This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2009 proceedings.

Graceful Network Operations

Saqib Raza, Yuanbo Zhu, Chen-Nee Chuah
University of California, Davis
Email: {sraza, juzhu, chuah}@ucdavis.edu

***Abstract*—A significant fraction of network events (such as topology or route changes) and the resulting performance degradation stem from premeditated network management and operational tasks. This paper introduces a general class of *Graceful Network Operation* (GNO) problems, where the goal is to discover the optimal sequence of operations that progressively transition the network from its *initial* to a desired *final* state while minimizing the overall performance disruption. We investigate two specific GNO problems: (a) Link Weight Reassignment Scheduling (LWRS) studies the optimal ordering of link weight updates to migrate from an existing to a new link weight assignment, and (b) Link Maintenance Scheduling (LMS) looks at how to schedule link deactivations and subsequent reactivations for maintenance purposes. LWRS and LMS are both combinatorial optimization problems. We use dynamic programming to find the optimal solutions when the problem size is small, and leverage Ants Colony Optimization to get near-optimal solutions for large problem sizes. Our simulation study reveals that judiciously ordering network operations can achieve significant performance gains. Our GNO solution framework is generic and applies to similar problems with different operational contexts, underlying network protocols or mechanisms, and performance metrics.**

I. Introduction

The Internet has been an enabling technology for mission-critical applications and services such as Voice over IP, VPNs, e-commerce applications, and multimedia streaming. Such applications rely upon consistent Quality of Service (QoS) provisioning by Internet Service Providers (ISPs), with five-nines availability (99.999% uptime) becoming the norm rather than the exception. The end-to-end perceived QoS can potentially be affected due to the dynamic nature of the networks. For instance, network topology may change due to transient router/link outages or long-term network engineering. Furthermore, protocol configuration parameters may be altered to migrate from one setting to another. Ideally, QoS guarantees should persist across such dynamic conditions.

Some of these dynamic changes are *inadvertent* e.g., ones due to faulty interfaces, router crashes, and accidental fiber cuts. However, other changes ensue from deliberate and *premeditated* actions of network operators (e.g., routine maintenance). A failure characterization study of the Sprint IP backbone [1] observed that planned maintenance activities account for more than 20% of transient failures. Other studies [2] have also observed the prevalence of such planned maintenance activities. Premeditated network tasks also include network upgrade activities such as adding new routers to the network and overhauling link capacity. Another example of a premeditated network task is migrating an existing OSPF [3] or IS-IS [4][1] link weight assignment to a new assignment that has been optimized based upon the most up-to-date traffic matrix estimates.

In the case of premeditated tasks, network operators have the prerogative to decide the sequence of *atomic* operations that comprise such a task. This paper introduces a general class of problems referred to as *Graceful Network Operation* (GNO) problems, which typically involve migrating a network from its *initial* state to a *final* state by executing a series of *atomic* operations. Each of these operations may cause some *performance disruption* that is a function of the network's state. The GNO problem is to discover the sequence of operations that progressively transition the network to the final state while minimizing the overall disruption. This paper looks at two specific *GNO* problems, as described below.

A. Link Weight Reassignment Scheduling

Setting link weights is the primary tool used by network operators to control network load distribution and to accomplish traffic engineering objectives [5–8]. Link weights are optimized based on an estimate of the traffic matrix. Link weights are not dynamically modified in response to short-term fluctuations in the traffic matrix. However, the estimated traffic matrix may change significantly over a longer period of time, prompting network operators to re-optimize and reset link weights. In such a case, network operators need to migrate from one weight setting to another. The sequence in which the link weights are changed determines the disruption to network traffic during this migration process.

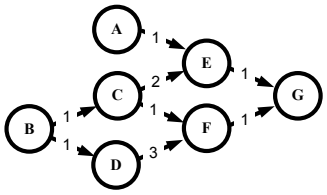


Fig. 1. Example Network

This can be illustrated with an example. Fig. 1 gives an example network with the arcs labeled with IGP link weights. Suppose all links have capacity c, and traffic demands between node pairs (a, g) and (b, g) are both $c/2$. The traffic demand between all other node pairs is 0. The link weights depicted in Fig. 1 are optimal for such a traffic matrix given the objective of minimizing the maximum link utilization (MLU).

[1]Most common intra-domain (IGP) protocols.

978-1-4244-3513-5/09/$25.00 ©2009 IEEE

	Schedule 1		Schedule 2	
Step	**Operation**	**MLU**	**Operation**	**MLU**
1	Switch *w(c,e)*	58.33%	Switch *w(d,f)*	50%
2	Switch *w(d,f)*	45.83%	Switch *w(c,e)*	45.83%
	Cost (Avg.. MLU)	52.08%	**Cost** (Avg.. MLU)	47.91%

TABLE I
LINK WEIGHT REASSIGNMENT SCHEDULING

	Naive Scheduling		Optimal Scheduling	
Step	**Operation**	**MLU**	**Operation**	**MLU**
1	De-activate *(b,c)*	50%	De-activate *(b,c)*	50%
2	Re-activate *(b,c)*	50%	De-activate *(c,f)*	50%
3	De-activate *(c,f)*	100%	Re-activate *(c,f)*	50%
4	Re-activate *(c,f)*	50%	Re-activate *(b,c)*	50%
	Cost (Avg. MLU)	62.5%	**Cost** (Avg. MLU)	50%

TABLE II
LINK MAINTENANCE SCHEDULING

Suppose that the traffic demand between node pair (a, g) changes to $c/3$. Shortest path routing using Equal Cost Multi-Path (ECMP) implies that the reoptimized weight setting corresponds to all weights being 1 [5]. This means that two link weights, $w(c, e)$ and $w(d, f)$, have to be set to 1. Table I shows the two possible migration schedules, where weight changes happen one link at a time. We can see that choosing Schedule 2 minimizes the disruption to network traffic.

A *Link Weight Reassignment Scheduling* (LWRS) problem is characterized by a set of links that we refer to as our *job-set*. All links in the job-set have an *old* weight and a *new* weight. An *atomic* operation involves switching the weight of a link in the job-set from the *old* weight to the *new* weight. The LWRS problem is to find the *minimum-cost* sequence of atomic operations to switch the weights of all links in the job-set. The notion of the cost of a sequence of operations can vary. Table I uses the maximum link utilization at any instant as a measure of disruption cost.

B. Link Maintenance Scheduling

Network links need to be temporarily taken down for maintenance purposes [1]. Since these link deactivations act as normal failures, they have the same impact on network traffic as normal failures. The difference is that, when more than one link needs to be maintained, network operators can determine the order in which links are deactivated and reactivated. Current practice is to fail and restore elements one by one. The intuition behind such a scheme is that the greater the amount of network resources available, the lesser is the disruption to network traffic. This may not always hold as can be seen with the following counter example.

Again we refer to the network in Fig. 1. All links have capacity c. We have a traffic demand of $c/2$ each between node pairs (a, g) and (b, g), and zero elsewhere. Suppose we need to take down links (b, c) and (c, f) for maintenance. Table II gives two possible schedules for the required maintenance. We find, somewhat counter-intuitively, that the solution that has both links simultaneously deactivated is better than the solution that fails and restores the links one by one.

A *Link Maintenance Scheduling* (LMS) problem is characterized by a set of links we refer to as our *job-set*. All links are initially active. All links in the job must be deactivated at least once for maintenance, and then must be reactivated. An *atomic* operation involves either deactivating an active link or activating a deactivated link. The LMS problem is to find the *minimum-cost* sequence of operations to complete the job. Table II uses the maximum link utilization at any instant as a measure of disruption cost.

Both LMS and LWRS fall in the general category of GNO problems. A key contribution of this paper is formulating the general GNO problem and providing a generic solution framework that can be leveraged for a host of problems not restricted to LWRS and LMS. For instance, the GNO solution framework can be used to determine the optimal sequence of upgrading a network by adding new routers and links. Our solution framework can leverage different underlying routing protocols as well as incorporate different metrics of network disruption. The rest of this paper is organized as follows: We discuss related work in Section II. Section III presents our generic GNO problem formulation and shows how LWRS and LMS can be framed within the GNO context. Section IV presents two generic solutions for GNO: a) a Dynamic Programming solution, and b) a solution based on the Ants Colony Optimization meta-heuristic. Section V-A gives the results of our simulation study for LWRS and LMS, and we conclude in Section VI.

II. RELATED WORK

There exists a rich body of work geared towards avoiding disruption to network traffic in the presence of failures and reconfigurations. In networks employing link-state routing protocols such as OSPF and IS-IS, traffic balancing is achieved by judiciously configuring link weights. [7, 8] present solutions that attempt to optimize weights such that the configured weights remain optimal even if network nodes or links fail. In such networks network failure triggers protocol re-convergence which may result in transient routing loops. Extensions to routing protocols have been proposed to avoid such transient loops [9]. A common theme of such solutions has been ordering forwarding table updates during protocol convergence so that transient loops are circumvented. Proactive approaches to mitigate the effects of failure have also been proposed, especially for Multi-Protocol Label Switching (MPLS) networks. Bandwidth guaranteed backup paths can be pre-configured so that the traffic can immediately be transplanted onto backup routes in event of failure. [10, 11] suppresses failure notification and leverages interface-specific forwarding tables to reroute packets across alternative loop-free paths.

All the above solutions attempt to achieve disruption-free network operation by making routing mechanisms and parameter settings resilient to unexpected failures that we have no control over. However, a significant fraction of failures are actually part of planned maintenance activities. For such premeditated maintenance activities we can control the order

and timings of such failures and reconfigurations. Not much attention has been directed towards examining *graceful* network operations for premedicated tasks. Certain best-practices exist such as scheduling maintenance activities in the evening when traffic load is low [1]. This increases the operating costs of the network and falls short of optimizing network operations for the existing traffic conditions. Other solutions involve heuristics such as unit increments to a link's weight till no traffic traverses it [12], setting a link weight to the maximum value to gracefully reroute traffic before failure [13], and failing links at most one at a time to minimize disruption.

More recently, Francois et al. proposed an attractive solution that involves progressively iterating through a sequence a link weight changes in order to fail a link or reconfigure its weight [14]. Their solution is closest in spirit to ours since it leverages the network administrators control over the maintenance activity to mitigate network disruption. However, our focus is entirely different. [14] looks at the progressive weight changes for a single link in order to migrate the link from an initial to a final state. We do not allow such progressive changes but focus on how atomic jobs on a set of links (or nodes) should be ordered so as to minimize overall network disruption. Hence, once an optimal sequence has been determined for the LWRS problem, the solution in [14] can be employed to progressively change the weight for each link.

III. Problem Formulation

A. Graceful Network Operations (GNO)

Our definition of LMS and LWRS problem applies to networks employing IGP protocols such as OSPF and IS-IS. We, therefore, restrict our problem formulation to the context of OSPF/IS-IS for ease of exposition. However, the class of problems falling within the rubric of GNO is much broader. For instance, one can consider a counterpart of the LMS problem for MPLS networks. Our problem formulation can easily be extended to cover such problems.

We consider a network $G(V, E)$, where V is the set of nodes and E is the set of links. We have a traffic matrix $\{\Omega\}_{(i,j)\in V\times V}$ that gives the traffic demand between node-pairs. We define w to be a weight function, $w : E \rightarrow Z^+ \cup \infty$, that assigns a positive integer or ∞ to each link in E. $w(e) = \infty$ implies that link e is down. We also define a *progress* vector $\mathcal{I}$ used to indicate the overall job progress. We use the 4-tuple $(G, \Omega, w, \mathcal{I})$ to represent a network *state*. Let S denote the set of all network states. The network can be in a single unique state at any given point in time. We transition from one network state $s_a = (G, \Omega_a, w_a, \mathcal{I}_a)$ to another $s_b = (G, \Omega_b, w_b, \mathcal{I}_b)$ by altering the weight function from w_a to w_b, where $w_a \neq w_b$. In reality a change of the weight function can result in a change in the traffic matrix as is the case if the BGP egress point for a prefix changes at a router [15]. Our formulation and solution framework can easily account for this. However, for purposes of simplicity we assume that a change in the weight function does not affect the traffic matrix, i.e., we assume $\Omega_a = \Omega_b$. We also have a disruption function $d : S \times S \rightarrow \Re$ that gives the disruption cost to transition from one network state to another. We discuss $d(.)$ in detail in Section III-B.

As mentioned in Section I, GNO basically involves migrating the network from an initial state $s_{\text{initial}} \in S$ to a final state $s_{\text{final}} \in S$. This migration can be realized by executing a series of permissible *atomic network operations*. For every given state $s \in S$, we let the $\mathcal{N}(s)$ denote the set of network states to which we can transition by a single atomic network operation. In other words, the set $\mathcal{A} = \{(s_a, s_b)|s_a \in S \text{ and } s_b \in \mathcal{N}(s_a)\}$ corresponds to the set of all permissible atomic network operations. $\mathcal{N}$ and $\mathcal{A}$ depend upon the specific problem context.

A solution to the GNO problem is a sequence of state transitions $x = (s_{\text{initial}} = s_0, s_1, s_2, ..., s_{n-1}, s_n = s_{\text{final}})$. Eq. 1-5 give the formal specification of the GNO problem.

$$\min_{(s_0,s_1,s_2,...s_{n-1},s_n)} \Gamma((s_0, s_1, s_2, ...s_{n-1}, s_n)) \tag{1}$$

subject to:

$$s_0 = s_{\text{initial}} \tag{2}$$

$$s_n = s_{\text{final}} \tag{3}$$

$$(s_i, s_{i+1}) \in \mathcal{A} \qquad \forall 0 \leq i < n \tag{4}$$

$$n \leq B \tag{5}$$

Eq. 2, 3 stipulate that the solution represents a migration from the initial state to the final state. Eq. 4 constrains the transitions to correspond to atomic network operations. Eq. 5 represents the budget constraint that stipulates the maximum number of transitions (B) allowed to complete the migration. Our objective (Eq. 1) is to minimize the overall cost function $\Gamma(x)$, where x is the sequence of state transitions $(s_0, s_1, s_2, ...s_{n_1}, s_n)$. $\Gamma(.)$ is explained in Section III-B.

We now formulate the LMS and LWRS problems within this framework.

1) Link Weight Reassignment Scheduling: In the Link Weight Reassignment Scheduling (LWRS), let $w_{initial}$ represent the old weight setting and let w_{final} represent the new weight setting to which we need to migrate. Therefore, $s_{\text{initial}} = (G, \Omega, w_{initial}, \mathcal{I}_{initial})$ and $s_{\text{final}} = (G, \Omega, w_{final}, \mathcal{I}_{final})$. Let $J \subseteq E$ represent our *job-set* that contains those links j for which $w_{initial}(j) \neq w_{final}(j)$. The progress vector $\mathcal{I}$ is a $|J|$ element vector, with an indicator variable associated with each link in the job-set:

$$\mathcal{I}(e) = \begin{cases} 0 & \text{if } e \text{ has weight set to } w_{initial}(e) \\ 1 & \text{if } e \text{ has weight set to } w_{final}(e) \end{cases} \tag{6}$$

Hence, $\mathcal{I}_{initial}(e) = 0\ \forall e \in J$, and $\mathcal{I}_{final}(e) = 1\ \forall e \in J$. We define an atomic operation as switching the weight of a link in J from $w_{initial}(j)$ to $w_{final}(j)$. Therefore, $(s_a = (G, \Omega_a, w_a, \mathcal{I}_a), s_b = (G, \Omega_b, w_b, \mathcal{I}_b)) \in \mathcal{A}$ implies that:

- $w_a(e) = w_b(e)$ for all links except for a single link $j \in J$, for which $w_b(j) = w_{final}(j)$.
- $\mathcal{I}_a(e) = \mathcal{I}_b(e)$ for all links except for a single link $j \in J$, for which $\mathcal{I}_b(j) = 1$ and $w_b(j) = w_{final}(j)$.

We set $B = |J|$, which is the minimum number of transitions required to migrate from the initial state to the final state.

2) Link Maintenance Scheduling: In the Link Maintenance Scheduling (LMS), let $J \subseteq E$ represent our *job-set*. Let w_0 represent the starting weight function. All links are initially active i.e. $w_0(e) \neq \infty \ \forall e \in E$. The progress vector $\mathcal{I}$ is a $|J|$ element vector, with a variable associated with each link in the job-set.

$$\mathcal{I}(e) = \begin{cases} 0 & \text{if } e \text{ has never been deactivated} \\ 1 & \text{if } e \text{ is deactivated} \\ 2 & \text{if } e \text{ has been reactivated after deactivation} \end{cases} \tag{7}$$

All links in J must be deactivated at least once for maintenance, and then must be reactivated. s_{initial} is given by $(G, \Omega, w_0, \mathcal{I}_{initial})$, where $\mathcal{I}_{initial}(e) = 0 \ \forall e \in J$. Similarly, s_{final} is given by $(G, \Omega, w_0, \mathcal{I}_{final})$, where $\mathcal{I}_{final}(e) = 2 \ \forall e \in J$.

We consider a link activation or deactivation to be an atomic network operation. Therefore, $(s_a = (G, \Omega_a, w_a, \mathcal{I}_a), s_b = (G, \Omega_b, w_b, \mathcal{I}_b)) \in \mathcal{A}$ implies that:

- $w_a(e) = w_b(e)$ for all links except for a single link $j \in J$, for which either $w_a(j) = w_0(j)$ and $w_b(j) = \infty$, or $w_a(j) = \infty$ and $w_b(j) = w_0(j)$.
- $\mathcal{I}_a(e) = \mathcal{I}_b(e)$ for all links other than j, for which:

$$\mathcal{I}(j) = \begin{cases} 1 & \text{if } w_a(j) = w_0(j) \text{ and } w_b(j) = \infty \\ 2 & \text{if } w_a(j) = \infty \text{ and } w_b(j) = w_0(j) \end{cases} \tag{8}$$

The initial and final state both correspond to the starting weight setting w_0. The definition of s_{initial}, s_{final}, and $\mathcal{A}$ ensures that all links in J are deactivated and re-activated at least once. We set $B = 2 \times |J|$, which is the minimum number of transitions required to migrate from the initial state to the final state.

B. Network Disruption Metric

We now detail the overall cost function $\Gamma(x)$, where x is the sequence of state transitions $(s_0, s_1, s_2, ...s_{n_1}, s_n)$. We define $(s_i, s_{i+1}) \in x$ if and only if there exists a transition from s_i to s_{i+1} in x. Section III-A defined the disruption function $d : S \times S \rightarrow \Re$ that gives the disruption cost to transition from one network state to another. $\Gamma(x)$ is basically a function of the individual disruption costs across all transitions in x, i.e., $d(s_i, s_{i+1})$ for all $(s_i, s_{i+1}) \in x$.

The disruption function can be defined in a variety of ways. In our context of OSPF/IS-IS, moving from one network state to another can result in transient routing loops and packets losses during the time it takes the routing protocol to converge. Once the routing protocol has converged, the traffic distribution across the network may change. Our problem formulation and solution is independent of the choice of disruption function. However, for ease of exposition we confine ourselves to looking at measures of link utilization seen on the network after the routing protocol has converged. In other words, the disruption cost $d(s_a, s_b)$ of a transition from $s_a = (G, \Omega_a, w_a, \mathcal{I}_a)$ to $s_b = (G, \Omega_b, w_b, \mathcal{I}_b)$ is a function of the link utilizations resulting from routing Ω_b on G according to w_b. More specifically, we use two metrics in this paper. One is the utilization of the most congested link, referred to as the MLU metric. Our second metric, F&T, is the widely used metric proposed in [5] that represents a piecewise increasing linear convex envelope of a non-linear link cost function. Our model can easily be extended to account for other and more complex measures of network disruption.

Similarly there exist a number of ways to aggregate the individual disruption costs to compute the overall cost $\Gamma(x)$ for a solution x. To conserve space we restrict ourselves to the *mean* of the individual disruption costs. Hence,

$$\Gamma((s_0, s_1, s_2, ...s_{n-1}, s_n)) = \frac{\sum_{i=0}^{n-1} d(s_i, s_{i+1})}{n} \tag{9}$$

IV. GNO Solution Framework

A. Dynamic Programming Solution

We define a dynamic programming formulation to compute the optimal sequence for GNO. We exploit a special property of our problem: the transition cost from one state to another state *only* depends upon the two states (independent of how we arrived at the current state). Hence we can reduce the number of stages by half by breaking up the dynamic programming formulation for our optimal scheduling problem.

Let $P_k(x, z)$ give the cost of going from state x to state z in k steps. We can, therefore, define the following recurrence relation:

$$P_{2k}(x, z) = \min_{y \in S} \left(P_k(x, y) + P_k(y, z) \right) \tag{10}$$

Eq. 10 implies that the cost of going from state x to state z in $2k$ steps can be obtained by choosing the minimum value from the cost of going from state x to a possible state y in k steps plus the cost to go from state y to state z in k steps. The boundary condition is given by Eq. 11.

$$P_1(x, z) = \begin{cases} \frac{d(x,z)}{B} & \text{if } (x, z) \in \mathcal{A}, \\ \infty & \text{otherwise} \end{cases} \tag{11}$$

The solution of the optimal cost is given by $P_B(s_{initial}, s_{final})$ if B is a multiple of 2. Else, it is given by $\min_{y \in \mathcal{N}(s_{intial})}(d(s_{initial}, y) + P_{B-1}(y, s_{final}))$. In Eq. 11 we divide $d(x, z)$ by B so that, corresponding to Eq. 9, the final cost represents the mean of the individual disruption costs.

Note that the state-space S grows exponentially with the size of the job set $|J|$. The LWRS problem has $2^{|J|}$ states corresponding to the $2^{|J|}$ unique values the progress vector $\mathcal{I}$ can take (Eq. 6). Similarly, the LMS problem has $3^{|J|}$ states corresponding to the $3^{|J|}$ unique values the progress vector $\mathcal{I}$ can take (Eq. 7). Hence the utility of our dynamic programming solution is restricted to those job-sets for which the dynamic programming solution remains tractable. The next section presents an approximation algorithm based on Ants Colony Optimization that can be used as a heuristic for large problem sizes.

B. Ant Colony Optimization based Scheduling

Ant Colony Optimization (ACO) algorithms have been used to produce near-optimal solutions to combinatorial optimization problems, such as the traveling salesman problem [16]. ACO is motivated by the foraging behavior of ants in nature. Ants traveling from nests to food sources deposit a chemical, *pheromone*, along their routes. Ants following them choose routes based upon the deposited pheromone and deposit pheromone themselves along their routes. Shorter pheromone trails get reinforced since the pheromone along less attractive routes evaporates. The ant colony is, therefore, able to converge to the optimal route. A detailed exposition of ACO can be found in [17].

Algorithm 1 gives our algorithm that represents the ACO meta-heuristic adapted for GNO. In each iteration of the outer loop n_{ant} ants independently explore a sequence. This process is detailed in Algorithm 2. We let W represent the set of all atomic operations. Hence, for the LWRS problem, since an atomic operation entails switching a link weight, W contains an operation for each link in the job-set. For LMS, since an atomic operation entails either deactivating or reactivating a link, W contains one activate and one deactivate operation for each link in the job-set. We define the set W^T to represent the set of permissible operations that can be performed next. For LWRS, $W^T = W$. However, for LMS, W^T is defined as having all operations in W except for link reactivation for links whose corresponding deactivation operations are also in W. This is because one can only reactivate a link after it has been deactivated.

Algorithm 1 AOS

initialize pheromone values
$seq_{global} \Leftarrow$ NULL
for $l = 1$ to l_{max} **do**
  $seq_{local} \Leftarrow$ NULL
  for $i = 1$ to n_{ant} **do**
    $seq \Leftarrow$explore()
    $seq_{local} \Leftarrow \min(seq, seq_{local})$
  end for
  update pheromone values based on seq_{local}
  $seq_{global} \Leftarrow \min(seq_{local}, seq_{global})$
end for

Algorithm 2 progressively chooses operations to perform until W is empty. Two values $p(c,x)$ and $k(x)$ are used to guide the choice of the next operation x at any stage. $p(c,x)$ represents the amount of pheromone on the route segment (c,x) where c is the last operation performed. $k(x)$ represents the network disruption cost to go from the current network state to the next state if we carry out operation x (as defined in Section III-B). Initially, all $p(u,v)$ are initialized to a common value. We assign a score to each operation $x \in W^T$ that measures the attractiveness of choosing x to be our next operation. The score is set to $p(c,x) * k(x)^{-\beta}$. It should be evident that an operation $x \in W^T$ has a higher score if the pheromone along (c,x) is high and the associated network disruption cost is low. β is a configurable parameter that determines the relative weight to be given to the pheromone value vs. the network disruption cost. The next operation is chosen to be the operation in W^T with the highest score, with probability q_0. To avoid getting stuck in local optima, the next operation is randomly chosen in proportion to its score, with probability $1 - q_0$.

Algorithm 2 explore

$seq \Leftarrow$ NULL
randomly choose c *from* W^T
append c *to* seq *and remove from* W
while $W \neq \emptyset$ **do**
  $q \Leftarrow$ uniform$(0,1)$
  if $q \leq q_0$ **then**
    $n \Leftarrow \max_{x \in W^T}(p(c,x) * k(x)^{-\beta})$
  else
    $n \Leftarrow n'$ *with probability* $\frac{p(c,n') * k(n')^{-\beta}}{\sum_{x \in W^T}(p(c,x) * k(x) - \beta)}$
  end if
  append n *to* seq *and remove from* W
  $c \Leftarrow n$
end while
return seq

At the end of each iteration of the outer loop in Algorithm 1, the least cost sequence is used to update pheromone values,

$$p(u,v) = p(u,v) * (1 - e_f) + (1/cost(seq_{local})) * e_f \quad (12)$$

where $c(seq_{local})$ represents the cost of seq_{local} and e_f represents the evaporation factor which determines the weight to be given to previous pheromone values. seq_{global} represents the best sequence returned by our algorithm. l_{max}, n_{ant}, q_0, β, and e_f are tunable parameters of our algorithm.

V. Simulation Experiments

A. Simulation Setup

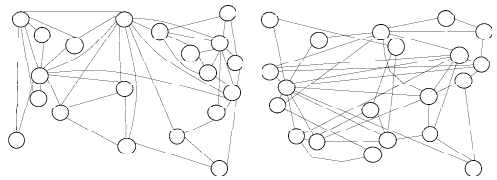
This section describes a detailed simulation study to evaluate the performance of GNO scheduling algorithms.

1) GNO Schemes:

- **Naive Scheduling (NS)**: NS represents the crude heuristic currently employed. In the context of LMS, the NS solution entails failing at most one link at a time. It should be evident from Eq. 9 that, if we fail at most one link at a time, the overall cost is independent of the order in which links are failed. The intuition behind NS is that since the number of simultaneous failures is never greater than one, it would serve to keep the overall network disruption low. In the context of LWRS, NS computes a random permutation of links in the job-set which represents the order in which they are reassigned weights. For LWRS we also define a scheme NS+ that entails computing 10 such random permutations and selecting the one that yields the minimum network disruption cost.

- **Optimal Scheduling (OS)**: OS computes schedules that yield the optimal network disruption cost using dynamic programming as described in Section IV-A. OS is feasible for small problem sizes and we use it to benchmark the performance of our heuristic algorithm.
- **ACO based Scheduling (AOS)**: AOS uses the ACO meta-heuristic as described in Section IV-B.

2) Simulation Topologies: We use three networks for our simulations. The *Abilene* network has 11 nodes and 28 directional links with 10 Gbps capacity. The other two networks are Tier 1 POP-level topologies ISP A and ISP B, shown in Fig. 2. The link capacity for each link is set as 1000 units in each direction, modeling the capacity of OC-192 circuits. There are 110 ingress-egress pairs in the Abilene network, and 190 ingress-egress pairs in each of the other two networks. For generality, our problem formulation in Section III presented the job-set J as a set of unidirectional links. However, in practice failure of a link in one direction implies failure of a link in the other direction [1]. Therefore, in this section we simplify the notation and let the definition of J depend on the context. Specifically, J represents a set of bidirectional links if the problem under consideration is LMS, and it represents a set of unidirectional links when we discuss the LWRS problem.



(a) ISP A (b) ISP B

Fig. 2. Simulation Topologies

3) Synthetic Traffic Matrix Generation: We use three synthetic traffic matrix generation methods employed by [18].

- Gravity Model (GM): The GM model stems from the observed characteristics of PoP-to-PoP traffic matrices in Sprints IP backbone [19]. GM specifies three volume categories for traffic demands [20]. The fan-out of traffic originating at a given node is then determined as per the observations in [19].
- NegativeExponential Model (NegExp): The NegExp model is motivated by studies revealing that a fraction of demands in the Sprint IP Network can be explained by the negative exponential distribution [20]. NegExp, therefore, generates traffic matrix entries according to a negative exponential distribution.
- LogNormal Model (LogNorm): The LogNorm model generates traffic matrix entries according to a log-normal distribution with mean m. [21] finds that LogNorm describes a subset of traffic demands seen in actual networks. The estimated parameters for the distribution from [21] were $\mu = 16.06$ and $\sigma = 1.04$. Therefore, we use log-normal distributions where $\sigma = \frac{1.04}{16.06}\mu$.

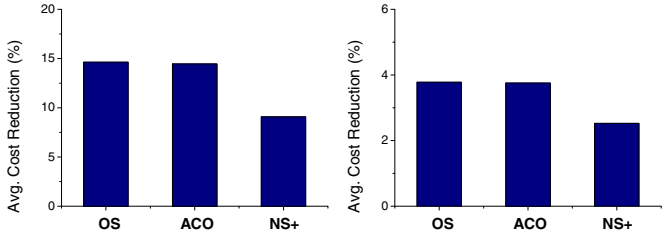
		n_{ant}	β	e_f	q_0	l_{max}
Abilene	**LWRS**	20	3	0.1	0.7	20
	LMS	20	3	0.1	0.7	20
ISP A & ISP B	**LWRS**	20	2	0.3	0.9	20
	LMS	80	2	0.3	0.7	20

TABLE III
ACO PARAMETER SETTINGS

4) ACO Parameter Settings: As mentioned in Section IV-B, l_{max}, n_{ant}, q_0, β, and e_f are the configurable parameters for AOS. We perform sensitivity analysis for each parameter and tune them for the LMS and LWRS problems to ensure fast exploration of the solution space and to avoid convergence to a local optima. We configure the parameters for the smaller Abilene network separately from the two Tier 1 POP-level topologies ISP A and ISP B. Table III lists the parameter settings we used in our experiments.

B. GNO for the Abilene Network

We first conduct a preliminary evaluation using the Abilene network. Our choice of Abilene is influenced by the fact that problem sizes for Abilene are small enough to be solved by OS. We use the GM model to synthetically generate traffic demands for the Abilene network. We set the mean traffic for the GM model such that it yields a maximum link utilization of approximately 33%. We use the well known local-search meta-heuristic proposed in [5, 6] to optimize link weights with respect to a given traffic matrix.



(a) Disruption Metric: MLU (b) Disruption Metric: F&T

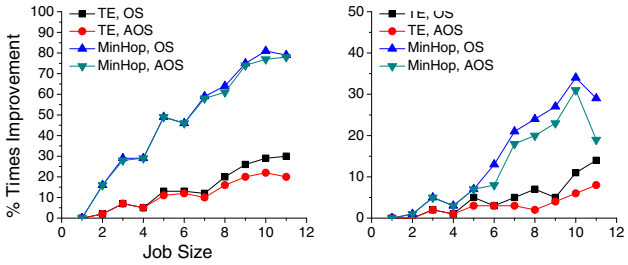
Fig. 3. LWRS (Abilene): Average Cost Reduction (%)

1) Link Weight Reassignment Scheduling (LWRS): We first look at the LWRS problem. An LWRS experiment comprises generating an *initial* and a *perturbed* traffic matrix, that represents how the initial traffic matrix has evolved over time. Let the optimized weight setting for the initial traffic matrix be $w_{initial}$. We introduce random perturbation in the initial traffic matrix by multiplying the demand between each node pair by a fraction uniformly distributed between $1-p_f$ and $1+p_f$, where p_f is the perturbation factor used to model change in traffic demands. For the preliminary evaluation we set $p_f = 0.5$. We optimize the weights for Abilene links with respect to the *perturbed* traffic matrix to get w_{final}. The LWRS problem is to find the optimal schedule to migrate from $w_{initial}$ to w_{final} given that Ω is equal to the perturbed traffic matrix.

Fig. 3 plots the performance of OS, AOS, NS+. The results are averaged over 100 experiments. AOS and NS+ show better

results than NS for all 100 experiments. We use the cost of NS as a benchmark and report the reduction in overall network disruption cost with respect to NS achieved by the other three schemes. Fig. 3(a) plots results using MLU as the disruption metric, while Fig. 3(b) has results for the F&T metric. In either case, we see that all three schemes can reduce the overall disruption cost incurred by NS by $8\% - 14\%$ for the MLU metric, and $3\% - 4\%$ for the F&T metric. We also see that AOS performs very close to OS which represents the optimal schedule.

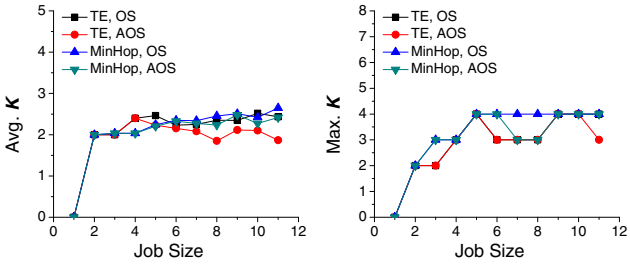
2) Link Maintenance Scheduling (LMS): We now look at the LMS problem. For each LMS experiment, we have a job size that represents the number of bidirectional links that need to be maintained. For a particular job size, we randomly select links to include in the job set. We expect NS to work well for



(a) Disruption Metric: MLU (b) Disruption Metric: F&T

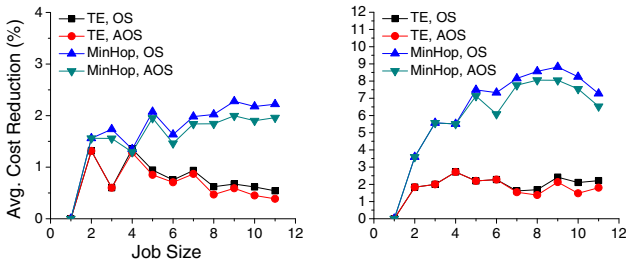
Fig. 4. LMS (Abilene): Times Improvement Over NS (%)

LMS, since it makes intuitive sense that if there is no more than one link deactivated at a time, the resulting performance degradation will be low. What we want to evaluate is whether there exist cases where this intuition fails and a different scheduling yields better results. Fig. 4(a) and 4(b) show the percentage of times OS and AOS result in an improvement over NS for the MLU and the F&T metrics, respectively. The results are the number of improved experiments out of 100 experiments for each job size. We also report the results for *MinHop* routing where instead of optimizing weights as is the case in *TE* routing [5, 6], we assign unit weights to each link. The horizontal axis plots the size of the job-set and the vertical axis gives the percentage of times that a scheme resulted in lower disruption cost than NS. Fig. 4 shows that GNO scheduling schemes yield better results than NS. Furthermore, in almost all experiments where OS registers an improvement over NS, AOS also does so. The improvement is greater for *MinHop* routing, which suggests that the further away the initial weights are from the optimal, the greater is the advantage of employing GNO scheduling. If a GNO schedule represents an improvement over NS, then there will be at least one stage with more than one link deactivated in that schedule. Let κ represent the maximum number of links that are simultaneously down in a schedule. Fig. 5(a) and 5(b) plot the average and maximum κ for OS and AOS, across all the experiments where an improvement over NS is achievable. Similarly, Fig. 6(a) and 6(b) give the average and maximum percentage reduction in network disruption costs of



(a) Average κ (b) Maximum κ

Fig. 5. LMS (Abilene): Simultaneous Deactivations



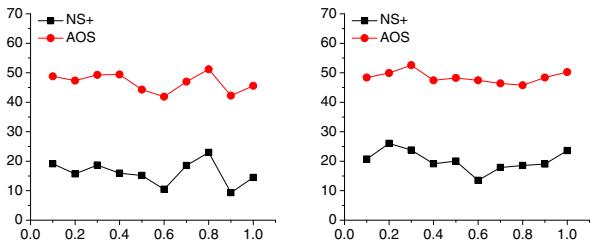
(a) Avg. Cost Reduction (%) (b) Max.Cost Reduction (%)

Fig. 6. LMS (Abilene): Cost Reduction (%)

OS and AOS over NS, across all the experiments where OS results in an improvement over NS. Fig. 5 and 6 use MLU as the disruption metric. We see that the average improvement is small but the maximum improvement can be as much as 7%. Section V-C will show how the improvement is more significant for larger networks and job sizes.

C. GNO for Tier 1 Pop Level Topologies

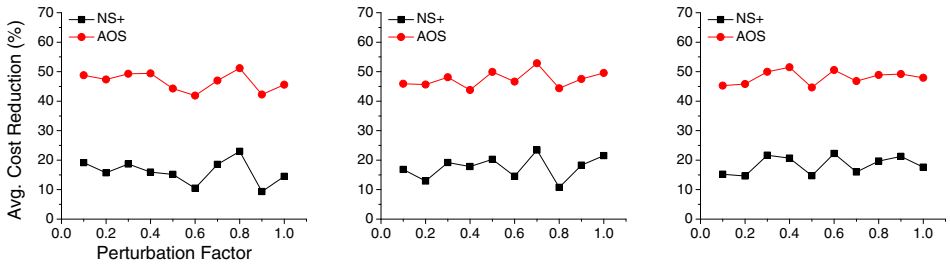
We now evaluate the performance of AOS for our larger networks representing Tier 1 POP level topologies. All subsequent experiments use MLU as the disruption metric.



(a) ISP A (b) ISP B

Fig. 7. LWRS: Average Cost Reduction (%)

1) Link Weight Reassignment Scheduling (LWRS): AOS results in a lower disruption cost than NS and NS+ in 100% of our LWRS experiments. Fig. 7 gives the average reduction in the disruption cost compared to NS for AOS and NS+. We plot results for different values of the perturbation factor p_f. We can observe that AOS results in approximately 50% reduction

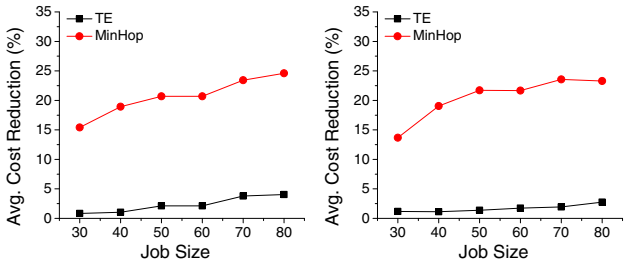


(a) GM (b) LogNorm (c) NegExp

Fig. 8. LWRS (ISP A): Performance for different Traffic Distributions

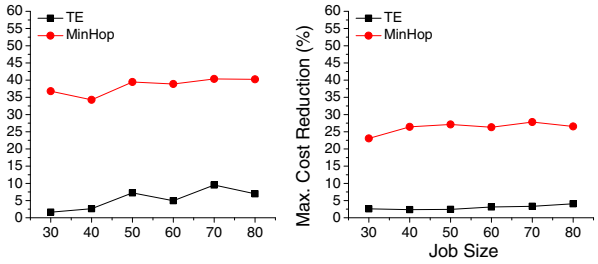
in the disruption cost that would have ensued if we used NS. Also NS+ only yields a cost reduction of around 20%, which is significantly worse than that achieved by AOS. Note that these cost reductions for ISP A and ISP B are greater than what was observed for the Abilene network. This can be attributed to the larger solution space that exists in the case of the larger topologies. Hence, the difference between a random schedule and one computed through an approximation algorithm can be significant.

We also study whether the performance of our GNO schemes is sensitive to the distribution of traffic demands. Fig. 8 shows the performance of AOS for our three different traffic matrix generation models. The LogNorm and NegExp models are configured to have the same mean traffic demand as our GM model. For all three synthetic models, AOS shows a significant reduction in overall disruption cost over NS.



(a) ISP A (b) ISP B

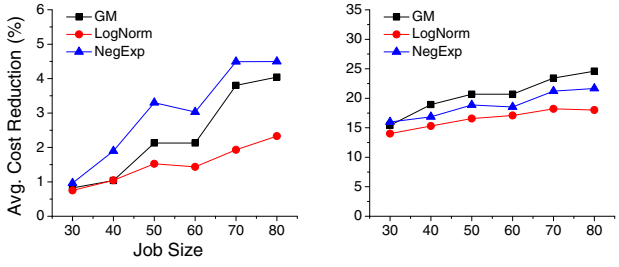
Fig. 9. LMS: Average Cost Reduction (%)



(a) ISP A (b) ISP B

Fig. 10. LMS: Maximum Cost Reduction (%)

2) Link Maintenance Scheduling (LMS): Fig. 9 and 10 show the performance of AOS using our larger topologies for the LMS problem. In all experiments, AOS performs better than NS. We see that AOS can result in modest reductions in the average disruption cost achieved with NS. Consistent with our results for the Abilene network, we see AOS performing much better for MinHop routing as compared to TE routing. The modest average cost reduction for TE routing is expected since we expect NS to perform well for the average case. However, Fig. 10(a) shows that even for TE routing, the difference in disruption costs can be as high as 10%. Fig. 11 and 12 show that like LWRS, the performance gain of AOS persists across different distributions of traffic demands. We see that the maximum cost reduction achieved by AOS can be as large as 17% and 40% for TE and MinHop routing, respectively. Fig. 13 plots AOS performance for different values of the perturbation factor, p_f, which models magnitude of change in traffic demands. We see that the performance improvement of AOS over NS increases with p_f when we employ TE routing. This is consistent with our previous observations which saw greater improvement for MinHop routing as compared to TE routing. We conjecture that if the original weights are not optimal, we witness a greater advantage of employing AOS. With higher p_f the original weights become more sub-optimal with respect to the perturbed traffic matrix and the cost reduction achieved by AOS increases.

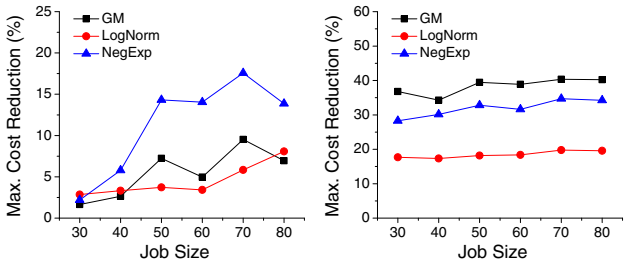


(a) TE (b) MinHop

Fig. 11. LMS (ISP A): Performance for different TM Models

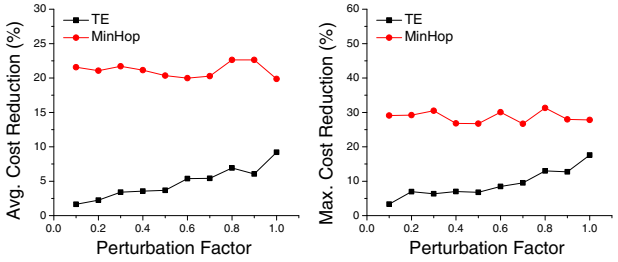
D. Solution Computation Times

Table IV compares the solution computation times of OS and AOS for ISP A. Our algorithms are implemented in Java are run on a PC with a 3 GHz CPU and 3 MB memory, with the maximum Java heap size set to 512 MB. Our



(a) TE (b) MinHop

Fig. 12. LMS (ISP A): Performance for different TM Models



(a) Avg. Cost Reduction (%) (b) Max. Cost Reduction (%)

Fig. 13. LMS (ISP A): Varying Perturbation Factors

implementation is not optimized for running time, and we report the computation times to give a rough picture of their rate of growth as a function of job size. Table IV shows that the average computation time for OS grows much faster than that for AOS, and quickly becomes intractable. We can see that on average it takes 331 times longer to solve an LWRS problem with a job size of 22 using OS as compared to AOS. Similarly the average solution computation time for an LMS problem with a job size of 13 is 6.5 times greater for OS as compared to AOS.

VI. Conclusion

Our work was motivated by the observation that a significant fraction of network failures, topology changes, and parameter configurations stem from deliberate and premeditated management and operational tasks. Since network operators have the prerogative to decide the exact sequence of such operations, judicious scheduling can minimize network performance disruption. We formulated GNO as a general class of problems that study how to migrate from an initial to a final network state by executing a series of atomic operations. We presented LWRS and LMS, two problems routinely encountered by network operators, as case studies for GNO. We also presented an Ants Colony Optimization inspired heuristic for LWRS and LMS. Our simulation study demonstrates that our solution delivers major improvements over current practices for LWRS. For the LMS problem we saw that, although current practice perform well on average, there exist cases where the difference between current practice and our solution can be significant. We believe that a major strength of this work is the generality of the GNO problem formulation and solution framework. It is independent of the underlying network and link layer mechanisms, and can incorporate different metrics for network disruption. We speculate that extending the GNO framework to deal with a host of similar problems within different operational contexts is a rich area for future work.

Job Size	2	6	10	13	20	22	50	80
LWRS (OS)	0.02	0.1	0.7	5	924	6624		
LWRS (AOS)	0.3	2	5	7	16	20	93	234
LMS (OS)	0.04	0.2	4	144				
LMS (AOS)	0.7	4	13	22	43	58	316	523

TABLE IV
Solution Computation Times (Seconds)

References

[1] A. Markopoulou, G. Iannaconne, S. Bhattacharrya, C. N. Chuah, and C. Diot, "Characterization of failures in an IP backbone," in *IEEE INFOCOM*, March 2004.
[2] N. Dubois, B. Fondeviole, and N. Michel, "Fast convergence project," presented at RIPE47, www.ripe.net/ripe/meetings/ripe-47/presentations/ripe47-routing-fcp.pdf, January 2004.
[3] J. Moy, "OSPF Version 2," RFC 2328, Apr. 1998.
[4] Dave Oran, "OSI IS-IS Intra-domain Routing Protocol," RFC 1142, Feb. 1990.
[5] B. Fortz and M. Thorup, "Internet Traffic Engineering by Optimizing OSPF Weights," in *IEEE INFOCOM*, March 2000.
[6] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional IP routing protocols," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 118–124, 2002.
[7] A. Nucci, B. Schroeder, S. Bhattacharyya, N. Taft, and C. Diot, "IGP Link Weight Assignment for Transient Link Failures," in *ITC*, 2003.
[8] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 4, pp. 756–767, 2002.
[9] P. Francois and O. Bonaventure, "Avoiding transient loops during IGP convergence in IP networks," 2005.
[10] S. Lee, S. Nelakuditi Y. Yu, Z-L. Zhang, and C. N. Chuah, "Proactive vs. reactive approaches to failure resilient routing," in *IEEE INFOCOM*, March 2004.
[11] Z. Zhong, S. Nelakuditi, Y. Yu, S. Lee, J. Wang, and C. N. Chuah, "Failure Inferencing based Fast Rerouting for Handling Transient Link and Node Failures," *Global Internet Symposium, Miami*, 2005.
[12] S. Bryant and M. Shand, "a framework for loop-free convergence," IETF Draft, December 2006.
[13] R. Teixeira and J. Rexford, "Managing routing disruptions in internet service provider networks," *IEEE Communications Magazine*, March 2006.
[14] Pierre Franois, Mike Shand, and Olivier Bonaventure, "Disruption-free topology reconfiguration in OSPF networks," in *IEEE INFOCOM*, May 2007.
[15] M. Caesar and J. Rexford, "BGP routing policies in ISP networks," *Network, IEEE*, vol. 19, no. 6, pp. 5–11, 2005.
[16] J. L. Bentley, "Fast algorithms for geometric traveling salesman problems," in *ORSA J. Comput.*, 1992.
[17] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," in *IEEE Trans. Evol. Comput*, April 1997.
[18] Antonio Nucci, Supratik Bhattacharyya, Nina Taft, and Christophe Diot, "IGP link weight assignment for operational tier-1 backbones.," *IEEE/ACM Trans. Netw.*, vol. 15, no. 4, pp. 789–802, 2007.
[19] A. Medina, N. Taft, S. Battacharya, C. Diot, and K. Salamatian, "Traffic matrix estimation: Existing techniques compared and new directions," in *ACM SIGCOMM*, Aug. 2002.
[20] S. Bhattacharyya, N. Taft, J. Jetcheva, and C. Diot, "POP-level and access-link-level traffic dynamics in a tier-1 POP," in *Proceedings of 1st ACM Sigcomm Internet Measurement Workshop (IMW)*, Nov. 2001.
[21] Antonio Nucci, Ashwin Sridharan, and Nina Taft, "The problem of synthetically generating ip traffic matrices: Initial recommendations," in *Computer Communications Review*, July 2005.