Planning the Transformation of Distributed Messaging Middlewares

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

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

Abstract. Refining a topology of a distributed messaging middleware (DMM) is an important management technique to provide a better service to the end-users. Nevertheless, determining the appropriate steps to transform a DMM 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 DMMs hosting mission-critical services. In this paper, we introduce the incremental topology transformation (ITT) problem. To address it, we draw parallels to automated plan automated planning techniques for solving the ITT problem. While state-of-the-art domain-independent planning techniques were effective for solving small problem instances, we found that they did not scale to the level necessary to solve large ITT problem instances. To address this shortcoming, we developed a suite of planners that use novel domain-specific heuristics to guide the search for a solution. We empirically evaluated our planners on a wide range of topologies. Our results illustrate that automated planning offers a viable solution to a diversity of ITT problems. We envision that our approach could eventually provide a compelling addition to the arsenal of techniques currently employed by the administrators of DMM to support its runtime refinement with minimal disruption to services.

1 Introduction

Messaging middleware is a critical communication substrate for distributed systems. One of the prevalent paradigms seen in the messaging middleware is a distributed publish/subscribe system formed in an overlay network [13, 5] which we refer to as distributed message middleware (DMM) in general. The success of DMM can be seen in large-scale enterprise systems such as Google Pub/Sub (GooPS) [21] and Amazon Simple Notification Service [1], to name a few. Recently, DMM also penetrated the domain of Internet infrastructure to enable content-oriented communication (PSIRP [2]).

Envisioning the success of DMM, researchers devised numerous management techniques to ensure high performance and efficiency. One of the classes of techniques deals with reconfiguration of the underlying topology of DMM [3, 14, 6, 18, 20, 16]. The algorithms suggested in these works 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. In fact, reconfiguring a DMM topology is an issue seen in many real-world systems such as GooPS [21]. For example, suppose we have a publish/subscribe overlay, 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, That is, 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.

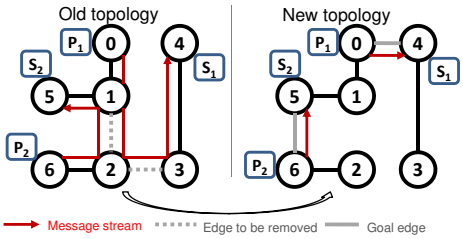


Fig. 1. Example of reconfiguring the topology of a pub/sub overlay

However, existing reconfiguration works have largely overlooked another practical issue of how to actually transform an existing deployed DMM from one topology to another in a way that minimizes the disruption of the services to the end-users.

This is an important problem, since disruptions can be particularly harmful and costly for DMM hosting mission-critical services. Without systematic and principled approaches for avoiding disturbances to the service, an arbitrary transformation of the running DMM to the new optimized topology can yield unpredictable and undesirable consequences.

In consideration of this, we develop a generic transformation framework that can determine the appropriate *steps* to evolve a DMM into a given new topology. The key challenge is produce the steps in a timely manner with the least disruption to the services. These steps involve atomic operators [23], each of which adjusts a single edge in the network in a way that maintains the integrity of the message streams in DMM.

Naturally, we can pose the problem of finding transformation steps as an automated planning problem [19] and explore the application of planning techniques and principles to this problem. Broadly speaking, the planning problem is to find a structure of actions, called a *plan*, that achieves a set of goal conditions when executed from a given starting state. For example, in the current setting the starting-state is a representation of the initial graph, and the goal conditions define features of the optimized topology that we are aiming to achieve, in particular connections between certain nodes. Domain-independent planning systems solve problems encoded in an input language, such as PDDL [9]. We found that the state-of-the-art domain-independent planning systems we tried (LAMA [22] and PROBE [17]) are unable to solve PDDL encodings of our problems, especially the large ones, effectively. This motivated us to develop new planning algorithms using novel *domain-specific* heuristics, which are proven to be effective through an empirical evaluation on a wide range of automatically generated DMM topologies. With our planning tools, we envision that the administrators in the field can more easily adopt various DMM topology reconfiguration techniques without worrying too much about the disruption during the realization of a refined topology.

This paper makes the following contributions: (1) we introduce the incremental topology transformation (ITT) problem for DMM and develop a metric to measure the quality of any solution in terms of the disruption to message streams (Section 2); (2) we cast the transformation problem as an automated planning problem (Section 3); (3) we present a number of properties about the ITT problem that can help a planning algorithm reduce the search space (Section 4); (4) we develop four planning algorithms that exploit the above properties(Section 5) and offer different tradeoffs on the overhead and plan quality; and (5) our algorithms are evaluated under various parameters, such as the size of the network topology and the degree of transformation (Section 6).

2 Problem Definition

2.1 Topology Transformation

Formally, in this paper 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 (v_i, v_j), where v_i and $v_j \in V$, also sometimes denoted $e_x(v_i, v_j)$. 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. 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 and the set of *goal edges* G that need to be realized in T_G. In this paper, we focus on a valid constraint prevalent in DMM that a topology is a connected undirected and acylic graph. Notable large-scale DMMs such as GooPS [21] and PSIRP [2] are in acyclic topologies. Also, IBM WebSphere MQ [8] supporting distributed content-based routing [5] recommends a tree topology in certain cases.

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

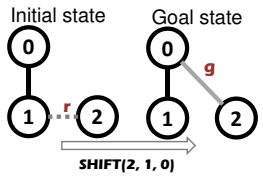


Fig. 2. Example of SHIFT operation

Suppose that we can use the following two simple transformation operations on a topology: (1) ADD an edge and (2) DELETE an edge. In this case, the simplest way to make a transition from T_S to T_G is to DELETE all edges in R from T_S and then ADD the edges in G.However, the disruption caused by such a transformation may be unacceptable. First, because such operations will violate the *integrity* requirements, *i.e.*, connectedness and acyclicity we imposed on our topologies. Second, because the disruption to DMM services the transformation causes may be unacceptable, if a topology is currently in use. We will formalize disruption in the subsequent section.

Using an *incremental* transformation approach with composite transformation operations, the structural properties of the topologies can be maintained, and disruption to services can be minimized. An operation that is suitable for an incremental transformation of a running topology was first introduced in [23], *i.e.*, SHIFT(v_i, v_j, v_k).

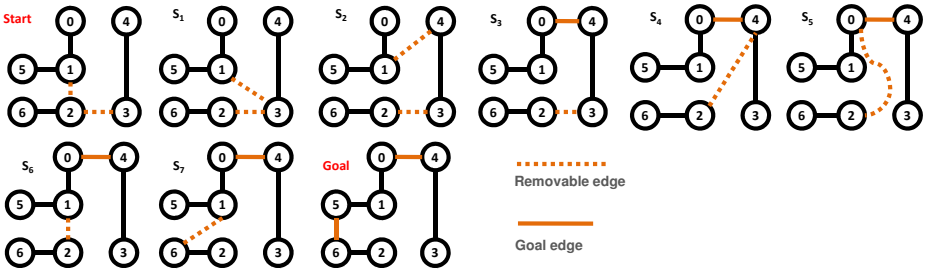


Fig. 5. A plan with 7 SHIFT steps

As shown in Figure 2, 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. In addition to the SHIFT operator, we introduce another atomic operation, MOVE. MOVE(v_i, v_j, v_k, v_l) directly replaces the goal edge $g(v_k, v_l)) \in G$, with the removable edge $r(v_i, v_j) \in R$, as the example shown in Figure 3.

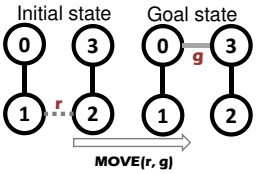


Fig. 3. Example of MOVE operation

It is proven that there exists a sequence of these operations to transform any topology of the type we consider into any other with the same type [23]. Both of the operations we introduce can be thought of as macros for sequences of DELETE and ADD operations that we assume to be executed atomically [23]. In the following section, we articulate our objective of improving the quality of the solution to the incremental topology transformation (ITT) problem by minimizing the disruption.

2.3 Solution Quality

If a topology is in use, then there are streams of messages flowing through the network to serve the end-users. In this paper, 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). Given a topology, an end-to-end path between the producer and the consumer ($\overrightarrow{p}^{\diamond}(v_p$ and $v_c)$) through which the messages traverse can be induced. Given the notion above, we illustrate a case where a message stream can be disrupted as shown in Figure 4. Suppose, a message stream exists between v_p (the producer) and v_c (the consumer), 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 ($v_p \rightarrow v_c$). Careful coordination among the vertices is required, otherwise transient loops or message order violation can occur as well. A reference implementation of a synchronous coordination among DMM nodes is presented in [23].

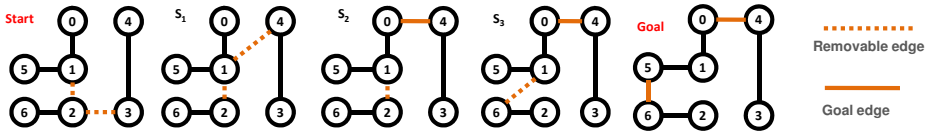


Fig. 6. A shorter plan of the problem in Figure 5

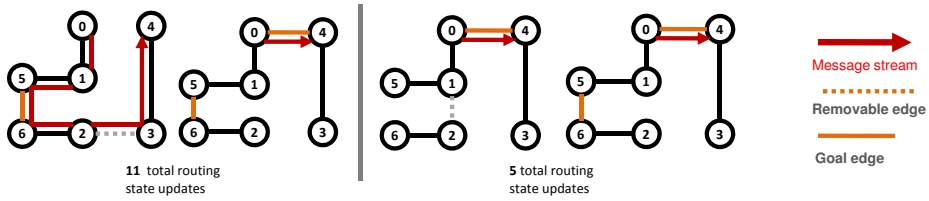


Fig. 7. A plan with MOVE operations. The order of executing the two MOVE operations can yield different number of routing state updates.

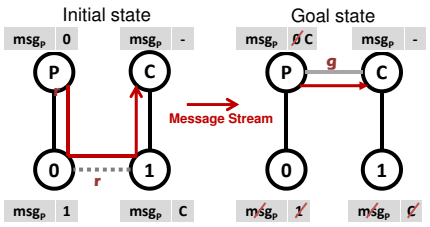


Fig. 4. Example of routing states (*message ID* and *next hop destination*) being updated during the topology transformation

Now, the key objective is to execute these operations in a particular sequence that causes minimal disturbance to the running system, more specifically the message streams. However, this can conflict with a goal that the transformation should be executed as quickly as possible as well, *i.e.*, the number of steps should be small.

Before we present the design and the implementation of a flexible and readily adaptive framework to derive a sequence of the transformation operations, we formalize the quality of a transformation solution with two factors. First, the number of steps in the transformation, and second, the degree of impact to the message streams flowing during the transformation. Consider the transformation problem in Figure 1 that was previously introduced. As a slight change, assume that there is just one message stream flowing between v_0 and v_4. There can be two sequences of SHIFT operations differing in length as shown in in Figures 5 and 6.Refer to another type of transformation that uses only MOVE operations as shown in Figure 7. With the first plan, the original message stream flowing between a PCPAIR(v_0, v_4) is prolonged by the execution of the first MOVE operation. This plan requires all vertices to update their routing states to reliably re-route the messages stream. For the second plan, the goal edge between v_0 and v_4 is established first. As a result the original message stream changes its routing path from $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$ to $v_0 \rightarrow v_4$. In total the second plan requires 6 routing state updates less than the first plan. This is a clear evidence that the ordering of the MOVE operations becomes relevant to the quality of a plan. Interestingly, the plan in Figure 6 also achieved the goal edge between v_0 and v_4 first, with a series of smaller-scale SHIFT operations. This yields a conjecture

that the number of steps can be reduced if the SHIFT operations follow certain order of MOVE operations as trajectories. This hints at how a plan quality can be improved.

2.4 Discussion

We assume that the environment in which the topology operates is static long enough to allow the finding and implementing of a transformation. Also, we focus a fixed provisioning of resources, such as the number of connections and nodes, in the DMM topology being examined. Thus, we do not consider the operations of attaching/detaching vertices which is relatively trivial [23].

Here we focus on acyclic topologies. But advanced message routing techniques on cyclic topologies have emerged as well [15]. In cyclic topologies, complexity arises in quantifying the disruption to the services running on it. In order to understand the disruption in cyclic topologies, we need to derive principles on how various custom algorithms exploit the redundant paths to route message streams during the transformation. In contrast, disruption model we constructed earlier can be generalized for any acyclic topology, since there is only one path between any two nodes. It is a future work to consider a transformation of a topology with cycles.

3 Mapping to a Planning Problem

An incremental topology transformation (ITT) problem can naturally be encoded as an automated planning problem. Such an encoding is intuitively appealing as the 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 ITT 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 Formal Definition of a Planning Problem

Formally, a propositional planning problem Π is a tuple $\langle \Lambda, \Sigma, \mathcal{O}, s_0, \mathcal{G} \rangle$. Here, Λ is a set of predicates with arguments and Σ is a set of objects. The grounding of Λ with Σ creates a set of propositional variables $propositions$. For example, let $\Sigma \equiv \{v_P, v_1, v_2, v_C\}$ be a set of objects representing the vertices in the graphs in Figure 4, where all objects have the type $node$. Next, let Λ contain a single predicate $conn(?x - node, ?y - node)$. We then have the set of propositions

$$propositions \equiv \{conn(v_P, v_1), conn(v_1, v_P), conn(v_P, v_2), ...\}$$

Π then has the *state-space* $\mathcal{S}$, where each $s \in \mathcal{S}$ is a valuation of the variables in $propositions$. Here, we may represent a state $s \in \mathcal{S}$ as the set of variables it assigns to *true*. $s_0 \in \mathcal{S}$ is the initial state of the problem (the start state) and $\mathcal{G}$ is a set of propositions representing a set of goal states. A state $s \in \mathcal{S}$ is a goal state *iff* $s \subseteq \mathcal{G}$. For instance, the initial state shown in Figure 4 is represented by the following set:

$$\{conn(v_p, v_1), conn(v_1, v_p), conn(v_1, v_2), conn(v_2, v_1), conn(v_2, v_c), conn(v_c, v_2)\}$$

$\mathcal{O}$ is a set of first-order planning operators. For each $o \in \mathcal{O}$, we have that $o = \langle O(\overrightarrow{x}), pre(o), add(o), del(o)\rangle$, where O is the operator name; $\overrightarrow{x}$ is a set of typed variables, $pre(o)$, the operator precondition, is a propositional formula with terms made from Λ and variables from $\overrightarrow{x}$. $add(o)$ and $del(o)$ are the add effects and delete effects (resp.) of o, with elements made from Λ and $\overrightarrow{x}$.

An action a is a grounding of an operator $o \in \mathcal{O}$ with the objects in Σ, respecting types. An action a, that instantiates operator o, has a precondition formula and an add and delete lists made up of $propositions$. Let $\mathcal{A}$ be the set of action produced by grounding the operators in $\mathcal{O}$ with Σ. An action a is *applicable* in a state s *iff* $s \models pre(a)$. The resulting state $s' \equiv s \setminus del(a) \cup add(a)$.

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 $\langle s_0, ..., s_h, s_{h+1}$, where state s_0 is the initial state, action $s_i \models pre(a_i)$ and produces s_{i+1}, and finally, state $s_h + 1 \subseteq \mathcal{G}$.

The quality of a plan may be measured in a number of ways. One common approach is to define a metric function that assigns a cost to the execution of each action and then define the cost of a plan as the sum of the costs of the actions that it contains. Alternatively, preferences may be expressed over features of the state trajectories implied by plans [12]. These preferences are often expressed as weighted temporal logic formulae. A high quality plan is then one which maximizes the weight of satisfied preference formulae. Detailed presentation of these aspects of planning are outside the scope of this exposition.

3.2 Encoding The Transformation Problem

As suggested by the examples in the previous section, the ITT problem $\mathcal{T} = \langle T_S, T_G, O\rangle$, where O contains SHIFT and MOVE operators can be posed naturally as a planning problem. Let $\Pi_{\mathcal{T}} = \langle \Lambda_{\mathcal{T}}, \Sigma_{\mathcal{T}}, \mathcal{O}_{\mathcal{T}}, s_{0\mathcal{T}}, \mathcal{G}_{\mathcal{T}}\rangle$ be an encoding of the topology transformation problem $\mathcal{T}$. First, we have the following set of predicates:

$$\Lambda_{\mathcal{T}} \equiv \{conn(v_i, v_j), eq(v_i, v_j), rem(v_i, v_j)\}$$

$conn$ represents that vertices are connected, eq represents that they are equal, and rem represents that an edge between these vertices can be removed. Next, if $V_{\mathcal{T}}$ is the set of vertices in the graphs in $\mathcal{T}$, then we have the following set of objects $\Sigma_{\mathcal{T}} \equiv V_{\mathcal{T}}$. Operator $o_1 and o_2 \in \mathcal{O}$ are SHIFT and MOVE respectively, defined Section 2.2, but where

$pre(o_1) \equiv conn(v_i, v_j) \wedge conn(v_j, v_k) \wedge rem(v_i, v_j) \wedge \neg eq(v_i, v_j) \wedge \neg eq(v_j, v_k)$
$add(o_1) \equiv \{conn(v_i, v_k)\} del(o_1) \equiv \{conn(v_i, v_j)\}$

$pre(o_2) \equiv conn(v_i, v_l) \wedge conn(v_j, v_k) \wedge conn(v_i, v_j) \wedge rem(v_i, v_j) \wedge \neg eq(v_i, v_j)$
$\wedge \neg eq(v_j, v_k) \neg eq(v_k, v_l)$
$add(o_2) \equiv \{conn(v_k, v_l)\} del(o_2) \equiv \{conn(v_i, v_j)\}$

For example, Figure 4 has a grounding of SHIFT *i.e.*, $a = \text{SHIFT}(v_p, v_1, v_2)$, where:

$pre(a) \equiv conn(v_p, v_1) \wedge conn(v_1, v_2) \wedge rem(v_p, v_1) \neg eq(v_p, v_1) \wedge \neg eq(v_1, v_2)$
$add(a) \equiv \{conn(v_p, v_2)\}$
$del(a) \equiv \{conn(v_p, v_1)\}$

For this example, the following sequence of actions is a plan that transforms the initial state to the goal state (Figure 4): $\langle \text{SHIFT}(v_p, v_1, v_2), \text{SHIFT}(v_p, v_2, v_c) \rangle$

The start state $s_{0\mathcal{T}}$ contains one $conn$ proposition for every edge in T_S one rem proposition for every edge in T_S that is not in T_G and one predicate $eq(v_i, v_i)$ for every vertex $v_i \in V_{\mathcal{T}}$. In this case the goal $\mathcal{G}_{\mathcal{T}}$ defines a single goal state,$\mathcal{G}_{\mathcal{T}}$ contains one $conn$ proposition for every edge in T_G that is not in T_S.

3.3 Performance of Existing Planners

We used the encoding described in the previous section to specify a set of randomly generated ITT problems into PDDL. Details about the problem generator we used can be found in Section 6. We found that the state-of-the-art domain-independent planning systems are unable to solve PDDL encodings of our problems effectively. For example, neither LAMA [22] nor PROBE [17] were able to solve 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.

Both LAMA and PROBE are forward state-space search algorithms. That is, during search they have a frontier of states, here each representing a topology, and iteratively select a state to expand based on a heuristic estimate of the goal distance from each state. While these two algorithms perform very well on some classes of planning problems, our ITT problem presents them with several significant difficulties. First, a topology transformation problem has $O(n^3)$ actions, where n is the number of vertices in the topology. For realistically sized problem, this overwhelms LAMA and PROBE, both of which produce all ground actions prior to beginning search. Another issue that particularly LAMA faces here is that there are many symmetries in the search space. For example, the states reached by $\text{SHIFT}(v_i, v_j, v_k)$ and $\text{SHIFT}(v_j, v_i, v_k)$ often have the same value. This is less of a problem for PROBE, which breaks these symmetries by computing probes. In principle, other existing planning algorithms that do not completely ground the problem up front should have more success with this problem. 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 Domain-Specific Properties

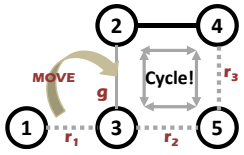


Fig. 8. Invalid MOVE

To overcome the difficulty that the state-of-the-art domain-independent planners have in solving topology transformation problems, we require customized planners that avoid generating all ground actions, allow for the use of MOVE operators, and employ domain-specific heuristics to improve performance and use appropriate objective functions.

To derive the useful heuristics, we need to take a more in-depth look at our problem domain. We first examine the conditions under which a MOVE operation is valid. Suppose a part

of some topology is given in Figure 8, where a goal edge $g(v_2, v_3)$ needs to be realized. Here we have the removable edges $r_1(v_1, v_3)$, $r_2(v_3, v_5)$ and $r_3(v_4, v_5)$ along the path between v_1 and v_2 (denoted $\overrightarrow{p}(v_1, v_2)$). In this situation we could not execute the action MOVE(v_1, v_3, v_1, v_2) because, as r_1 is not in $\overrightarrow{p}(v_2, v_3)$, the execution of the action MOVE(v_1, v_3, v_1, v_2) would cause the topology to be cyclic and disconnected, violating the structural requirements imposed on topologies in the problem definition. A difficulty also presents itself with the execution of SHIFT actions. In this case, there are SHIFT actions that, while not violating the structural requirements imposed on our topologies, cannot be a useful part of a plan. We use the topology and the goal edge to realize as in Figure 9 (same as the ones in Figure 8). Similar to the case with the MOVE operator, suppose we select r_1 to involve in the very first SHIFT action, *i.e.*, SHIFT(v_1, v_3, v_5). As mentioned previously, r_1 is not on $\overrightarrow{p}(v_2, v_3)$. Suppose a planner considers the intermediate edges that are not on $\overrightarrow{p}(v_2, v_3)$ as well for the actions following SHIFT(v_1, v_3, v_5). That is, the planner generates the sequence:

$$\langle \text{SHIFT}(v_1, v_3, v_5), \text{SHIFT}(v_1, v_5, v_4), \text{SHIFT}(v_1, v_4, v_2) \rangle$$

After applying these actions, the topology reaches a state where the only applicable action regarding the intermediate removable edge $r(v_1, v_2)$ is SHIFT(v_1, v_2, v_4). The add effect of SHIFT(v_1, v_2, v_4) is the same as that of SHIFT(v_1, v_4, v_2), meaning that SHIFT(v_1, v_2, v_4) revisits the previous state and makes no progress towards the goal state.

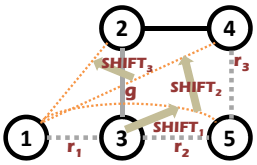


Fig. 9. Unnecessary SHIFT actions

These cases can be avoided if a planner abides by a special rule we set as follows. We denote the path between the two nodes (v_x and v_y) that consist of a goal edge g as $\overrightarrow{p}^{\diamond}(g)$ ($\equiv \overrightarrow{p}(v_x, v_y)$). Given this notation and the observation we made above, we set a rule for selecting a removable edge for the transformation operations as specified in Theorem 1 that is based on Corollary 1.

Theorem 1. *A* MOVE *or a* SHIFT *operation a to achieve a goal edge, $g(v_x, v_y)$, is valid,* iff *a removable edge that is on $\overrightarrow{p}^{\diamond}(g)$ is involved in a.*

Corollary 1. *There must be at least one removable edge on the path between v_x and v_y of a goal edge $g(v_x, v_y)$ in a problem $\mathcal{T}$ of transforming a connected acyclic topology T_S to another connected acyclic topology T_G.*

Proof. Suppose an incremental topology transformation problem $\mathcal{T}$ has only one removable edge $r(v_i, v_j)$ to remove and one goal edge $g(v_x, v_y)$ to establish. Assume that r is not on $\overrightarrow{p}^{\diamond}(g)$. The only way to reach the goal is to remove the single removable edge, and add an edge between v_x and v_y. However, this contradicts with the specification of $\mathcal{T}$ that its goal graph T_G is acyclic and connected.

We consider another rule as follows.

Property 1. A stationary edge should not be involved in any transformation operation.

A removable edge is an edge that should be removed from the initial graph. In contrast, stationary edges do not have to be disconnected during the transformation, since the stationary edges are, by default, already parts of the goal graph. Thus, intuitively, involving the stationary edges in any of the transformation operations is unnecessary. It is currently not known whether there is a special circumstance where involving the stationary edges in the transformation operations can be beneficial. However, according to our observation, the plan search space can grow significantly if the stationary edges are also considered in the transformation. So, in order to meet one of the objectives of finding a plan as quickly as possible, we adhere to Property 1.

But, these rules are practically infeasible to encode according to the conventional formalism we presented in Section 3. Thus, these rules are implemented as heuristics for the custom planning algorithms we will discuss in the next section.

5 New Planning Techniques

5.1 Greedy

Our first algorithm (GREEDY) focuses on quickly finding a plan consisting of SHIFT operations.

Algorithm 1: Greedy

Input: T_S, T_G, a set of goal edges G
Output: A sequence of SHIFT operations

1 **for** $g(v_x, v_y)$, $\in G$ **do**
2 $\quad R(g)$ = A set of removable edges of g;
3 $\quad$ Search for $\overrightarrow{p}^{\diamond}(g)$;
4 $\quad$ Add removable edges on $\overrightarrow{p}^{\diamond}(g)$ into $R(g)$;
5 $\mathcal{CA}$= a set of candidate actions;
6 **for** $g(v_x, v_y) \in G$ **do**
7 $\quad$ **for** $r \in R(g)$ **do**
8 $\quad\quad$ Add candidate actions into $\mathcal{CA}$;
9 Apply an action in $\mathcal{CA}$;
10 Update current topology and repeat 1 - 9, until T_G is achieved;

For each goal edge $g(v_x, v_y)$ in the set of goal edges G, the algorithm first searches for the path $\overrightarrow{p}(v_x, v_y)$ (denoted also as $\overrightarrow{p}^{\diamond}(g)$). While searching for $\overrightarrow{p}^{\diamond}$, ordering information between the vertices along $\overrightarrow{p}^{\diamond}$ is updated. For instance, GREEDY identifies $\overrightarrow{p}^{\diamond}(g(v_4, v_5)) = v_4 \rightarrow v_2 \rightarrow v_0 \rightarrow v_1 \rightarrow v_3 \rightarrow v_5$. With this information GREEDY knows that v_2 precedes v_0 and succeeds v_4, for example. While searching for $\overrightarrow{p}^{\diamond}$ for every goal edge, GREEDY identifies whether an edge between any two adjacent vertices along $\overrightarrow{p}^{\diamond}$ is a removable edge (*e.g.*, $e(v_0, v_1)$ on $\overrightarrow{p}^{\diamond}(g(v_4, v_5))$ in Figure 10). Such removable edges are tracked for every goal edge. Once this process (line 1-4) gets completed, GREEDY generates a set of all candidate actions that are applicable in the current topology. For example, GREEDY associates the removable edge $r(v_0, v_1)$ with the goal

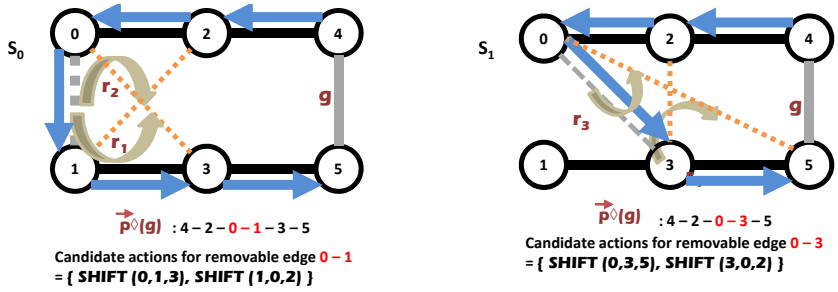


Fig. 10. An example of identifying candidate actions using the removable edges along the path for a goal and planning the subsequent actions.

edge $g(v_4,v_5)$ in Figure 10. Here, the aforementioned ordering information GREEDY associated with $\overrightarrow{p}^{\diamond}(g)$ provides a trajectory for the gradual SHIFT operations that involve $r(v_0, v_1)$. For example, $r(v_0, v_1)$ in Figure 10 can be shifted either towards v_2 (*preceding* vertex v_0) or towards v_3 (*succeeding* vertex v_1). After all possible candidate actions are collected in this way for every goal edge, GREEDY picks the most appropriate action and then makes a transition to the updated subsequent state. In practice, the appropriateness of a candidate action can be judged by a user-defined condition, *e.g.*, the availability of the vertices. We assume that these conditions are profiled in advance. GREEDY repeats the process (Line 1-9) until the goal topology is realized.

Note that SHIFT-based planning is susceptible to cause a *conflict*. A conflict occurs if $\overrightarrow{p}^{\diamond}$ 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 $\overrightarrow{p}^{\diamond}$s overlap with each other. If the conflict is not resolved, the transformation problem can enter an invalid state. The conflict is resolved by assigning a new $\overrightarrow{p}^{\diamond}$ and a new set of removable edges for any goal edges affected by the conflict.

The overall disruption to the message streams in the network is determined by the total number of actions in the plan, where the number of routing state updates is constant for each SHIFT operation, *i.e.*, 3. Another algorithm in the following section attempts to find a plan with a minimal number of *actions*.

5.2 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 domain-specific heuristic. Before we outline the algorithm, we must define some required data structures. Here, a state s is a set of true fluents (*i.e.*, propositions that appear as the delete effect of some action). Let CL, called the closed list, be a map from states to an ordered 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. It is ordered so that the list with the lowest $go + hval$ is at the front, breaking ties randomly.

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 π from $sl.s$;
15 $bestplan = \pi, bestscore =$ cost of π;
16 restart at line 1;
17 **return** $bestplan$;

In each iteration this algorithm expands the best state from the OL that has either 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. Those actions are used to produce successor states. If a successor state is 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 CL. This restarting, with the fact that states are ranked on the OL 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 $\overrightarrow{p}^{\diamond}$ of a goal edge.

5.3 Macro Planner

This algorithm (MACRO) finds a plan only with the MOVE operations. In contrast to the two previous algorithms, the MACRO algorithm generates a plan with a fixed number of actions which is equal to the number of goal edges to achieve. But, each MOVE operation can involve relatively more vertices for the routing state updates than what a

Algorithm 3: Macro

Input: Planning encoding $\Pi_{\mathcal{T}}$ of a problem $\mathcal{T}$
Output: A sequence of MOVE operations
1 Same as Algorithm 1:1-4;
2 Find a set (S) of unique pairs of a removable edge and a goal edge;
3 Generate X uniquely random ordering of S;
4 Find the best ordering (P) that imposes the least disruption to the PCPAIRS ;
5 Construct a MOVE operation for every pair in P;

SHIFT involves. MACRO starts with associating a set of removable edges and a $\overrightarrow{p}^{\diamond}$ for every goal edge, similar to the way GREEDY starts (Algorithm 1). At this point, multiple removable edges may get associated with a goal edge. Vice versa, a removable edge can be associated with more than one goal edge as well. In order to have a fixed number of actions in a plan, a unique pairing between the goal edges and the removable edges should be found (Line 2). Given N goal edges and A average number of removable edges associated with each goal edge, the search space for finding the unique pairing is $O(2^{AN})$. Depending on the order of choosing a removable edge for the goal edges, it may become impossible to assign some removable edges to a goal edge, and an applicable MOVE operation may not be found for the goal edge. Finding a valid unique pairing is a non-trivial problem considering the search space, which may have many invalid pairings. However, thanks to the advancement of the propositional However, thanks to the advancement of the propositional satisfiability (SAT) solvers, this can be solved quite efficiently, if we encode it into a SAT problem.

An instance of the SAT problem consists of a set of propositional variables β and a conjunctive normal form (CNF) formula ϕ over the variables in β. A CNF formula is a conjunction of clauses, where each clause is a disjunction of literals, and each literal is either a variable in β or its negation. The solution to a SAT problem (β, ϕ) is a valuation $\mathcal{V}$ over the variables in β, such that ϕ evaluates to true.

Suppose we have a set of goal edges G and removable edges R, such that for each $g \in G$, and the set $R(g) \subseteq R$ is the set of removable edges on the path between the vertices in g ($\overrightarrow{p}^{\diamond}(g)$). 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.

Such a SAT problem can be solved with any number of generic SAT solvers. We used the solver PRECOSAT [4] as it performed particularly well on the SAT instances we generated. Once PRECOSAT has solved the SAT problem of this form, the assignments of removable edges to goal edges can be extracted from the valuation returned as a solution. Now, MOVE actions are created from the set of unique assignments obtained above. MACRO randomly shuffles the set of the instantiated MOVE operations X times (Line 3) and finds the best order (actually a *solution*) that yields the least number of routing state updates. Note that there are $N!$ ways in total to order N MOVE actions. All

$N!$ orderings make a valid plan as long as the algorithm follows the domain-specific rules we discussed in Section 4. It is impractical to search for the best-quality plan by considering all possible orders. Therefore, we bound the generation of orderings to X which is a configurable parameter for this algorithm (Line 4).

5.4 Combined Planner

The number of actions in the MACRO planner is fixed. But, its potential drawback is the relatively higher number of routing state updates incurred by each MOVE action in the plan. If a large number of routing state updates incurred by each action is not desirable, then the action can be realized with multiple smaller actions (SHIFT). This gives a flexibility in controlling the disruption at each step in a plan. Algorithm 4 (COMBINED) specifies integration of SHIFT operations into the MOVE operations. The input to COMBINED is a plan obtained by the MACRO planner (Algorithm 3). Note that which goal edge to establish first is already determined in the the MACRO plan. Hence, 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 MACRO, the total number of actions in a plan can be expected to be reduced as well. Nevertheless, a conflict, which we explained in Section 5.1, is still possible. The resolution of the conflict is described in line:4-7. That is, the new $\overrightarrow{p}^{\diamond}$ of a goal edge is newly identified and a removable edge on that $\overrightarrow{p}^{\diamond}$ is assigned to the goal edge for the consideration of SHIFT operations in the subsequent states.

Algorithm 4: Combined Planner

Input: A sequence MOVE operations generated by MACRO, π; a set of goal edges, G
Output: A sequence of SHIFT operations

```
1 for MOVE(r, g) ∈ π do
2     Apply a SHIFT operation (a) on r;
3     for ∀g' ∈ G − g do
4         if path p⃗^◇(g') has to change as the effect of a then
5             Search new p⃗^◇(g');
6             if r(g') not on p⃗^◇(g') then
7                 Find a new r(g');
8     Repeat 2-7 until g is realized;
```

5.5 Discussion

We plan to extend our work to allow concurrent execution of multiple transformation actions. We also plan to allow the specification of temporal constraints on which traffic is acceptable to be disrupt. For instance, assume a single message stream flows on $\overrightarrow{p}^{\diamond}(v_0, v_1)$ in the previous case. 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, preferences can be precisely quantified 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.

6 Evaluation

We evaluated the planning algorithms developed in this paper with simulated DMM transformation problems. The simulation was conducted on Dell PowerEdge 2900 with two quad-core 3 GHz processors and 16GB of RAM. We have devised a tool that can generate transformation problems according to the parameters [4] in Table 1 that determine the characteristics for start and goal graphs. Given this tool we generated a set of

Parameter	Description
Number of nodes	The number of nodes in a start and a goal graphs (20 - 400)
Maximum degree	The maximum degree of any node
Maximum distance	The maximum length of any path in the generated graphs
Degree bias	The distribution of node degrees to aim for (-100 to 100)
Degree of change	Difference between a start and a goal graph (0 - 60%)
Start graph clustering	Degree of clustering in changing edges in the start state (0-1)
Goal graph clustering	Degree of clustering in changed edges in the goal state (-100 to 100)

Table 1. Configurable network and transformation problem characteristics

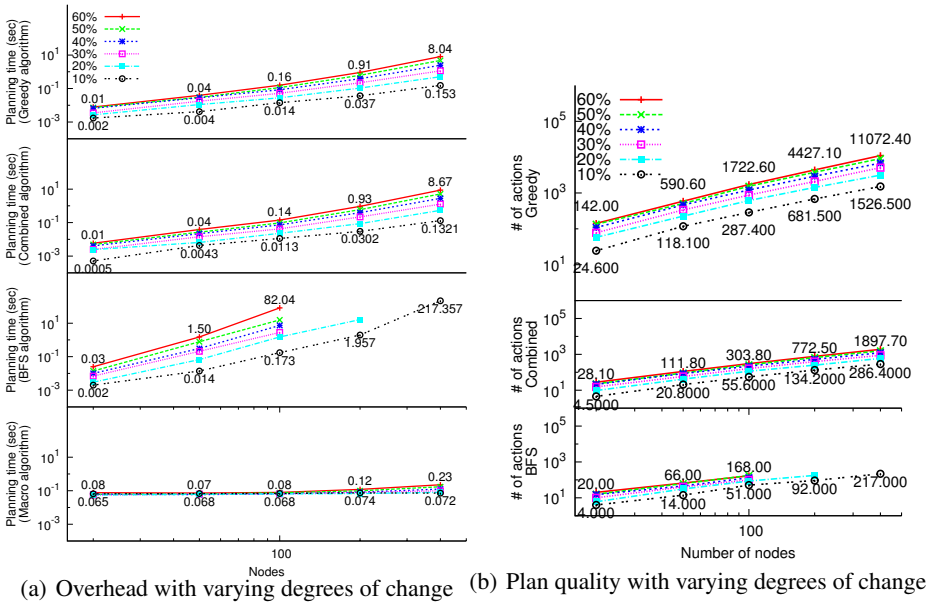
transformation problems to evaluate mainly: (1) the overhead of the planning algorithms and (2) the quality of the plan our algorithms find. Specifically, we vary the number of nodes in the start and goal graphs and the degree of change The degree of change varies, all other parameters are controlled (start graph clustering = 0.1, goal graph clustering = 100, maximum degree = 5 and degree bias = 10). Note that PCPairs that represent the message streams in DMM. It is generated at least one per end-to-end path ($\overrightarrow{p}^{\diamond}$) for a goal edge, so that we can assess the disruption a transformation execution plan can cause.

Interestingly, we have not found any meaningful correlation between the *overhead* and the problem characteristics other than size of the network and the degree of change, within the the fixed maximum distance of 15. There is an exception, however, for the assessment of the plan *quality*, as we vary the search space limit and the average length of message streams (*flength*).

6.1 Overhead

The overhead of a planning algorithm is measured as the elapsed time to find a plan. Figure 11(a) shows the time it took for our algorithms obtain a plan. GREEDY and COM-

[4] For degree bias and start/goal graph clustering, the parameter represents the exponent k, where $p(x(v)^k)$ is the probability of selecting vertex v and x is a function representing node degree, distance from a specified vertex, etc., as appropriate for the specified graph property.



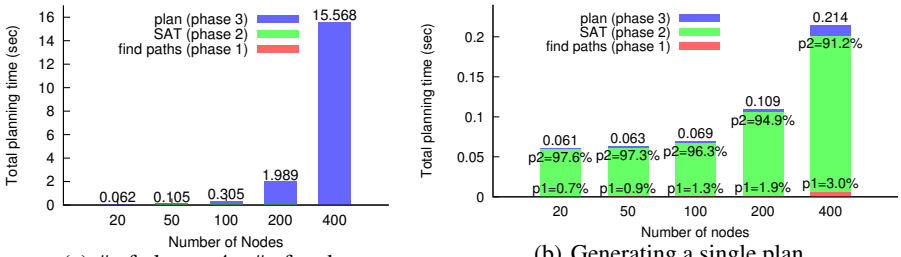
(a) Overhead with varying degrees of change (b) Plan quality with varying degrees of change

Fig. 11. Overhead and plan quality with varying degree of changes

BINED showed almost identical overhead. They found a plan in 1 ms for transforming a 20-node network with 10% change. For a larger topology with 400 nodes, it took only 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 topology with 60% change, GREEDY and COMBINED took about 70 ms to find a plan. For a 400-node topology with the same degree of change (60%), it took about 8 seconds to find a plan.

The BFS algorithm, which is tuned to find a higher-quality plan, performs considerably worse than both GREEDY and COMBINED. For a 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 is due to a larger search space to find a better plan. Figure 11(a) also shows that the MACRO planner's runtime, is relatively less sensitive to the number of nodes or the degree of change in the graphs. For the large number of nodes (400), it outperforms the GREEDY and COMBINEDby finding a plan in about 0.2 seconds.

The performance of our planners are considerably better than the generic state-of-the-art planners presented in Section 3. Recall that those planners could not even find a plan for transforming a graph with more than 50 nodes with over 50% change, whereas GREEDY and COMBINED can finds a plan well under a second. This is a clear empirical evidence that the incorporation of the domain-specific heuristics we derived in Section 4 are very effective. Figure 12 shows the runtime of each key phase MACRO has to go through: *phase 1* where the planner searches the $\overrightarrow{p}^{\diamond}$ for a goal edge and asso-



(a) # of plans = 4 x # of nodes (b) Generating a single plan

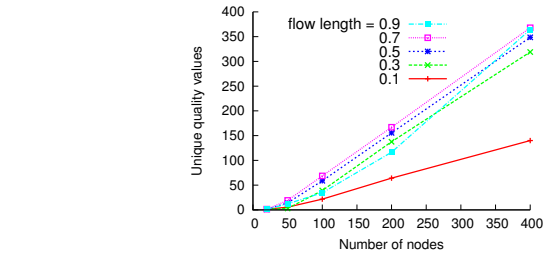
Fig. 12. Profiled overhead of MACRO planner with varying number of search states and degree of change for an ITT problem with 60% change)

ciates removable edges with the goal edge; *phase 2* where it uses the SAT solver to find unique pairings between the goal edges and the removable edges; and *phase 3* where a solution is generated out of the unique pairings obtained from *phase 2*. According to Figure 12(b), the overall time to find a plan, *i.e.*, from *phase 1* to *phase 3*, was at most 0.24 seconds for finding a plan for transforming a 400-node graphs with 60% change with MACRO. We see in Figure 12(a) that for a 400-node graphs with 60% change, the overall planning can take up to 16 seconds when considering 1,600 plans. But, for the smaller graphs with fewer than 400 nodes, planning takes less than 2 seconds. We see that if the MACRO planner is only focused on finding a solution, then *phase 2* dominates the planning time. On the other hand, if more plans need to be considered to find a better plan with fewer routing state updates, then, *phase 3* dominates (Figure 12(a)).

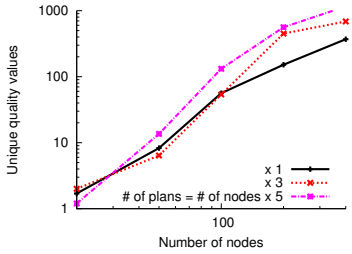
6.2 Plan Quality

Here we discuss the quality of a plan that are generated by our planners. Figure 11(b) shows the number of actions in a plan that is found by GREEDY, BFS and COMBINED planners. The settings of the problem is the same as the experiments in the previous section that assessed overhead. We see that GREEDY generated about 4 times more actions than what COMBINED generated. On the other hand, the quality of the plans generated by COMBINED and BFS are relatively closer to each other, according to Figure 11(b). 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, COMBINED yielded about 63% and 36% more actions than what BFS yielded, respectively. Althought it is a noticeable difference, COMBINED yielded the SHIFT actions, within a very small fraction of BFS's overhead. Note that, for the 100-node topology with 30% change, BFS found the first plan in a minute, whereas COMBINED found one with a close quality in 0.13 seconds on average.

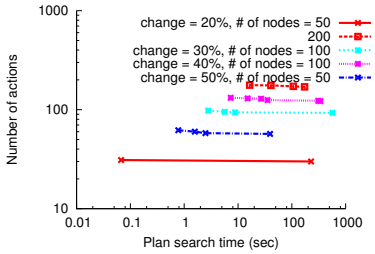
In case of the MACRO plan, its main objective is to find a plan that incurs less routing state updates for guiding the message streams to the newly updated paths. In addition to the previous experiment settings, we vary the number of plans MACRO generates and the average flength of the message streams. We set the problem to have at least one message stream that traverses the part of the $\overrightarrow{p}^{\diamond}$of a goal edge. Given this setting, we measure the total number of routing state updates ($\mathbb{U}$). We also counted the number of



(a) Varying flength (MACRO)



(b) Varying search states (MACRO)



(c) Varying search states (BFS)

Fig. 13. Effect of flength and the number of search states on the plan quality for MACRO and BFS

unique quality values measured as the number of routing state updates ($\mathbb{Q}$). With $\mathbb{U}$, $\mathbb{Q}$ can represent the variance in the routing state updates. We varied the average flength ranging from 0.1 to 1.0. This range indicates how much of the path of the message stream overlaps with $\overrightarrow{p}^{\diamond}$ of a goal edge. For example, flength = 0.1 means message path overlaps 10% of $\overrightarrow{p}^{\diamond}$. We anticipated that ($\mathbb{Q}$) would be sensitive to the flength. A longer flength is more likely to let the message streams traverse through multiple removable edges, thus there can be more diverse plans around those edges. Our conjecture did hold as Figure 13(a) shows that and ($\mathbb{Q}$) grew with increasing flength.

Whether the quality is close to optimal is another important thing to observe. However, it is infeasible to assess this by searching for a formidably large number of plans. Instead, we show how the plan quality increases, as we increase plan search states.

For MACRO planner, we vary the number of plans to search for (in *Phase 3* as described Section 5.3). MACRO yielded higher ($\mathbb{U}$) for larger topology, but ($\mathbb{Q}$) did not change much by increasing the plan search states. Figure 13(b) shows that the least number of routing state update found by assessing 2000 plans for a 400-node topology with 10% change did not differ according to from the number obtained by assessing 400 plans. BFS exhibited a similar behavior. The quality (measure in terms of the number of actions) of the very first plan obtained by BFS did not change much as we increased the limit on the number of search states. For example, before the time-out of 10 minutes, BFS was able to find 4 different solutions for the 100-node topology with 30% change. But the last solution found by BFS only had 5 less number of steps than the first solution. From this outcome, we can infer that there are many plans with identical quality and the quality is bounded within a narrow range, in our problem settings. The results of BFS proves the effectiveness of its heuristic function that gauges the distance between the goal edge and the removable edges.

7 Related Work

In this section, we introduce a few relevant DMM reconfiguration works. In [6], algorithms are developed to maintain topic connectivity in a publish/subscribe overlay while minimizing node degree. But, the configuration is done offline with little concern on the runtime implication. In [3, 14], nodes in a DMM overlay coordinate themselves in runtime to determine a better topology. However, the overhead of the operations to realize the new topology is not studied in depth. Cugola *et al* devised an algorithm to restrict subscription table update only on the brokers along the *reconfiguration path* to minimize the reconfiguration overhead [7]. But, the transformation disruption can still be high, if the reconfiguration path is long. Instead, primitive operations suitable for small-scale incremental transformation were developed in [23]. A notable non-DMM work [11] quantifies disruption in terms of the number of transient loops for reconfiguration of OSPF network topologies. [23], [7] and [11] are based on the theory of graph re-write system [10]. Note that our work is beyond developing topology transformation operations, as our work systematically plans runtime execution of *multiple* transformation operations for DMM topologies with minimal disruption to the service, which is very novel to the best of our knowledge.

8 Conclusion

We have discussed the challenges of incrementally transforming a DMM topology to a reconfigured one, without disrupting the DMM services, by using the automated planning framework. We modeled the disruption in terms of the number of steps in the transformation plan or the number of routing state update required, which is quantified with the objective functions. We have highlighted the shortcomings of the existing generic planning algorithms. To address these shortcomings, we developed domain-specific heuristics and incorporated these into novel custom planning systems. These planning systems significantly outperformed existing planners and quickly found high quality solutions to realistically sized problems. We envision that with our planning systems, topology optimization work studied in the academic literature can become practically useful, as they can be reliably and efficiently deployed in runtime with no concern on the disruption to DMM services. Many intriguing challenges can be addressed in the future. Our objective functions can be augmented to reflect the real world costs of transforming a DMM topology more accurately. Also, future systems should have the flexibility to respect various restrictions imposed by administrators on the DMM transformation. such as limits on the node degrees, path lengths and other network characteristics during the intermediate steps of a transformation plan. Lastly, we plan to investigate the transformation of cyclic DMM topologies.

References

1. Amazon simple notification service. http://aws.amazon.com/sns/.
2. Publish-Subscribe internet routing paradigm. http://www.psirp.org.

3. R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. Subscription-driven self-organization in content-based publish/subscribe. In *ICAC*, 2004.
4. A. Biere. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. In *FMV Reports Series*. Institute for Formal Models and Verification, Johannes Kepler University, 2010.
5. A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *SIGCOMM*, 2003.
6. C. Chen, R. Vitenberg, and H.-A. Jacobsen. Scaling construction of low fan-out overlays for topic-based publish/subscribe systems. In *ICDCS*, pages 225–236, 2011.
7. G. Cugola, D. Frey, A. L. Murphy, and G. P. Picco. Minimizing the reconfiguration overhead in content-based publish-subscribe. In *SAC '04*, pages 1134–1140, 2004.
8. S. Davies, G. Baddeley, M. Cernicky, B. Cuttell, R. P. Jha, B. Sapolyo, A. Shivaprasad, V. Suarez, and L. Thyagaraj. Websphere mq v7.0 features and enhancements, 2009. http://www.redbooks.ibm.com/abstracts/sg247583.html.
9. S. Edelkamp and P. Kissmann. PDDL 2.1: The language for the classical part of IPC-4. In *IPC-4*, 2004.
10. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
11. P. Francois, M. Shand, and O. Bonaventure. Disruption free topology reconfiguration in ospf networks. In *INFOCOM'07*, 2007.
12. A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artif. Intell.*, 173(5-6):619–668, 2009.
13. H.-A. Jacobsen, A. K. Y. Cheung, G. Li, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh. The padres publish/subscribe system. In *Principles and Applications of Distributed Event-Based Systems*, pages 164–205. 2010.
14. M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann. Self-organizing broker topologies for publish/subscribe systems. In *SAC*, SAC '07, 2007.
15. G. Li, V. Muthusamy, and H.-A. Jacobsen. Adaptive content-based routing in general overlay topologies. In *Middleware*, pages 1–21, 2008.
16. D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *PODC*, 2002.
17. N. Lipovetzky and H. Geffner. Searching for plans with carefully designed probes. In F. Bacchus, C. Domshlak, S. Edelkamp, and M. Helmert, editors, *ICAPS*. AAAI, 2011.
18. M. Migliavacca and G. Cugola. Adapting publish-subscribe routing to traffic demands. In *DEBS*, pages 91–96, 2007.
19. D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
20. M. Onus and A. Richa. Minimum maximum-degree publish/subscribe overlay network design. *Networking, IEEE/ACM Transactions on*, 19(5):1331 –1343, oct. 2011.
21. J. Reumann. Pub/Sub at Google, 2009. Lecture at EuroSys and CANOE Summer School.
22. S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)*, 39:127–177, 2010.
23. Y. Yoon, V. Muthusamy, and H. Jacobsen. Foundations for highly available content-based publish/subscribe overlays. In *ICDCS*, pages 800 –811, 2011.