Planning the Transformation of Network Topologies

Young Yoon[1], Nathan Robinson[2], Vinod Muthusamy[*3], Hans-Arno Jacobsen[1], and Sheila A. McIlraith[2]

[1]Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada
[2]Department of Computer Science, University of Toronto, Toronto, Canada
[3]IBM T.J. Watson Research Center, Hawthorne, NY, USA

Abstract

Refining a network topology is an important network management technique. Nevertheless, determining the appropriate steps to transform a network from one topology to another, in a way that minimizes service disruptions, has received little attention. This is a critical problem since service disruptions can be particularly harmful and costly for networks hosting mission-critical services. In this paper, we introduce the incremental network transformation (INT) problem and explore this problem in the context of automated planning. We define two metrics to measure the quality of generated transformation plans, one of which is amenable to classical propositional planning. We find that while state-of-the-art domain-independent planning techniques are effective at finding high-quality solutions for small problem instances, they cannot scale to solve realistically sized INT instances. To address the shortcomings of existing approaches, we developed a number of domain-dependent planners that use novel domain-specific heuristics. We empirically evaluated our planners on a wide range of synthetic network topologies. Our results illustrate that our automated planning inspired techniques are effective on realistically sized INT problems. We envision that our approach could eventually provide a compelling addition to the arsenal of techniques employed by network practitioners to support network refinement with minimal disruption to running services.

1 Introduction

To maintain high quality of service for the end-users of a messaging system in a dynamic environment, it may be necessary to frequently reconfigure the underlying network topology. For example, suppose we have a network for publish/subscribe-based messaging system (Jacobsen et al. 2010; Carzaniga and Wolf 2003), as shown in Figure 1. Brokers (B) are interconnected to route publication messages from publishers P_1 and P_2 to subscribers S_1 and S_2, who are interested in the messages. Assume that S_1 and S_2 demand more timely delivery of messages. This demand may be met by reducing the average length of the paths over which the messages produced by P_1 and P_2 must travel. This reduction can be achieved by reconfiguring the topology, as shown in Figure 1. In this case, new routing paths B_0-B_4 and B_5-B_6 are established, while B_1-B_2 and B_2-B_3 are disconnected.

[*]This majority of this work was done while the author was at the University of Toronto.

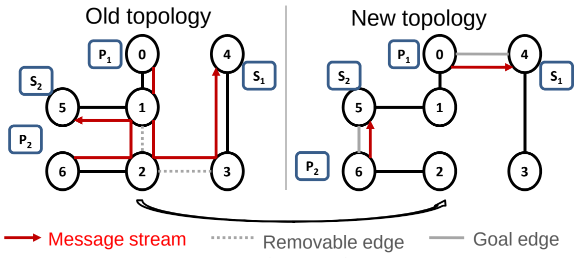


Figure 1: Example of reconfiguring a topology

Reconfiguring a network topology is an issue seen in many real-world distributed messaging systems. For example, the need for frequent reconfiguration is observed in GooPS, a messaging substrate for their public services (Reumann 2009) and Yahoo! PNUTS that provides a large-scale distributed data dissemination service (Cooper et al. 2008). OpenFlow (McKeown et al. 2008) supports rewiring of virtualized networks.

To address the needs from the enterprise examples, the academic literature suggests various algorithms to devise new network topologies. These algorithms incorporate many different optimization criteria, including minimizing the number of nodes in the network to reduce cost, optimizing path lengths or network latencies, controlling node degrees, and providing sufficient processing and network capacities (Baldoni et al. 2004; Jaeger et al. 2007; Chockler et al. 2007; Migliavacca and Cugola 2007; Onus and Richa 2011; Liben-Nowell, Balakrishnan, and Karger 2002; De Santis, Grandoni, and Panconesi 2007; Lau et al. 2007). However, these works do not deal with the practical issue of transforming an existing deployed network from one topology to another during runtime with minimal disruption.

Considering this, we develop a generic transformation framework that can determine the appropriate steps to evolve a network into a new topology (derived offline by the various algorithms suggested in the aforementioned academic works). The key objective of this framework is to produce a transformation plan in a timely manner that minimizes the disruption to the services. Such a plan can be considered as a sequence of atomic operators, each of which adjusts a single edge in the network in a way that maintains the integrity of the message streams of the network services. We call this the incremental network transformation (INT) problem.

We pose the problem of finding such a transformation as an automated planning problem. We construct a PDDL encoding of the problem, which we call *network*. The initial state of the problem is the initial graph representation, the goal defines the features of the optimized topology, and in particular a certain configuration of connections between nodes. Operators in the problem represent different transformation operations. The quality of a plan can then be defined in terms of the number of transformation operations or another metric that captures, for example, disruption caused to messages traversing the network. As we will see in the paper, *network* presents an interesting challenge problem to the International Planning Community. Its interesting features include its large size and the fact that, in its current formulation, it lacks dead-ends.

We experimented with several state-of-the-art domain-independent planning systems, in particular LAMA (Richter and Westphal 2010) and PROBE (Lipovetzky and Geffner 2011), and found that they were unable to solve PDDL encodings of our problems effectively. The large size of grounded representations of our problems, which most modern planning systems operate on, prevented these approaches from working effectively. Yet, we feel that we can leverage the considerable recent advances in automated planning to inform solutions to our problem. Therefore, based on our knowledge of the workings of existing planning approaches, we developed a suite of domain-dependent planners that use novel domain-specific heuristics.

We empirically evaluated the planning systems we developed on a wide range of generated network topologies. With such tools, we envision that the administrators of the messaging system can more easily adopt various network topology refinement techniques while minimizing the disruption to running services.

The contribution of our work is outlined as follows. First, in Section 2 we define the INT problem. We then map this problem to automated planning in Section 3.2 and discuss the performance of existing planning systems on such problems in Section 3.3. Next, we present our novel domain-specific planning systems in Section 4 and evaluate them empirically in Section 5. Finally, we conclude in Section 6.

2 Problem Definition

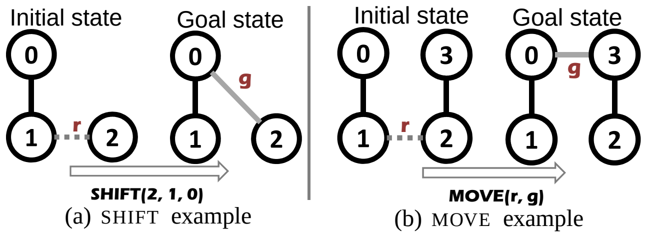
2.1 Network Topology Transformation

We formally introduce the INT problem. Here, we abstract a network topology as a connected undirected acylic graph. Formally, a topology T is a pair (V, E), where V is a set of vertices and E is a set of edges. An edge $e_x \in E$ is a pair of distinct vertices $e_x(v_i, v_j)$, where v_i and $v_j \in V$. A topology transformation problem $\mathcal{T}$ is a tuple $\langle T_S, T_G, O \rangle$, where we want to transform the initial topology T_S into the goal topology T_G using the transformation operators in O. We assume that the environment where the topology operates is static long enough to allow for the determination and execution of a transformation.

Given such a problem $\mathcal{T}$, we can derive the set of *removable edges* R, that are in T_S, but not in T_G, the set of *stationary edges* ST, that are in both T_S and T_G and the set of *goal edges* G that need to be realized in T_G. We now need to define the transformation operations O that are used to remove the edges in R and add those in G.

2.2 Transformation Operations

We use an *incremental* transformation approach with atomic composite-transformation operations that maintain the structural properties of the topologies and minimize the disruption to network services. We first employ SHIFT(v_i, v_j, v_k) (Yoon, Muthusamy, and Jacobsen 2011). As shown in Figure 2(a), SHIFT(v_i, v_j, v_k) replaces the edge between v_i and v_j with one between v_i and v_k, where $\{v_i, v_k\} \in \mathbb{N}(v_j)$ and $v_i \neq v_k$. Here $\mathbb{N}(v_x)$ is the set of neighbors of v_x.



(a) SHIFT example (b) MOVE example

Figure 2: Example of topology transformation operations

Besides the SHIFT operator, we introduce MOVE. MOVE(r, g) replaces the goal edge $g \in G$, with the removable edge $r \in R$, as shown in Figure 2(b). Both SHIFT and MOVE can be thought of as compositions of DELETE and ADD operations that we assume to be executed atomically.

These operations should follow the following rules. Denoting the path between the two nodes (v_x and v_y) that consist of a goal edge g to be $\overrightarrow{p}(g)$ ($\equiv \overrightarrow{p}(v_x, v_y)$), only the removable edge on $\overrightarrow{p}(g)$ is involved in a transformation a. Also, a stationary edge is not considered for a transformation a. This rule is to avoid redundant or irrelevant transformation operations.

2.3 Solution Quality

Informally, a hiqh quality solution minimizes service disruption. If a network topology is in use, then there are streams of messages flowing through the network to serve end-users. We formally represent a message stream and its flow between the producer v_p and consumer v_c of the message as PCPAIR(v_p, v_c).

We illustrate a case where a message stream can be disrupted in Figure 3. Suppose, a message stream exists between v_p and v_c in the initial graph. When the removable edge between v_0 and v_1 is taken down by a DELETE operation, temporarily the vertices v_p, v_0 and v_1 must cease forwarding messages sent from v_p, otherwise the messages through $v_p \rightarrow v_0 \rightarrow v_1 \rightarrow v_c$ may get lost. While the goal edges are being connected in the subsequent step, routing states on the vertices, v_p, v_0 and v_1 need to be updated in order to correctly re-route the message stream through the new desired path $v_p \rightarrow v_c$. From this example, we can see that the transformation operations can disrupt message streams. Message delivery may have to be paused, or messages can get lost or be delivered in the wrong order.

To formalize network transformation quality, we consider

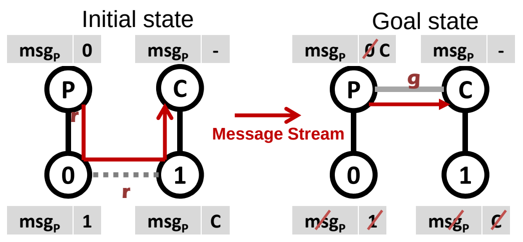


Figure 3: Example of a message stream disrupted while routing states (*message ID* and *next hop destination*) are updated during the transformation of a graph

both the number of steps in the transformation and the degree of impact to the message streams flowing during the transformation. These two factors apply generically to any acyclic topology regardless of the routing protocols running on top. In case of cyclic topologies, issues such as the routing decision given alternative paths affect disruption. We plan to investigate this more general case in future work.

3 Topology Transformation as Planning

An INT problem can naturally be encoded as an automated planning problem. Such a transformation is intuitively appealing as it means that recent advances in automated planning can be leveraged. In this section, we briefly describe the automated planning problem and techniques and propose a translation of the INT problem into a planning problem. We also evaluate some existing automated planning systems to assess whether they are suitable to solve our problem.

3.1 Propositional Planning

Formally, a propositional planning problem Π is a tuple $\langle \Lambda, \Sigma, \mathcal{O}, s_0, \mathcal{G}, \mathcal{C} \rangle$. Here, Λ is a set of predicates with typed arguments and Σ is a set of typed objects. The grounding of Λ with Σ, respecting types, creates a set of propositional variables $\mathcal{P}$. Π then has the set of states $\mathcal{S}$, where each $s \in \mathcal{S}$ is a valuation of the variables in $\mathcal{P}$, represented as the set of variables it assigns to *true*. $s_0 \in \mathcal{S}$ is the start state and $\mathcal{G} \subseteq \mathcal{S}$ is a set of propositions representing a set of goal states. A state $s \in \mathcal{S}$ is a goal state *iff* $s \subseteq \mathcal{G}$. $\mathcal{O}$ is a set of first-order STRIPS-like planning operators of the form $o = \langle O(\overrightarrow{x}), pre(o), add(o), del(o) \rangle$, where $pre(o)$, $add(o)$, and $del(o)$ are the precondition, add, and delete lists (resp.) of o. An action a is a grounding of an operator $o \in \mathcal{O}$ with the objects in Σ, respecting types. It has a precondition, and add and delete lists specified as sets of propositions in $\mathcal{P}$. Let $\mathcal{A}$ be the set of actions produced by grounding the operators in $\mathcal{O}$ with Σ. Finally, $\mathcal{C}$ is a function that assigns actions in $\mathcal{A}$ a non-negative cost. Propositions in s_0 that are not deleted by any action are *static* and can be omitted from the problem. Other propositions are referred to as *fluents*.

A solution to a planning problem Π is a plan π. In this setting, π is a sequence of actions $\langle a_0, ..., a_h \rangle$, such that there is a sequence of states $\langle s_0, ..., s_h, s_{h+1} \rangle$, where state s_0 is the initial state, $s_i \models pre(a_i)$ and a_i produces s_{i+1} when applied in s_i, and finally, state $\mathcal{G} \subseteq s_{h+1}$. The quality of a plan can be measured in a number of ways. Commonly, plans are preferred that have a low cost, where the cost of a plan is the sum of the costs of the actions that it contains, as defined by $\mathcal{C}$. A more general formulation that we do not explore in detail here is preference-based planning (Baier and McIlraith 2008), which allows preferences to be expressed over features of the state trajectories implied by plans, often expressed as weighted temporal logic formulae.

3.2 Encoding The Transformation Problem

We now formally define the *network* planning domain.[1] Let $\Pi_\mathcal{T} = \langle \Lambda_\mathcal{T}, \Sigma_\mathcal{T}, \mathcal{O}_\mathcal{T}, s_{0\mathcal{T}}, \mathcal{G}_\mathcal{T}, \mathcal{C}_\mathcal{T} \rangle$ be an encoding of the INT problem $\mathcal{T} = \langle T_S, T_G, O \rangle$, where O contains a SHIFT operator. Here, we assume all objects and variables are of the same type, so we omit types from our description.

First, we have the following set of predicates: $\Lambda_\mathcal{T} = \{conn(v_i, v_j), not\text{-}eq(v_i, v_j), rem(v_i, v_j)\}$, where $conn$ represents that vertices v_i and v_j are connected, $not\text{-}eq$ that they are not equal, and rem that they can be disconnected. Next, if $V_\mathcal{T}$ is the set of vertices in the graphs in $\mathcal{T}$, then we have the set objects $\Sigma_\mathcal{T} = V_\mathcal{T}$. $\mathcal{O}$ contains the following operator: SHIFT(v_i, v_j, v_k), where:

$$\begin{aligned} pre(a) = &\ \{conn(v_i, v_j), conn(v_j, v_k), rem(v_i, v_j), \\ &\ not\text{-}eq(v_i, v_j), not\text{-}eq(v_j, v_k)\} \\ add(a) = &\ \{conn(v_p, v_2), rem(v_i, v_k)\} \\ del(a) = &\ \{conn(v_p, v_1)\} \end{aligned}$$

The start state $s_{0\mathcal{T}}$ contains a $conn$ fluent for every edge in T_S, a rem fluent for every edge in T_S that is not in T_G and one predicate $not\text{-}eq(v_i, v_j)$ for every pair of distinct vertices v_i and v_j in $V_\mathcal{T}$. $\mathcal{G}_\mathcal{T}$ contains one $conn$ proposition for every edge in T_G that is not in T_S. Finally, there are numerous ways to define the cost function $\mathcal{C}$ that are problem-specific. Here we simply assume that all SHIFT actons have cost 1. The removal of static propositions means that all $conn$ propositions representing edges that are in T_S and not in T_G, all initial rem propositions, and all eq propositions will be static and not required in states or actions.

In the propositional planning formalism we have defined here, the definition of the MOVE operator described in Section 2 is problematic. A move operator MOVE(v_i, v_j, v_k, v_l) would delete an edge (v_i, v_j) and add an edge (v_k, v_l) and requires that (v_i, v_j) is located on the path between (v_k, v_l). The preconditions used by classical propositional planning makes the encoding of this constraint problematic. To allow the use of the MOVE operator and to allow plan quality to be determined by the number of routing state updates, rather than by the number of SHIFT actions, we write custom planning algorithms and cannot use the generic algorithms that will be discussed in the next section.

3.3 Performance of Existing Planners

We used the encoding described in the prevous subsection to generate a set of planning problems from randomly generated INT problems. These problems used unit cost SHIFT operators and were generated randomly as described in Section 5. We found that the state-of-the-art domain-independent planning systems we tried are unable to find

[1]A full PDDL definition of the *network* domain and a link to example problem files are included in an appendix in the online version of this paper.

plans for PDDL encodings of even relatively small instances of our problem. For example, neither LAMA (Richter and Westphal 2010) nor PROBE (Lipovetzky and Geffner 2011) were able to find plans for problems with more than 50 nodes, where 50% of edges were changed between the start and goal graphs, within 10 minutes on a machine with 16 GB of memory and a Intel Xeon 3.00 GHz processor. On a problem with 30 nodes and 10% of edges changed, PROBE was able to find a plan with 8 actions in 3.64s, while LAMA took 18.33.

While these two algorithms perform very well on some classes of planning problems, the INT problem presents them with several significant difficulties. First, a INT problem has $O(n^3)$ actions, where n is the number of vertices in the topology. For realistically sized instances of our problem, which have hundreds to thousands of nodes, this overwhelms both LAMA and PROBE, which ground operators to produce all actions prior to beginning search. Another issue that LAMA in partcular faces here is that there are many symmetries in the search space. For example, the states reached by SHIFT(v_i, v_j, v_k) and SHIFT$(v_j,\ v_i,\ v_k)$ often have the same value. This is less of a problem for PROBE, which breaks these symmetries by heuristically computing state trajectories, that it calls probes.

In principle, other existing planning algorithms should have more success with this problem, particularly those that do not completely ground the problem up front. However, we experimented with a number of planners and were not able to find one that performed better than the two algorithms described above.

4 New Planning Techniques

This section presents novel planning algorithms for the incremental INT problem.

4.1 Greedy Planner

The first algorithm finds plans using both SHIFT and MOVE actions. MOVE actions can cause a relatively higher number of routing state updates to occur. More fine-grained control of the disruption at each step in a plan can be achieved by realizing a MOVE action with multiple SHIFT actions, each of which involves only 3 nodes. Algorithm 1 (GREEDY) specifies how this realization is achieved.

Algorithm 1: Greedy Planner

```
  Input: A sequence MOVE operations, $\pi$; a set of goal edges, $G$
  Output: A sequence of SHIFT operations
1 for MOVE$(r, g) \in \pi$ do
2     Apply a SHIFT operation ($a$) on $r$;
3     for $\forall g' \in G - g$ do
4         if path $\overrightarrow{p}$(g') has to change as the effect of $a$ then
5             Search new $\overrightarrow{p}$(g');
6             if $r$(g') not on $\overrightarrow{p}$(g') then
7                 Find a new $r$(g');
8     Repeat 2-7 until $g$ is realized;
```

The input to GREEDY is a sequence of MOVE actions. This sequence is obtained by finding a unique pairing between the goal edges and the removable edges. Given N goal edges and A average number of removable edges associated to each goal edge, the search space for finding the unique pairing is $O(2^{AN})$. We solve this pairing problem by encoding it as a propositional SAT problem.

Suppose we have a set of goal edges G and removable edges R, such that for each $g \in G$, the set $R(g) \subseteq R$ is the set of removable edges on the path $\overrightarrow{p}(g)$ between the vertices in g. Then we can encode this problem in the following way. For each $g \in G$ and $r \in R(g)$, we have a variable $g : r$ in β. For each goal edge $g \in G$, we have a clause $\bigvee_{r \in R(g)} g : r$. For each removable edge $r \in R$, and unordered pair of distinct goal edges $g_1, g_2 \in G$ such that $r \in R(g_1)$ and $r \in R(g_2)$, we have a clause $(\neg g_1 : r \vee \neg g_2 : r)$. These clauses simply state that each goal edge must be assigned some removable edge, and each removable edge can be assigned to at most one goal edge. We solved the generated instances with PRECOSAT (Biere 2010), as it performed particularly well in this case.

Given the sequence of MOVE actions, which removable edge to SHIFT towards the goal edge is also pre-determined. This significantly reduces the number of actions to consider in each intermediate state during planning. Furthermore, as the trajectory of the incremental movements of the removable edges are predetermined by the MOVE sequence, the total number of actions in a plan can be expected to be kept small as well.

Note that this planner is subject to a conflict that occurs if $\overrightarrow{p}$ and the set of removable edges associated with some goal edge change as the add effect of applying an action for some other goal edge. Such a conflict is highly possible, especially when the $\overrightarrow{p}$s overlap with each other. A conflict must be resolved, otherwise the transformation problem falls into an invalid state. The resolution of the conflict is described in line 1:4-7. That is, the new $\overrightarrow{p}$ of a goal edge is newly identified and a removable edge on that $\overrightarrow{p}$ is assigned to the goal edge for the consideration of SHIFT operations in the subsequent states.

4.2 Best-First Search

This algorithm (BFS) is a best-first-search and is similar to the algorithm underlying LAMA and PROBE. Unlike these existing algorithms, it does not ground all actions upfront and uses a domain-specific heuristic. Before we outline the algorithm, we must define some required data structures. Let CL, called the closed list, be a map from states to a list of properties $\langle go, hval, a, closed \rangle$. Here, go is the number of steps to reach s, $hval$ is the heuristic estimate from s, a is the action that was executed to reach s and $closed$ is a flag indicating if the state has been expanded. Let OL, called the open list, be an ordered queue of the following lists $\langle s, go, hval, a \rangle$, where s is a state and the other entries are as previously defined. OL is ordered so that the list with the lowest $go + hval$ is at the front, breaking ties randomly.

In each iteration this algorithm expands the best state from the open list that has either so far not been expanded or has been reached more cheaply than the last time it was expanded (Line 6). When a state is expanded (Line 8), the actions that can be applied are generated.These actions are used to produce successor states. If a successor state is

Algorithm 2: Best-First Search

```
Input: Planning encoding $\Pi_{\mathcal{T}}$ of a problem $\mathcal{T}$, time limit
       $t_{lim}$, state limit $s_{lim}$
Output: A sequence of SHIFT actions
1  $CL[s_{0\mathcal{T}}] = \langle 0, h(s_{0\mathcal{T}}), -, 0 \rangle$;
2  push $\langle s_{0\mathcal{T}}, 0, CL[s_{0\mathcal{T}}].hval, - \rangle$ on $OL$;
3  $bestscore = inf$, $bestplan =$ empty;
4  while $OL$ not empty and run time $< t_{lim}$ do
5      if size of $CL > s_{lim}$ then restart at line 1;
6      get next state list $sl \in OL$ s.t. $(CL[sl.s].open$ or
       $sl.g < CL[sl.s].g)$ and $sl.g < bestscore$;
7      if no $sl$ or $\mathcal{G}_{\mathcal{T}} \subseteq sl.s$ then break;
8      Generate all valid SHIFT actions for $sl.s$, $\mathcal{A}(sl.s)$;
9      for $a \in \mathcal{A}(sl.s)$ and the resulting successor state $s'$ do
10         if $s' \notin CL$ or $sl.g + 1 < CL[s'].g)$ and
           $sl.g + 1 < bestscore$ then
11             $CL[s'] = \langle sl.g + 1, h(s'), a, 1 \rangle$;
12             push $\langle s', sl.g + 1, CL[s'].hval, a \rangle$ on $OL$;
13 if last state $sl.s$ was a goal state then
14     extract $\pi$ from $sl.s$;
15     $bestplan = \pi, bestscore =$ cost of $\pi$;
16     restart at line 1;
17 return $bestplan$;
```

open, or has been reached more cheaply than the last time it was expanded, then it is evaluated according to the heuristic function (Line 11) and added to the open and closed lists. The algorithm restarts whenever a plan is found, or s_{lim} states are on the closed list. This restarting, with the fact that states are ranked on the open list with ties broken randomly, encourages the algorithm to explore different parts of the search space. The algorithm terminates after a specified time out has elapsed t_{lim}.

The heuristic function $h(s)$ measures a goal distance for a state s by assigning removable edges in s to unsatisfied goal edges such that goal edges are assigned removable edges greedily, according to distance. The distance from a removable edge r to a goal edge g is computed as the number of SHIFTs it would take to move r to g. Note that, as discussed in Section 2, a removable edge can only be moved to a goal edge, if it is currently on $\overrightarrow{p}$ of a goal edge. This heuristic function is not admissible as removable edges are assigned to goal edges greedily.

4.3 Discussion

The objective functions we use either assume that all transformations on a network have the same cost, or that there is a static volume of traffic flowing over a network, as described by a set of message streams. On most networks it is unrealistic to assume a fixed pattern of network traffic. In this case, one can assume that the properties of the traffic on a network are static over short intervals, in which case the network optimization and transformation process can be carried out regularly, as needed.

In the future, we plan to extend our work in a number of ways to more realistically and flexibly deal with the transformation of real world networks. One such way is allowing concurrent execution of network transformation actions.

We also plan to allow the specification of temporal constraints on which traffic is acceptable to be disrupted. For example, assume a slightly different case than the example in Figure 3, where a single message stream flows on $\overrightarrow{p}(v_0, v_1)$. If the volume of the message stream drops sharply after time t_1, then the transformation operation to achieve g_1 better be planned to take action after t_1, so that major disruption to the message stream before t_1 can be avoided. Besides the temporal constraints above, preferences can be specified as well. For example, a network practitioner may prefer a big one-time disruption that involves a large number of vertices to many short disruptions over a longer period of time. Also, such a preference can be more precisely quantified.

Another important area of future research is the integration of the network optimization and transformation procedures. For example, the difficulty of a certain transformation may affect decisions regarding how to optimize a network. Options for doing this are integrated in existing planning frameworks. For example, the planning goal specification we have used throughout this paper allows the specification of a partial goal, that represents a set of goal states. A goal can be used to specify a set of hard constraints and then an appropriate objective function can be defined to express preferences over the resulting goal states that takes into account the cost of achieving each.

5 Evaluation

In this section, we briefly summarise the results of the extensive evaluation we carried out on our developed planning algorithms. We simulated INT problems on a Dell PowerEdge 2900 with two quad-core 3 GHz processors and 16 GB of RAM. We have devised a tool that can generate INT problems according to various network parameters.

With this tool we generated a set of transformation problems to evaluate: (1) the efficiency of the planning algorithms, and (2) the quality of the plans found by our algorithms. The experimental setup is as follows. We vary the number of nodes in the start and goal graphs from 20 to 400. The degree of change in network edges varies between 10% and 60%. All generated graphs had a maximum degree of 5 and a maximum path length of 15 and were generated with a fixed set of parameters controlling, for example, the distribution of node degrees. These parameters were set to values that produced realistic network topologies with up to 400 nodes. Interestingly, we have not found any meaningful correlation between planning time and the problem characteristics other than network size and the degree of change.

5.1 Solution Time

Figure 4 shows the time it took for our algorithms to obtain a plan with varying network size and amount of change. Both axes of Figure 4 are log scaled. GREEDY found a plan in 1 ms for transforming a 20-node network with 10% change. For a larger network with 400 nodes, it took about 0.1 seconds to find a plan to transform 10% change. As we increase the degree of change, the overhead increases as well. For a 20-node network with 60% change, GREEDY took about 70 ms to find a plan. For a 400-node network with the same degree of change, it took about 8 seconds to find a plan.

The BFS algorithm, which attempts to find a higher-quality plan, is considerably slower than GREEDY. For a

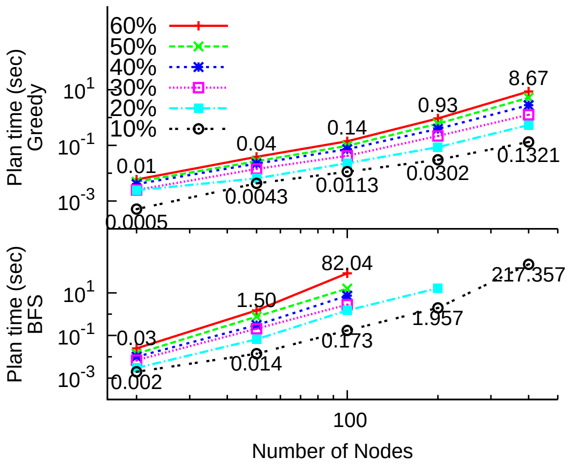


Figure 4: Solution time vs network size with varying degrees of network change

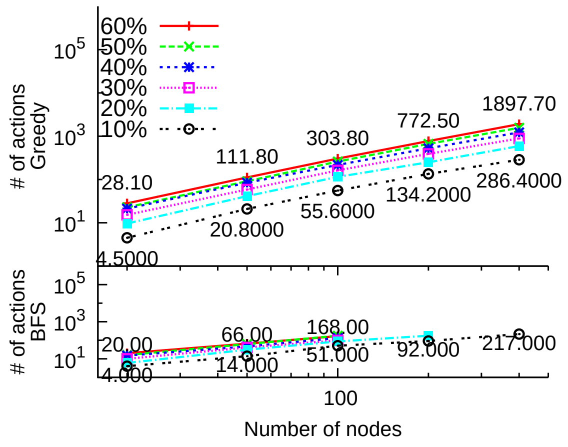


Figure 5: Plan quality vs network size with varying degrees of change

20-node with 10% change, BFS took about 2 ms to find a plan. For a 400-node with 10% change, it took about 3 minutes to find a plan. As we increase the degree of change, the planning time increases significantly. With a 30% change, the planner could not find a solution within the preset timeout of 10 minutes. For a 100-node graph with 60% change, BFS took about a minute to find a plan. This increase of runtime was expected as the algorithm has to search a larger space to find a better plan.

On the INT problem our planners significantly outperform the domain-independent planners that we experimented with, which could not find a plan for any problem with more than 50 nodes with 50% change.

5.2 Plan Quality

In this section, we discuss the quality of the plans that are generated by our planners. The quality of a plan is measured as the number of actions in a plan and the number of routing state updates the plan incurs.

Figure 5 shows the number of actions in a plan that is found by GREEDY and BFS. The graph properties are the same as in the previous section. In general, the length of the plans generated by BFS are somewhat shorter than the plans generated by GREEDY. For example, for a 20-node topology with 10% change, both planners yielded an average of 4 actions in the plan. For a 100-node topology with 30% change and a 400-node topology with 10% change, GREEDY yielded about 63% and 36% more actions than what BFS yielded, respectively. However, we note that this higher plan quality is balanced by the faster planning time of GREEDY. For example, for the 100-node topology with 30% change, BFS found the first plan in a minute, whereas GREEDY found a plan in just 0.13 seconds on average. In general, GREEDY did not exceed 0.24 seconds for finding a plan.

6 Conclusion

In this paper, we introduced the INT problem and discussed the challenges of transforming a network topology to a reconfigured topology, without stopping services that are running on the network. Our objective is to make the transition to a goal topology incremental, to minimise the disruption to these services. We modelled the disruption in the network and defined the objective functions that quantify this. We have explored the transformation process in an automated planning framework and introduced the *network* planning domain. We highlighted shortcomings of the existing generic planners with respect to this problem. To address these shortcomings, we developed domain-specific heuristics and incorporated these into a number of novel domain-specific planning systems. These planning systems significantly outperform existing planners on a suite of automatically generated problems and are able to quickly find high quality solutions to realistically sized problems. We envision that with our planning systems, network topology optimization work studied in the academic literature can become practically useful, as they can be reliably and efficiently deployed in runtime while minimizing the disruption to network services.

Nevertheless, there are a few formidable, yet intriguing challenges that remain to be addressed. We believe that our objective functions can be further developed to more accurately reflect the real world costs of transforming a network topology. Also, future systems should have the flexibility to respect restrictions imposed by network practitioners on the transformation of networks. For example, limits may be imposed on the node degrees, path lengths and other network characteristics during the intermediate steps of a transformation plan. As another research avenue, we can develop systems that can parallelize a plan. Moreover, future planning systems can work in concert with the network optimization frameworks. Lastly, we can observe the practical implications of the execution of transformation plans on various networks that enforce different message routing schemes and coordination protocols. All in all, we have set a novel foundation to evolve network topologies in a principled way. This work has the potential for high impact to the networking field and poses many interesting future research challenges for automated planning.

References

Baier, J. A., and McIlraith, S. A. 2008. Planning with preferences. *AI Magazine* 29(4):25–36.

Baldoni, R.; Beraldi, R.; Querzoni, L.; and Virgillito, A. 2004. Subscription-driven self-organization in content-based publish/subscribe. In *Autonomic Computing, 2004. Proceedings. International Conference on*, 332 – 333.

Biere, A. 2010. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. In *FMV Reports Series*. Linz, Austria: Institute for Formal Models and Verification, Johannes Kepler University.

Carzaniga, A., and Wolf, A. L. 2003. Forwarding in a content-based network. In *Proceedings of 2003 ACM Special Interest Group on Data Communication Conference (SIGCOMM 2003)*, 163–174.

Chockler, G.; Melamed, R.; Tock, Y.; and Vitenberg, R. 2007. Constructing scalable overlays for pub-sub with many topics. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing (PODC-07)*, 109–118. New York, NY, USA: ACM.

Cooper, B. F.; Ramakrishnan, R.; Srivastava, U.; Silberstein, A.; Bohannon, P.; Jacobsen, H.-A.; Puz, N.; Weaver, D.; and Yerneni, R. 2008. PNUTS: Yahoo!'s hosted data serving platform. *Proceedings of the 34th Conference on Very Large Data Bases (VLDB 2008)* 1:1277–1288.

De Santis, E.; Grandoni, F.; and Panconesi, A. 2007. Fast low degree connectivity of ad-hoc networks via percolation. In *Proceedings of the 15th Annual European Symposium on Algorithms (ESA 2007), Eilat, Israel, October 8-10*.

Jacobsen, H.-A.; Cheung, A. K. Y.; Li, G.; Maniymaran, B.; Muthusamy, V.; and Kazemzadeh, R. S. 2010. The PADRES publish/subscribe system. In *Principles and Applications of Distributed Event-Based Systems*. 164–205.

Jaeger, M. A.; Parzyjegla, H.; Mühl, G.; and Herrmann, K. 2007. Self-organizing broker topologies for publish/subscribe systems. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, 543–550. New York, NY, USA: ACM.

Lau, L. C.; Naor, J. S.; Salavatipour, M. R.; and Singh, M. 2007. Survivable network design with degree or order constraints. In *Proc. ACM STOC'07*.

Liben-Nowell, D.; Balakrishnan, H.; and Karger, D. 2002. Analysis of the evolution of peer-to-peer systems. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, PODC '02, 233–242. New York, NY, USA: ACM.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS 2011), Freiburg, Germany June 11-16*.

McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; and Turner, J. 2008. Openflow: enabling innovation in campus networks. *SIGCOMM Computer Communication Review* 38(2):69–74.

Migliavacca, M., and Cugola, G. 2007. Adapting publish-subscribe routing to traffic demands. In *Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems (DEBS 2007)*, 91–96. New York, NY, USA: ACM.

Onus, M., and Richa, A. 2011. Minimum maximum-degree publish/subscribe overlay network design. *Networking, IEEE/ACM Transactions on* 19(5):1331 –1343.

Reumann, J. 2009. Pub/Sub at Google. Lecture at EuroSys and Canada-Norway Partnership Program in Higher Education (CANOE) Summer School.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)* 39:127–177.

Yoon, Y.; Muthusamy, V.; and Jacobsen, H.-A. 2011. Foundations for highly available content-based publish/subscribe overlays. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems (ICDCS 2011)*, 800–811. Washington, DC, USA: IEEE Computer Society.