of α can be set quite high without much danger of the chain getting trapped in local maxima. Nevertheless, performance begins to deteriorate once α surpasses a certain threshold, reducing the algorithm to greedy search.

Choosing α remains something of a black art. [Jerrum and Sinclair 1996] describe some techniques for doing so, though to this day there are few rigorous results. In general, we wish to minimize *hitting time*, the time it takes to reach one state when starting from another. Preliminary results with our evaluation function and α parameter suggest that the hitting time of our Markov chain algorithm grows in polynomial time with the size of the input graph G, though this requires further investigation. [Jerrum and Sinclair 1996] and [Sinclair 1992] discuss hitting time and convergence in substantial detail.

Related Spanning Tree Problems

A key benefit of the MCMC approach is that algorithms for related problems can be devised using the same neighborhood structure. For example, we now propose an algorithm to solve the MLST problem that uses the same neighborhood structure as DCST but has its own evaluation function and other considerations for choosing α. In MLST, the task is to determine whether there exists a spanning tree for an input graph G in which K or more vertices have degree 1.

We use an algorithm nearly identical to the one proposed for DCST. Generally, the algorithm encourages the addition of new edges and discourages the removal of existing edges. With MLST, however, we wish to find trees with K or more leaves. Thus, we modify our algorithm so that when it suggests transitions to subgraphs with K or more leaves, we impose no additional restrictions on adding or removing edges. When it suggests transitions to subgraphs with fewer than K leaves, however, we reject such transitions in proportion to the difference between the resulting number of leaves and K. The rules governing edges that can be added or removed remain the same.

In some ways, MLST is the opposite of DCST, and this influences our choice of evaluation function and of α. Whereas the DCST algorithm has little trouble keeping the maximum degree of any vertex below K while searching for a spanning tree, the MLST algorithm must constantly trade off between trying to increase the size of the subgraph and keeping the number of leaves above K. Backtracking several steps in sequence occurs much more frequently with the MLST algorithm. Hence, our evaluation function places greater weight on tree growth when the subgraph is small and greater weight on maintaining the number of leaves as the subgraph gets larger. To provide the chain enough mobility to accommodate the two conflicting goals, we use a relatively small value for α. As with DCST, we can't allow the value of α to become too small, since the Markov chain becomes a random walk as α approaches 1.

Like MLST, an approximation algorithm for solving KBST can be developed using the original neighborhood structure. Since the task is to determine whether there are K distinct spanning trees for input graph G, each with total weight B or less, there are now three possibly conflicting goals: growing the size of the tree, keeping the weight of the tree below B, and finding trees different from the ones already discovered. Again, we can reuse the neighborhood structure and basic algorithm. As with MLST, our evaluation function weights the signficance of each goal depending on the development of the subgraph. It also uses an even smaller value of α to encourage exploration. While experimental results are not the focus of this paper, we

have implemented algorithms for all three spanning tree problems, and preliminary results for convergence have been promising.

Generalizing the Approach

Having discussed how MCMC can be applied to three different but related spanning tree decision problems, we now consider the general principles involved in developing solutions to decision and optimization problems.

First, we must define a set of all feasible solutions to the problem, along with the features that characterize optimal solutions. Furthermore, we must identify simple operations that transform one candidate solution to another. It's important that every solution to be reachable from any other solution and appear exactly once in the set. Otherwise, some optimal solutions may be left out or some states may receive undue preference.

Next, we must engineer a Markov chain that performs a biased random walk on the search space and that satisfies previously defined convergence conditions. So that the chain will favor more desirable states over less desirable ones, we define a heuristic evaluation function on states. This function should capture the essence of optimal solutions, assigning higher values to states closer to them. If some cases, the evaluation function will need to weigh the tradeoffs between conflicting optimizing criteria.

The choice of the α parameter depends heavily on the specific problem. As a rule of thumb, α should be small when the Markov chain requires substantial mobility throughout the solution space but can be large when the chain performs little backtracking. Usually, α must be refined experimentally.

Conclusions and Future Work

The primary goal of this paper has been to show how the Markov Chain Monte Carlo method can be applied to solving whole classes of decision problems. We have demonstrated this by describing in detail how the same basic approach can produce approximation algorithms for three different spanning tree problems: Degree Constrained Spanning Tree, Maximum Leaf Spanning Tree, and K^{th} Best Spanning Tree. Each solution uses the same neighborhood structure and basic Markov chain algorithm. They differ only in their evaluation functions and selection of α parameter. Thus, MCMC provides a practical alternative to developing and using complex, ad hoc solutions to closely related problems.

A second contribution is the adaptation of Jerrum and Sinclair's basic Markov chain algorithm to decision problems involving spanning trees. We provide rules explaining how the sequence of subgraphs in the chain can be generated without recomputing the neighbors of each vertex after every transition.

A third contribution is the specification and description of heuristic evaluation functions and the α parameter for driving the evolution of the Markov chain for each problem, particularly DCST.

We have implemented algorithms for DCST, MLST, and KBST, and they have produced promising experimental results. We are currently comparing how these approximation algorithms perform in practice against other types of approximation algorithms for the same problems.

Finally, there remains much room for stronger theoretical results. We are working to develop techniques for finding mathematical bounds for the convergence rates of our algorithms and for determining optimal values of the parameter α.

Acknowledgements. This work was supported by Middlebury College and by the National Science Foundation (NSF) under grant #9876181.

References

Chou H., Premkumar G., and Chu C.-H. 1999. "Genetic Algorithms and Network Design: An Analysis Of Factors Influencing Network Performance." *EBusiness Research Center Working Paper.*

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

Jerrum, M.R. and Sinclair, A.J., 1996. "The Markov Chain Monte Carlo Method: An Approach to Approximate Counting and Integration," In *Approximation Algorithms for NP-Hard Problems*, Hochbaum D.S., ed. Brooks/Cole.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. 1983. "Optimization by Simulated Annealing." *Science*, 220:671–680.

Mao, L.-J., Deo, N., and Lang, S.-D. 1998. "A Parallel Algorithm for the Degree-Constrained Minimum Spanning Tree Problem Using Nearest Neighbor Chains." In *Proceedings of the Fourth International Symposium on Parallel Architectures, Algorithms, and Networks.*

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., and Teller, E. 1953. "Equation of State Calculation by Fast Computing Machines." In *Journal of Chemical Physics*, 21:1087–1092.

Sinclair A. 1992. "Improved Bounds for Mixing Rates of Markov Chains and Multicommodity Flow." In *Combinatorics, Probability and Computing*, 1:351-370.

Taylor, H.M., and Karlin, S. 1984. *An Introduction to Stochastic Modeling.* Orlando, FL: Academic Press.

Tierney, L. 1996. "Introduction to General State-Space Markov Chain Theory." In *Markov Chain Monte Carlo in Practice.* Gilks W.R., Richardson S., and Spiegelhalter D.J., eds. Boca Raton, FL: Chapman & Hall.