H1N1 Pandemic

21-393 Fall 2009

Reducing the Spread via

Optimal Placement of Hand Sanitizers

Team D

Da Jeong Ha

Jai Woo Lee

**Abstract** Most networks use shortest path to find the most efficient and optimal way. Given proper weight for the links of the network, shortest paths should be found with respect to demand of people and these weights (lengths) from their starting point (source) to their ending point (destination). In this report, we present an integer programming algorithm which maximizes total weights (demand of people) on shortest path problem since the higher demand or weight means the higher need for hand sanitizers. First, given a directed or undirected graph and a set of communication demands, the goal is to find routing lengths from a unique shortest path for each demand. We follow the general search algorithm approach our algorithm is rooted from, present the integer and linear programming models used to solve the some special problems, and discuss the most important operation aspects. Finally, we report computational results for various problems, which derive the efficiency of our algorithm from our problems.

**Introduction** In this paper, we present an integer programming algorithm to place the sanitizers in optimal position of communication networks based on shortest path routing and search algorithms. And, this paper suggests that school government use this mathematical model as one of the solutions to set hand sanitizers by lowering expected costs and time.

**What is 2009 H1N1 (swine flu)?** 2009 H1N1 (sometimes called “swine flu”) is a new influenza virus causing illness in people. This new virus was first detected in people in the United States in April 2009. This virus is spreading from person-to-person worldwide, probably in much the same way that regular seasonal influenza viruses spread. On June 11, 2009, the [World Health Organization](http://www.who.int/csr/disease/swineflu/en/index.html) (WHO) signaled that a pandemic of 2009 H1N1 flu was underway.

**Why is 2009 H1N1 virus sometimes called “swine flu”?**
This virus was originally referred to as “swine flu” because laboratory testing showed that many of the genes in this new virus were very similar to influenza viruses that normally occur in pigs (swine) in North America. But further study has shown that this new virus is very different from what normally circulates in North American pigs. It has two genes from flu viruses that normally circulate in pigs in Europe and Asia and bird (avian) genes and human genes.

**Is 2009 H1N1 virus contagious?**
The 2009 H1N1 virus is contagious and is spreading from human to human.

**How does 2009 H1N1 virus spread?**
Spread of 2009 H1N1 virus is thought to occur in the same way that seasonal flu spreads. Flu viruses are spread mainly from person to person through coughing or sneezing by people with influenza. Sometimes people may become infected by touching something – such as a surface or object – with flu viruses on it and then touching their mouth or nose.

**Take these everyday steps to protect your health:**

Cover your nose and mouth with a tissue when you cough or sneeze. Throw the tissue in the trash after you use it.

* Wash your hands often with soap and water. If soap and water are not available, use an alcohol-based hand rub.[\*](http://www.cdc.gov/h1n1flu/qa.htm#antibacterial)
* Avoid touching your eyes, nose or mouth. Germs spread this way.
* Try to avoid close contact with sick people.

**Notation and Preliminaries**

$δ$**(u, v) :** distance between u and v in the graph

$δ$**+(v) :** out-degree of vertex v

$δ$**-(v) :** in-degree of vertex v

**E(v) :** The set of edges incident on vertex v

$ε\_{M}$**(v) :** If v is not in M, then the set of edges adjacent to vertex v with weight greater than $δ$(u, M), otherwise$∅$

**Proposition:** Let G be a bipartite graph, with bipartition {A, B} say. If G contains a matching of A and a matching of B, then G has a 1-factor.

Proof: Let H be the multi-graph on V(G) whose edge set is the disjoint union of two matching (any edge that lies in both matching becomes a double edge in H) Every vertex in H has degree 1 or 2. Actually, it is clear to check that every component of H is an even cycle or an infinite path. Picking every other edge from each component, we obtain a 1-factor of G.

**Theorem 2.1.2:** G contains a matching of A if and only if |N(S)| $\geq $ |S| for all S $\in $ A

Proof: If G contains no matching of A, then by theorem that the maximum cardinality of a matching in G is equal to the minimum cardinality of a vertex cover of its edges, there is a cover U of E that consists of fewer than |A| vertices, say U=A’$∪$B’ with A’ in A and B’ in B.

 Then, |A’| + |B’| = |U| < |A|,

And hence,

|B’| < |A| - |A’| = |A\A’|

By definition of U, however, G has no edges between A\A’ and B\B’, so

|N(A\A’)|$\leq $|B’| < |A\A’|

And the marriage condition fails for S:= A\A’

 

**The Map of Building as a Graph**

Before setting integer programming algorithm, we should convert map of each floor to a graph. Each room is considered as a node, and the connection way (real or unreal) between every two different rooms is thought as an edge. First, we should start with a complete graph (every room (node) is connected to each other) and, remove unreal or possibly unconnected way between every two different rooms.

First, we should check whether this remaining graph has how many cycles, the existence of cycles may lower the efficiency of our shortest path search algorithms and actually, in order to find the shortest path, we should remove some edges of these cycles.

Second, we should check whether this remaining graph is a bipartite graph. For each room on the map, we alternatively color rooms in order that adjacent rooms have different colors. If the floor of building satisfies this condition, we can do more adjustments in order to find the shortest path by using the bipartition.

Finally, check the graph as a network. The direction can be given onto map (the direction may be from the entrance to the exit.)

I can check these three properties on the map of CMU building.

For example, this is a map of Wean Hall 5th floor.



1. There are three cycles to be considered.
2. This map as a graph is not bipartite. If I exclude one vertex on one of three cycles, then this graph can be bipartite.
3. If I give a direction onto this map from lower right to the left, it may form a network when given weights on each edge.

**Effectiveness and Running Time of Algorithms on Networks**

Prim’s algorithm finds a [minimum spanning tree](http://en.wikipedia.org/wiki/Minimum_spanning_tree) for a connected weighted [undirected graph](http://en.wikipedia.org/wiki/Undirected_graph). It finds a subset of the [edges](http://en.wikipedia.org/wiki/Edge_%28graph_theory%29) that forms a [tree](http://en.wikipedia.org/wiki/Tree_%28graph_theory%29) that includes every [vertex](http://en.wikipedia.org/wiki/Vertex_%28graph_theory%29), where the total weight of all the [edges](http://en.wikipedia.org/wiki/Graph_theory) in the tree is minimized. Prim's algorithm is an example of a [greedy algorithm](http://en.wikipedia.org/wiki/Greedy_algorithm).

 Runtime: O (E + V log(V)) as fastest , by using Fibonacci heap

Kruskal's algorithm is an [algorithm](http://en.wikipedia.org/wiki/Algorithm) in [graph theory](http://en.wikipedia.org/wiki/Graph_theory) that finds a [minimum spanning tree](http://en.wikipedia.org/wiki/Minimum_spanning_tree) for a [connected](http://en.wikipedia.org/wiki/Connectivity_%28graph_theory%29) [weighted graph](http://en.wikipedia.org/wiki/Glossary_of_graph_theory#Weighted_graphs_and_networks). This means it finds a subset of the [edges](http://en.wikipedia.org/wiki/Edge_%28graph_theory%29) that forms a tree that includes every [vertex](http://en.wikipedia.org/wiki/Vertex_%28graph_theory%29), where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest. Kruskal's algorithm is an example of a [greedy algorithm](http://en.wikipedia.org/wiki/Greedy_algorithm).

1. create a forest F (a set of trees), where each vertex in the graph is a separate [tree](http://en.wikipedia.org/wiki/Tree_%28graph_theory%29)
2. create a set S containing all the edges in the graph
3. while S is [nonempty](http://en.wikipedia.org/wiki/Nonempty) and F is not yet spanning
4. remove an edge with minimum weight from S
5. if that edge connects two different trees, then add it to the forest, combining two trees into a single tree
6. otherwise, discard that edge.

Runtime: O(E log E))

Dijkstra's algorithm

For a given source [vertex](http://en.wikipedia.org/wiki/Vertex_%28graph_theory%29) (node) in the graph, the algorithm finds the shortest path between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined.

 Runtime: O( | E | + | V | log | V | ) as fastest, by using Fibonacci heap

Bellman-Ford's algorithm

The Bellman–Ford algorithm, a label-correcting algorithm, computes single-source [shortest paths](http://en.wikipedia.org/wiki/Shortest_path) in a [weighted digraph](http://en.wikipedia.org/wiki/Weighted_digraph) (where some of the [edge](http://en.wikipedia.org/wiki/Edge_%28graph_theory%29) weights may be negative). [Dijkstra's algorithm](http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm) solves the same problem with a lower running time, but requires edge weights to be non-negative.

Runtime: O(V\*E)

**I will choose Dijkstra’s algorithm to find the shortest path on the map of buildings**

**Integer Programming Algorithm** Based on the graphs formed above, I formulate an integer linear program and solve with a branch-and-cut algorithm. Briefly explaining, branch and cut (sometimes written as branch-and-cut) is a method of [combinatorial optimization](http://en.wikipedia.org/wiki/Combinatorial_optimization) for solving [integer linear programs](http://en.wikipedia.org/wiki/Integer_linear_program), where some unknown variables are restricted to [integer](http://en.wikipedia.org/wiki/Integer) values. The method is a mixture of [branch and bound](http://en.wikipedia.org/wiki/Branch_and_bound) and [cutting plane](http://en.wikipedia.org/wiki/Cutting_plane) methods. The method solves the [linear program without the integer constraint](http://en.wikipedia.org/wiki/Linear_programming_relaxation) using the normal [simplex algorithm](http://en.wikipedia.org/wiki/Simplex_algorithm) including duality of algorithm. When we obtain an optimal solution, and the solution of this algorithm has a non-integer value for an unknown variable that was supposed to be integer, a cutting plane algorithm is used to find linear constraints which are satisfied by all feasible integer points but violated by the current fractional solution. If such an inequality is found, it is added to the linear program, such that resolving it will give a different solution which is more like an integer. This process is repeated until either an integer solution is found (which is then known to be optimal) or until no more [cutting planes](http://en.wikipedia.org/wiki/Cutting_plane) are found.

 Our first integer programming model was this;

Minimize z = $\sum\_{i=1}^{n}\frac{x\_{i}}{w\_{i}}$

Subject to: $\sum\_{i=1}^{n}x\_{i}$ $\geq $ 3 \* [$\frac{\sum\_{i=1}^{n}w\_{i}}{n}$]

 $\sum\_{i=1}^{k}x\_{i}\leq $ 2

 .

 .

 .

 $\sum\_{i=1}^{k}x\_{i}\leq $ 2

 $x\_{j1}$=1

 .

 .

 .

 $x\_{j1}$=1

 $x\_{i}\in $ {0, 1}; i=1, ,,,,, n

 Our assumption on this model was this; we give random value of weights (1-5) on each edge depending on the traffic on each edge. And, if there is a hand sanitizer needed, x =1, and if not, x=0. Each vertex is given a value of average weights on adjacent edges. The value z may give us the number of hand sanitizers possibly required. There are some limits and uncertain problems on this model

* Improvement: 1. the uncertainty of assigning random weights on each edge should be

 modified.

 2. Whether the existence hand sanitizers is only determined by checking

the traffic of each edge.

Instead of assigning weight variables on each edge, the formulation below contains special inequalities to exclude path configurations not proper on the shortest path. These inequalities are operated as cutting planes during the execution or the branch-and-cut algorithm (as in lecture 10/26/2009). Given a set of independent paths starting from each vertex computed by the branch-and-cut algorithm, the problem then is to find a metric of path lengths that induce exactly these paths. This problem is formulated and solved as a linear program. If the given paths indeed form a valid shortest path, the linear program is infeasible. In this case, the given paths contain a conflict that must not happen in any acceptable shortest path on this graph. This conflict is valid for all admissible shortest paths, but violated by the current algorithm. Adding this inequality to the problem, we then cut off the current non-admissible algorithm path and proceed with the branch-and-cut algorithm to compute another proper algorithm path.

. The decision variables used in this formulation are the variables $x\_{j}^{i}$ $\in $ {0, 1} for all (S, T) $\in $ K and $a\_{i}$ $\in $ A. These variables describe which arcs are contained in the paths. Variable $x\_{j}^{i}$ is supposed to be 1 if and only if arc a is contained in the path for commodity (S, T). $y\_{j}^{i}$ is the additional artificial variable in {0, 1}.

 Minimize $\sum\_{i=1}^{n}a\_{i}\*x\_{i}$

 Subject to: $\sum\_{a\_{i} \in δ+(v)}^{}x\_{j}^{i}$ - $\sum\_{a\_{i} \in δ-(v)}^{}x\_{j}^{i}$ = 1 if v=t; -1 if v=s; 0 if else.

 $x\_{j}^{i} \leq y\_{j}^{i}$

 $\sum\_{i=1}^{n}y\_{j}^{i}$ $\leq $ 1

 $x\_{j}^{i}$ $\in $ {0, 1}

 $\sum\_{i=1}^{n}x\_{i}$ $\geq $ 0

First, I minimize the total value of variable x multiplied by coefficient a. This value is ultimately what we want and decides the number of hand sanitizers we need. (i of variable x is the index for each room.). Since I do not consider weights of each edge, this formulation may represent un-weighted and directed graph(it is network)

 Second, I set the difference between out-degree and in-degree of each vertex in order to check the position of that specific vertex. Artificial variable should be greater than the variable whether hand sanitizers are needed or not.

 Third, the sum of all artificial variables should be less than or equal to 1. The variable x is 0, or 1 same as we did in our first model. And, the sum of values determining whether the hand sanitizers exist should be greater than or equal to 0.

After converting the map of buildings to a graph, we got the shortest path of each graph. And, by using the integer programming plus cutting planes, we get the minimum but optimal number of hand sanitizers. The last part we should consider is that our model fairly and greatly fits the real world. How can we check this?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problem | Nodes | Links(edges) | Demands(weights) | # hand sanitizers |
| Doherty 2nd | 23 | 32 | 100 | 5 |
| Doherty 1st  | 42 | 50 | 120 | 3 |
| Wean Hall 7th  | 22 | 25 | 90 | 3 |
| Wean Hall 6th  | 24 | 29 | 150 | 4 |
| Wean Hall 5th | 36 | 38 | 130 | 5 |
| Wean Hall 4th | 28 | 45 | 230 | 6 |
| Wean Hall 1st | 5 | 12 | 120 | 1 |
| Hunt Library 1st | 10 | 16 | 350 | 5 |
| Hunt Library 2nd | 8 | 20 | 120 | 2 |
| Hunt Library 3rd | 6 | 16 | 334 | 5 |
| Hunt Library 4th | 6 | 19 | 240 | 4 |
| UC 1st | 29 | 36 | 390 | 6 |
| UC 2nd | 27 | 40 | 280 | 6 |
| UC basement | 18 | 30 | 276 | 5 |
| Baker 1st | 19 | 32 | 126 | 3 |
| Baker 2nd | 21 | 38 | 128 | 3 |
| Porter 1st | 18 | 26 | 200 | 4 |
| Porter 2nd | 19 | 28 | 210 | 3 |
| CFA 1st  | 18 | 29 | 199 | 3 |

Our result directly shows that the number of nodes is always less than the number of edges.

There is a proportional relationship between the number of hand sanitizers and the demand (weights). The number of hand sanitizers can be affected by not only demand but also the interrelation of nodes and edges.

It is our approximate result. We cannot compare approximate (ultimately unreal, and hypothetical) result with real result (the real number of hand sanitizers existing in each building) since hypothetical values cannot be more accurate than real values in the end. However, we can make efforts to make unreal values closer to real values.

 One way is using the probabilistic model such as Bayesian theory (The analysis of prior data to get closer result to posterior data).

**Adding Probabilistic Model onto Our Integer Programming Result** Since a Bayesian network is a perfect model for the variables and their relationships it is used to answer probabilistic queries about them. For example, the network can be used to find out updated knowledge of the state of a subset of variables when other variables are observed. This process of computing the posterior distribution of variables given evidence is called probabilistic inference. The posterior gives a universal [sufficient statistic](http://en.wikipedia.org/wiki/Sufficient_statistic) for detection applications, when one wants to choose values for the variable subset which minimize some expected loss function, for instance the probability of decision error. A Bayesian network can thus be considered a mechanism for automatically applying [Bayes' theorem](http://en.wikipedia.org/wiki/Bayes%27_theorem) to complex problems such as integer programming algorithm.

The most common exact inference methods are: [variable elimination](http://en.wikipedia.org/w/index.php?title=Variable_elimination&action=edit&redlink=1), which eliminates (by integration or summation) the non-observed non-query variables one by one by distributing the sum over the product; [clique tree propagation](http://en.wikipedia.org/w/index.php?title=Clique_tree_propagation&action=edit&redlink=1), which caches the computation so that many variables can be queried at one time and new evidence can be propagated quickly; and [recursive conditioning](http://en.wikipedia.org/w/index.php?title=Recursive_conditioning&action=edit&redlink=1), which allows for a space-time tradeoff and matches the efficiency of variable elimination when enough space is used. All of these methods have complexity that is exponential in the network's [treewidth](http://en.wikipedia.org/wiki/Treewidth). The most common approximate inference algorithms are stochastic [mini-bucket elimination](http://en.wikipedia.org/w/index.php?title=Mini-bucket_elimination&action=edit&redlink=1) which generalizes [loopy belief propagation](http://en.wikipedia.org/wiki/Loopy_belief_propagation), and [variational methods](http://en.wikipedia.org/wiki/Variational_Bayes).

 By using the fact above, we can assign each prior probability on the variable x. First, we should find out what distribution (continuous, or discrete) and how our variables directly relate to each distribution such as Gamma, or Beta. After we get the data results above, based on the data result, we calculate prior and posterior distributions. The average or standard deviation of our data result can be shown after operating the Bayesian model. Then, our result obtained by integer programming can be modified to values much closer to real number of hand sanitizers we need. (or we can calculate the difference of the number of hand sanitizers we really needed and the number of hand sanitizers already there).

<Notes>

Graduate Texts in Mathematics by Reinhard Diestel

An Integer Programming Algorithm for Routing Optimization in IP Networks

Improved Algorithms for All-Pairs Approximate Shortest Paths in Weighted Graphs