Planning in the Large: Efficient Generation of IT Change Plans on Large Infrastructures

Sebastian Hagen*, Weverton Luis da Costa Cordeiro†, Luciano Paschoal Gaspary†,
Lisandro Zambenedetti Granville†, Michael Seibold*, and Alfons Kemper*
* Institut für Informatik, Technische Universität München (TUM), 85748 Garching, Germany
email: {hagen, seibold, kemper}@in.tum.de
† Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
email: {wlccordeiro, paschoal, granville}@inf.ufrgs.br

***Abstract*—Change Management, a core process of the Information Technology Infrastructure Library (ITIL), is concerned with the management of changes to networks and services to minimize costly disruptions on the business. As part of Change Management, IT changes need to be planned. Previous approaches to automatically generate IT change plans struggle, in terms of scalability, to properly deal with large Configuration Management Databases (CMDBs). To enable IT change planning in the large, in this paper we discuss and analyze optimizations for refinement-based IT change planning over object-oriented CMDBs. Our optimizations reduce the runtime complexity of several key operations part of refinement-based IT change planning algorithms. A sensitivity analysis shows that our optimizations outperform SHOP2 - the winner of a previous comparison among IT change planners - in terms of runtime complexity for several important characteristics of IT changes and CMDBs. A cloud deployment case study of a Three-tier application and a virtual network configuration case study demonstrate the feasibility of our approach and confirm the results from the sensitivity analysis: IT change planning has evolved from planning in the small to planning in the large.**

***Keywords*-ITIL Change Management, automated planning, IT change plan generation, artificial intelligence**

I. Introduction

The Information Technology Infrastructure Library (ITIL) [1] is a set of best practices on how to manage IT services and operations within a company that has become an important guideline for the implementation of proper IT Service Management. Change Management [2] is part of ITIL and ensures that changes to hardware and software are managed and conducted in a way that costs are met, risks are reduced, and business needs of a company are satisfied. To ensure that, ITIL introduces a Change Management process comprising the evaluation, authorization, planning, testing, scheduling, implementation, documentation, and review of IT changes [2].

IT change planning, an important step of this process, is concerned with the generation of detailed, executable, IT change plans to accomplish high-level Request for Changes (RFCs) [2]. IT change plan generation has been subject to several investigations, addressing topics such as the integration of planning and scheduling [3], reuse of knowledge in IT change design [4], algorithms for the hierarchical refinement of IT changes [5] [6] or algorithms that are not based on refinement [7] [8] [9]. Common to these works is that they have been evaluated to work on small to medium size IT change planning problems and that scalability to large CMDBs has either been out of the scope of these work or cannot be achieved by them. To make a progress towards planning in the large, we recently [10] compared different planning algorithms for IT change planning in terms of scalability and concluded that *Hierarchical Task Network* (HTN) algorithms are most suitable for IT change planning in terms of performance and usability by an IT change manager. However, even *SHOP2* [11] - an HTN planner and the winner of that comparison - struggles to maintain acceptable planning performance on large Configuration Management Databases (CMDBs) once they surpass a couple of thousand Configuration Items (CIs).

The contribution of this paper is the proposal and evaluation of optimization techniques for refinement-based IT change planning algorithms that make IT change planning feasible for even large infrastructures, such as today's Infrastructure-as-a-Service clouds. Our optimizations reduce the runtime complexity of several key operations part of the simple task network planning algorithm that is used by SHOP2 [11] and our IT change planning system. In a sensitivity analysis we examine the influence of several important characteristics of IT changes and the CMDB on the runtime complexity of the proposed optimizations. The analysis shows that our optimizations outperform SHOP2 [11] (the winner of a previous comparison among IT change planners [10]) in terms of runtime complexity for a large percentage of IT changes and CMDBs. In addition to that, the optimizations proposed herein are more robust in respect to the characteristics of IT changes and the CMDB, *i.e.*, the planner's runtime is less influenced by the characteristics of an IT change or the CMDB. A cloud deployment case study of a Three-tier business application and a virtual network configuration case study demonstrate the feasibility of the approach and confirm the results from the sensitivity analysis. Our optimizations beneficially influence related change management problems as well: A faster planner helps to improve the quality of plans as it can generate more plans within the same time and select the best one among them. Furthermore, more time is left to also schedule IT changes into change windows - a step traditionally kept separately from change planning because of its complexity.

The remainder of this paper is organized as follows: In Section II we discuss related work and contributions. Section III describes the planning algorithm and introduces the optimizations to improve SHOP2. In Section IV we compare both planning systems in a sensitivity and robustness analysis followed by a comparison using case-studies in Section V. Finally, we conclude in Section VI.

II. Related Work and Contributions

The *CHAMPS* system by Keller *et al.* [3] is the seminal work to address IT change planning. The work proposes a planning algorithm that traverses a dependency tree to derive a plan. Different to the planning algorithm used herein, CHAMPS can only plan for IT changes once dependencies among them have been explicitly specified and cannot infer a plan from preconditions and effects of IT changes. The applicability and optimization of planning over large infrastructures has not been explored by CHAMPS.

Maghraoui *et al.* [8] apply an optimized *Artificial Intelligence* (AI) planning algorithm (UCPOP) to IT change planning. Although the authors use a different case study, scalability is reported to be few hundred CIs. Our experience with UCPOP for IT change planning is that its performance is worse than that of all other planners we previously examined [10]. In addition to that, UCPOP relies on first-order unification of which we show herein that it has serious scalability issues.

In a previous work we proposed the *ChangeLedge* system [4] and a constraint aware refinement approach [12] for IT change planning. These works focus on the reuse of knowledge in IT change design by capturing best practices and plan generation taking into account the effects IT changes have on one another. Similar to CHAMPS, this early research on change plan generation does not address planning in the large.

Trastour *et al.* [6] propose *ChangeRefinery*, a system for operator-assisted refinement of IT change plans. SHOP2, ChangeRefinery, and our work share the same task network refinement algorithm. However, Trastour *et al.* report scalability to a few hundred CIs. Applicability to very large CMDBs, a comparison with SHOP2, and a complexity analysis has been out of the scope of their work.

In a previous work [5] we propose a hybrid HTN and state-space planning algorithm for IT change planning. Different to Trastour *et al.* [6], it supports planning according to state-based constraints as well. A performance comparison with SHOP2, runtime complexity optimizations, and scalability to large CMDBs have been out of the scope of that work.

Hagen *et al.* [9] also present a state-space planner for IT changes. The approach can only be used to plan for a small subset of IT changes. However, the focus on a smaller set of IT changes delivers scalability to several thousand CIs - still significantly less than the numbers reported herein while expressiveness is also larger for HTN planning.

In a recent work [13] we propose an algorithm for the adaptation of unfeasible change plans due to unpredictable changes to the IT infrastructure. Although not stressed in this work, the plans constructed by our algorithm are ready for adaptation using the techniques previously presented [13].

In a previous work [10], we compared several general purpose planners for IT change planning. This work extends the previous comparison on IT change planners [10] by proposing and evaluating optimizations to further reduce the runtime complexity of the algorithm that was found best suitable for IT change planning.

Several other aspects of IT management have been addressed in recently published work. Among these are important steps to schedule IT changes [14], to improve the process to manage IT incidents [15] or to staff people [16], and towards connecting risk analysis with IT change prioritization [17].

There are additional relevant research efforts published in the field of automated planning, which explore the generation of plans as well. However, all AI planners rely on a first-order satisfiability solver that works fine for smaller problem instances but - as shown in this work - does not scale to domain sizes as they appear for IT change planning. Due to this reason, and the fact that these works have not addressed planning for IT changes, they are not approached in this paper.

It is important to emphasize that our work aims at being generic enough for several controlled, well known IT environments. Examples include private data centers and cloud computing. As for the specific management interfaces required for IT changes, we rely on the research communities' efforts that have been carried out to standardize them. Important examples can be found in the context of cloud computing, where such efforts [18], [19] have been driven towards the interoperability across different cloud providers.

III. Hierarchical Task Network Planning (HTN)

A. Introduction to HTN planning

For IT change planning based on *Hierarchical Task Network Planning* (HTN), a planning problem is given by a Request for Change for which a sequence of IT activities needs to be computed so as to accomplish the objective of the RFC. Two different types of activities, *i.e.*, IT tasks, are distinguished: *abstract* activities and *atomic* activities. Abstract IT activities are those IT tasks expressed on a high level that are not directly executable. They need to be further (recursively) refined into finer grained activities until a sequence of atomic activities has been determined to implement the abstract activity. The goal to plan for is provided by the RFC which is directly mapped onto an abstract activity. For example, Fig. 1 depicts a decomposition tree for the abstract activity (gray) / RFC to deploy a database. The rules to refine abstract activities are called (HTN) *methods*. Several refinement rules, which represent different problem solving strategies, can exist to decompose an abstract activity. Atomic activities (white) in turn cannot be further refined because they are the most basic IT activities out of which abstract activities are composed. An atomic activity carries the specification of a precondition that needs to hold to apply it and a specification of its effects. A *Simple Task Network (STN) Planner*, a simpler form of HTN planning, is a forward-chaining (starting from the current state of the CMDB) ordered, depth-first search tree-decomposition algorithm. The computed plan to implement the RFC only comprises the leaf nodes of the decomposition

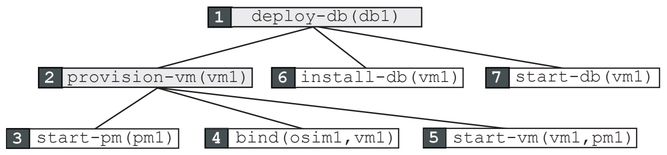


Fig. 1. HTN decomposition tree of abstract IT activity to deploy a database.

tree, *i.e.*, the atomic activities. For example, the computed plan to deploy a database for the decomposition tree in Fig. 1 comprises atomic activities 3, 4, 5, 6, and 7. The planning algorithm guarantees that preconditions and effects of the atomic activities complement each other in such a way that the plan is executable.

B. Algorithm

Algorithm 1 Total-order STN IT Change Planning Algorithm

```
 1: procedure PLAN(RFC, METHODS)
 2:     plan = <>                                          ▷ $O(1)$
 3:     stack = $< RFC >$                                  ▷ $O(1)$
 4:     while !stack.empty() do                            ▷ $O(1)$
 5:         act = stack.pop()                              ▷ $O(1)$
 6:         if act is atomic activity then
 7:             while !act.precondition() do               ▷ $O$(see Subsection IV-C)
 8:                 next-parameter-binding(act)            ▷ $O$(see Subsection IV-C)
 9:             end while
10:             if act.precondition() then                 ▷ $O$(see Subsection IV-C)
11:                 act.applyeffects()                     ▷ $O$(see Subsection IV-E)
12:                 plan = plan ∘ act                      ▷ $O(1)$
13:             else                                       ▷ atomic activity not applicable
14:                 if !backtrack() then                   ▷ $O$(see Subsection IV-D)
15:                     return $fail$                      ▷ If backtracking fails, no plan exists
16:                 end if
17:             end if
18:         else                                           ▷ $act$ needs to be further decomposed
19:             choose parameters for act and $m \in methods$ that $m$ can
20:             be decomposed by $act$                     ▷ $O$(see Subsection IV-C)
21:             if act.precondition() then
22:                 $< act_1, ..., act_n >$ = act.decompose(m)     ▷ $O(1)$
23:                 stack.push($< act_n, ..., act_1 >$)            ▷ $O(1)$
24:             else                                       ▷ act not refinable
25:                 if !backtrack() then                   ▷ $O$(see Subsection IV-D)
26:                     return $fail$                      ▷ If backtracking fails, no plan exists
27:                 end if
28:             end if
29:         end if
30:     end while
31:     return $plan$                                      ▷ Planning successful, return plan
32: end procedure
```

Algorithm 1 depicts the basic total-order STN planning algorithm, previously proposed by Nau *et al.* [11], that is the foundation of SHOP2 [11] and our IT change planner. However, key operations of the algorithm have been optimized - often making the difference between constant, linear, and polynomial runtime. The optimizations are discussed in Subsection III-C. The algorithm works as follows: As long as the stack of IT activities to plan for is not empty (Line 4) an activity is popped from the stack and planned for (depth-first search). If act is atomic (Line 6), its parameters are adapted (Line 8) until its precondition is satisfied. For example, when planning for the atomic `start-pm` activity in Fig. 1 the physical machine (PM) parameter needs to be set to a physical machine that satisfies the precondition that comes with the `start-pm` activity. For instance, the PM chosen needs to be in state off. In Subsection IV-C we compare the runtime complexity to find valid parameter bindings. If the atomic activity is executable (Line 10), its effects are applied to the CMDB (Line 11, see Subsection IV-E) and it is added to the end of the plan (Line 12). The planner has to backtrack if the atomic activity is not applicable, *i.e.*, at a previous decision point an abstract activity has to be decomposed differently or parameters of an activity have to be altered to explore a different plan. For example, assume that in Fig. 1 activity 6 to install the database on $vm1$ might not be executable. The planner would have to backtrack over activities 5, 4, and 3 to introduce a different virtual machine (VM) for deployment in activity 2. Subsection IV-D examines the backtracking performance for both planners. Similar to atomic activities, parameters have to be adapted such that a method can be used to decompose an abstract IT activity (Lines 19-20). For example, in Fig. 1 the VM parameter of activity `provision-vm` can be restricted to specific VMs depending on the precondition of activity 2 and the method used to decompose it. If a method is applicable, the method is used to decompose the abstract activity and the children are pushed on the stack (Line 23). If no decomposition is possible (Line 24), the planner needs to backtrack to choose different parameters or another decomposition alternative for an abstract activity that has been previously planned for.

C. Unification vs. Object-Oriented Planning

In this subsection we describe optimizations for several critical operations of Algorithm 1 that make the difference between constant, linear, and polynomial planning complexity compared to SHOP2 - previously to be found the fastest planner in a comparison of IT change planners [10]. Note that our optimizations do not depend on a specific programming language. However, some optimizations depend on the availability of pointers / references, which can typically be found in object-oriented languages.

1) Evaluation of preconditions: To decide whether methods or atomic activities are applicable, preconditions need to be evaluated over CIs. This happens several times in Alg. 1, *e.g.*, in Lines 7, 10, and 21. To evaluate a precondition, SHOP2 has to unify the precondition. Unification is an algorithmic process which tries to solve the satisfiability problem. The goal of unification is to find a substitution such that two seemingly different terms (the term describing the precondition and the terms stored in the CMDB describing the configuration of the CIs) are equal. Unification is a potentially expensive operation (between linear and polynomial runtime complexity). Readers interested in understanding the complexity, the logical background, and the algorithms for unification may be referred to [20], [21]. Different to SHOP2, we do not make use of unification to evaluate preconditions. Our preconditions are implemented as Boolean methods that are executed over an object-oriented CMDB model. As the properties of the CIs addressed by the precondition can be accessed in constant time (through references / pointers), a precondition can be evaluated in constant time.

2) Determining parameters of IT activities: During planning, parameters of an IT activity need to be linked to CIs, *e.g.*, in Lines 8 and 19, so as to satisfy the activity's precondition. To determine parameter bindings for activities, SHOP2 makes use of unification [20], [21], more precisely an algorithm previously proposed by Nilsson [22]. Different to SHOP2, we do not use unification but determine CIs of parameters by simply iterating over the CMDB until an assignment of CIs to parameters is found that satisfies the precondition. For this purpose we scan through the CMDB in linear time (to determine one parameter) which is quite often faster than the polynomial complexity caused by Nilsson's algorithm [22].

3) Application of effects: During planning, the effects of atomic activities need to be applied (Line 11, Alg. 1) to the CMDB to account for the influence on IT activities being planned for later. SHOP2 describes effects as a list of predicates to be added and deleted from the CMDB which is maintained as a list of predicates. In the worst-case, this operation involves a linear scan over the CMDB to find the predicate to remove it and eventually a costly unification step should predicates only be known partially. In turn, our CMDB is described by a set of (Java) objects each matching to a Configuration Item. Our effects are implemented as Java-methods which implement code that change these objects, i.e., the CIs of the CMDB, according to the effects of change activities. By the use of references and pointers the affected CIs can be accessed in constant time and the effects can be applied to them.

4) Undo of effects: When backtracking over a previously planned atomic activity (Lines 14 and 25), the effects of the activity need to be undone. SHOP2 needs to conduct a linear scan of all predicates to revert activities by removing predicates previously added by the activity to the CMDB. We optimize this process to constant runtime because our CIs maintain a copy of the original values of their properties to quickly restore them should changes need to be undone.

IV. Sensitivity and Robustness Analysis

A. Design of Microbenchmarks

Microbenchmarks (MBs) are used to practically analyze the runtime-complexity of the non-trivial parts of Alg. 1 depending on several characteristics of IT activities and the CMDB. Among these are the complexity of an activity's precondition or effects, the layout of the CMDB, and the proportion of CIs that qualify to satisfy preconditions. Note that we do not provide a theoretical complexity analysis because it would go far beyond the scope of this paper and it is practically impossible to analyze the complexity of a planner as complex as SHOP2. For readers interested in a rough analysis we recommend surveys on first-order unification [20], [21] which address the runtime complexity of unification algorithms. SHOP2 makes frequently use of unification and thus is strongly influences by the complexity of unification algorithms. Our optimization of the STN algorithm (Alg. 1) and the runs of the micro benchmarks have been carefully engineered such that the same search space is explored. Thus, SHOP2 and our optimized algorithm decompose and apply activities in the same order, choose parameters in the same order, and backtrack at the same time over the same activities. It is also important to keep in mind that the MBs do not have side-effects. Thus, their individual results can be combined to assess the runtime when combining different types of IT activities over different instances of a Configuration Management Database in a case study. Consequently, a case study is a weighted, cumulative combination of microbenchmarks because they individually stress a specific operation of Alg. 1 while solving other operations in constant time.

B. Parameters of Sensitivity Analysis

1) CMDB Layout: Let $p1$ be a parameter of IT activity act and $pre(p1)$ a precondition of act whose truth value depends on the value assigned to $p1$. Each Configuration Item (CI) can appear in either one of two roles in respect to $pre(p1)$: A CI ci is called **qualified CI** in respect to act and precondition $pre(p1)$ if pre accounts when the value of $p1$ is set to ci, *i.e.*, $pre(p1 := ci) == true$. A CI ci is called **unqualified CI** in respect to act and precondition $pre(p1)$ if pre does not account when the value of $p1$ is set to ci, *i.e.*, $pre(p1 := ci) == false$. For example, consider an activity act to start a database. A precondition p of act is that the database is installed. Thus, all database CIs currently in state installed are qualified CIs in respect to act and precondition p. Unqualified CIs are a threat to the planner's performance because they cannot be used to instantiate an IT activity such that its precondition is satisfied. Consequently, the existence of unqualified CIs can prolong the search for proper parameter bindings or can cause more backtracking. SHOP2 and our approach maintain the CMDB as a list of CIs. Thus, we distinguish two layouts of a CMDB: In **qualified-unqualified (Q-UQ) layouts** all qualified CIs appear before all unqualified CIs. In turn, in an **unqualified-qualified (UQ-Q) layout** all unqualified CIs appear at the front of the CMDB. Both layouts cover the best- and worst-case situations of SHOP2 and our algorithm in terms of the distribution of CIs in the CMDB. Consequently, we do not consider distributions for the location of CIs stored in the CMDB because the best- and worst-case are covered by the Q-UQ and UQ-Q layouts.

2) Selectivity: While the layout of the Configuration Management Database (CMDB) describes the order in which qualified and unqualified CIs appear in the CMDB, it does not characterize the percentage of qualified and unqualified CIs existing in the CMDB. The selectivity sel denotes the proportion of qualified CIs among all CIs of the CMDB. Thus, $sel = \frac{|\text{qualified CIs}|}{|\text{CMDB}|}$. Large selectivity means that many CIs can be chosen as valid parameters for IT activities, thus, leaving less wrong choice opportunities for the planner. The benchmarks consider three values for selectivity to assess a performance trend: **(1) 100% selectivity:** The CMDB only comprises qualified CIs, *i.e.*, all CIs of a certain type are suitable to render the precondition of an IT activity true. Note that the CMDB layout does not have any influence in this case because unqualified CIs do not exist. **(2) 50% selectivity:** Half of the CMDB's CIs qualify. Depending on the CMDB layout these are either located at the front (Q-UQ) or the end of the

	Microbenchmark 1 parameters				Complexity	
	CMDB	Pre. in	#CIs	Selectivity	OWN	SHOP2
(1)	Q-UQ	Method	1	0%	**const**	**linear**
(2)	Q-UQ	Method	1	50%, 100%	**const**	**polyn.**
(3)	Q-UQ	Method	2	0%, 50%, 100%	**const**	**polyn.**
(4)	Q-UQ	Activity	1	0%, 50%, 100%	**const**	[**const** : **linear**]
(5)	Q-UQ	Activity	2	0%, 50%	**const**	**polyn.**
(6)	Q-UQ	Activity	2	100%	**const**	[**const** : **linear**]
(7)	UQ-Q	Method	1	0%	**linear**	**linear**
(8)	UQ-Q	Method	1	50%	**linear**	**polyn.**
(9)	UQ-Q	Method	1	100%	**const**	**polyn.**
(10)	UQ-Q	Method	2	0%, 50%	**polyn.**	**polyn.**
(11)	UQ-Q	Method	2	100%	**const**	**polyn.**
(12)	UQ-Q	Activity	1	0%, 50%	**linear**	[**const** : **linear**]
(13)	UQ-Q	Activity	1	100%	**const**	[**const** : **linear**]
(14)	UQ-Q	Activity	2	0%, 50%	**polyn.**	[**const** : **linear**]
(15)	UQ-Q	Activity	2	100%	**const**	[**const** : **linear**]

(a) Runtime complexity of Microbenchmark 1

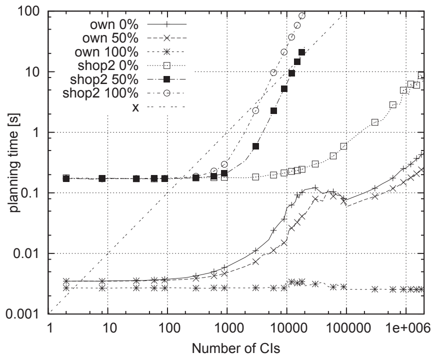
	Microbenchmark 2 parameters			Complexity	
	CMDB	effect?	Selectivity	OWN	SHOP2
(1)	Q-UQ	yes	0%, 50%, 100%	**const**	**polyn.**
(2)	Q-UQ	no	0%, 50%, 100%	**const**	**polyn.**
(3)	UQ-Q	yes	0%, 50%	**linear**	**polyn.**
(4)	UQ-Q	yes	100%	**const**	**polyn.**
(5)	UQ-Q	no	0%, 50%	**linear**	**polyn.**
(6)	UQ-Q	no	100%	**const**	**polyn.**

(b) Runtime complexity of Microbenchmark 2

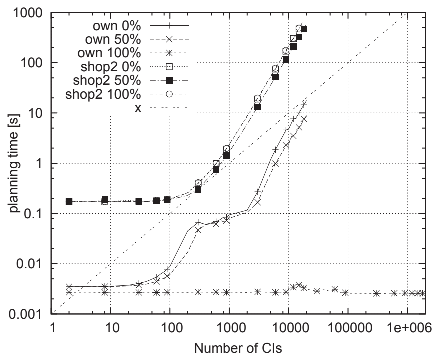
	Microbenchmark 3 parameters		Runtime complexity	
	CI at front	#effect?	OWN	SHOP2
(1)	yes	1, 4, 8	**const**	**linear**
(2)	no	1, 4, 8	**const**	**linear**

(c) Runtime complexity of Microbenchmark 3

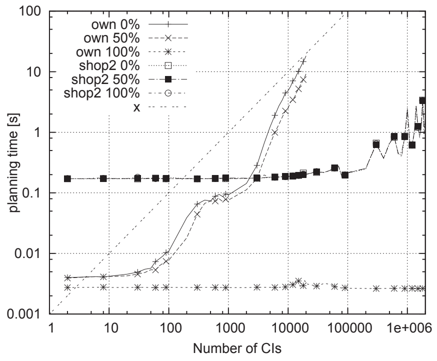
TABLE I
INFLUENCE OF CHARACTERISTICS OF IT CHANGES AND THE CMDB ON THE RUNTIME COMPLEXITY OF THE PLANNERS



(a) MB1 (7,8,9): UQ-Q, Method, 1 CI



(b) MB1 (10,11): UQ-Q, Method, 2 CIs



(c) MB1 (14,15): UQ-Q, Activity, 2 CIs

Fig. 2. Selected measurements of Microbenchmark 2 with similar runtime complexity for OWN and SHOP2

CMDB (UQ-Q). **(3) 0% selectivity:** The CMDB comprises the minimal number of qualified CIs for the planning problem to be still solvable. The CMDB layout determines whether these CIs appear at the beginning (Q-UQ) or the end of the CMDB (UQ-Q). The best- and worst-case runtimes are always covered by 0% and 100% selectivity. Similar to the CMDB layout, we omit a distribution for the selectivity as we are interested in a comparison of best- and worst-case performance.

C. Influence of Complexity of Preconditions

Microbenchmark 1 (MB1) compares the performance of both planners when it comes to adapt parameters of activities such that their preconditions are satisfied (Lines 7, 8, and 10) or that they can be decomposed by a refinement method (Lines 19-20). Besides the CMDB layout and the selectivity, the following parameters also influence the matching of preconditions: (1) the number of CIs the precondition addresses and (2) whether parameters are chosen in an atomic activity, *i.e.*, in Line 8, or during decomposition of an abstract activity, *i.e.*, in Lines 19-20 of Alg. 1. Table Ia depicts the complexity results for all 24 configurations of MB1. Our optimizations result in constant runtime on qualified-unqualified (Q-UQ) CMDBs as the first CIs chosen for the parameters satisfy the precondition. SHOP2 shows polynomial and linear runtime for Q-UQ CMDBs depending on whether the unification algorithm (see Subsection III-C2) to compute parameter bindings scales linearly or polynomially with the size of the CMDB. Thus, for Q-UQ CMDBs our planner always outperforms SHOP2 in runtime complexity. For unqualified-qualified (UQ-Q) CMDBs runtime complexity is closer. Figure 2 depicts the $log - log$[1] plots of planning time depending on CMDB size for three interesting UQ-Q CMDB cases of MB1. Figure 2a depicts the case of matching over one CI inside a method (Line 19-20, Alg. 1) on a CMDB where unqualified CIs precede qualified CIs (UQ-Q layout). For 100% selectivity our optimizations have constant runtime, because in this case the CMDB only comprises qualified CIs and the first choice for a parameter satisfies the precondition. Conversely, SHOP2 has polynomial complexity. For 50% selectivity SHOP2 still shows polynomial complexity because of the unification costs (see Subsection III-C2) and our optimizations only degrade to linear runtime because the planner has to search the prefix of unqualified CIs at the front of the CMDB until a qualifying CI is found. For 0% selectivity SHOP2's runtime becomes linear - the same complexity as our planner. However, our planner outperforms SHOP2 for CMDBs of up to 2 million CIs.

Note that the actual runtime complexity of our algorithm is

[1]In $log - log$ plots the processing complexity is $O(\alpha n) \leftrightarrow slope == 1$, $O(\alpha n^{slope}) \leftrightarrow slope > 1$ and sub-linear for $slope < 1$. All measurements were made on an Intel Xeon Processor with 2.8Ghz and 4 GB of RAM.

sometimes disguised by Java's *Just In Time Compiler* (JITC). For example, in Fig. 2a, for 0% and 50% selectivity we can observe linear growth between 1,000 and 10,000 CIs. Then, the JITC jumps in and compiles frequently used code for native execution instead to interpret it in the Java virtual machine. Performance becomes even faster for larger problem instances (between 10,000 and 100,000) because more code is being compiled with increasing size of the CMDB. For even larger problem instances runtime however tends to increase again once the performance advantage of compiled code vs. interpreted code is outweighed by the increasing problem size (beyond 100,000 CIs). However, growth in runtime is less strong than at the beginning as compiled code takes less execution time.

Figure 2b depicts a similar configuration of MB1 but the precondition needs to be matched over two CIs. Again, our optimizations result in constant runtime at 100% selectivity because the CIs chosen first satisfy the precondition. SHOP2 has polynomial runtime for all selectivities, similar to our optimized version for 0% and 50% selectivity because it has to search the whole prefix of unqualified CIs of the CMDB until the first pair of qualifying CIs is found. The larger the selectivity, the smaller the UQ part of the CMDB, resulting in faster planning times. Despite having the same runtime complexity as SHOP2, our approach nevertheless outperforms SHOP2 (see Fig. 2b). Figure 2c depicts the only case of MB1 in which SHOP2 has better runtime complexity than our optimizations: matching a precondition over two CIs in an atomic activity on a UQ-Q CMDB. For 100% selectivity, the CMDB only comprises qualified CIs immediately yielding a valid choice of CIs for our approach. For lower selectivities a polynomial search through the prefix of unqualified CIs to find the first pair of matching qualified CIs becomes necessary for our approach. However, SHOP2 always shows linear runtime on unqualified-qualified (UQ-Q) CMDBs when matching the precondition in an activity because unification in activities matches from the back of the CMDB leading to an immediate hit because of the UQ-Q layout. All in all, our approach outperforms SHOP2 in 22 out of 24 configurations of MB1 (92%).

In addition to that, our optimizations are more robust to inputs - an important benefit when an algorithm-agnostic IT operator formalizes or tunes a change planning domain. In 50% of all configurations of Microbenchmark 1 our planner's runtime is independent of the selectivity of CIs. In turn, this only accounts for 16% of all configurations for SHOP2. Similarly, in 100% vs. 14% of all configurations our planner's performance is the same for matching a precondition in an atomic or abstract activity. In 66% vs. 33% of all configurations of MB1 our optimizations show the same performance when matching one or two preconditions.

D. *Influence of Backtracking*

Backtracking is an inherent part of the planning algorithm (Alg. 1, Lines 14 and 25). During backtracking previously planned IT activities need to be undone and one alternative decomposition choice needs to be made. *Microbenchmark 2* (MB2) measures the runtime complexity taking into account (1) the layout of the CMDB (Subsection IV-B1), which determines whether backtracking occurs, (2) the selectivity (Subsection IV-B2) influencing the extent to which backtracking occurs, and (3) whether the planner has to revert effects during backtracking. Table Ib depicts the runtime complexity of both planners for all 12 configurations of MB2. SHOP2's runtime complexity is always polynomial because with linear increasing size of the CMDB the time spent for first-order unification increases polynomially. We can observe a polynomial increase in the number of calls made to Nilsson's first-order unification algorithm (see Subsection III-C2). In 66% of all cases - when qualifying CIs are ordered at the beginning of the CMDB or selectivity equals 100% - our algorithm solves the backtracking benchmark in constant time because backtracking does not occur for this layout or selectivity as the first choice made for the parameters satisfies the preconditions right away (see Subsection III-C2). In 33% of the cases - when unqualified CIs exist at the front of the CMDB - our approach shows linear runtime complexity because our optimized version of the algorithm has to backtrack through a prefix of unqualified CIs at the front of the CMDB. In these cases, planning time increases linearly with decreasing selectivity because the number of unqualified CIs at the front of the CMDB increases linearly.

Besides planning performance, robustness of planning time is important as well. A robust planner delivers the same performance independent of a specific characteristic of an IT activity or the CMDB. For example, the planning duration might be independent of the CMDB layout or the selectivity. Although not shown in Table Ib, for Q-UQ layouts selectivity does not influence the runtime of SHOP2 and our approach. Different to UQ-Q layouts, where selectivity influences the planning duration. For both planners backtracking without effects is cheaper than backtracking with effects, whereas this gap is larger for SHOP2 than for our approach because of the different complexity to undo activities (see Subsection III-C4). All in all, both planners show similar robustness for MB2 but our optimizations outperform SHOP2 in all cases.

E. *Influence of Complexity of Effects*

Microbenchmark 3 (MB3) analyzes the influence that the complexity of effects of atomic activities have on the runtime of the planner (Line 11). The following parameters influence the costs to apply effects: (1) the positions of the CIs in the CMDB that are changed by the effects of the atomic activity (either at the front or the back of the CMDB) and (2) the number of CIs affected by the activity (1, 4, or 8). For each CI affected, a mixture of six blind writes, arithmetic effects, and reference manipulation effects is applied to the CI. MB3 measures the time to perform 10 IT activities to distinct CIs. Table Ic depicts the complexity results for the six configurations of MB3. Our approach always outperforms SHOP2 by constant instead of linear runtime because our optimizations can access CIs in constant time to directly apply an effect. Consequently, our approach is insensitive to the location of a CI in the CMDB and the size of the CMDB

	Case study parameters			Runtime complexity	
	CMDB	BT possible	Selectivity	OWN	SHOP2
(1)	Q-UQ	Yes	0%	**const**	**polyn.**
(2)	Q-UQ	Yes	50%	**const**	**polyn.**
(3)	Q-UQ	Yes	100%	**const**	**polyn.**
(4)	Q-UQ	No	0%	**const**	**polyn.**
(5)	Q-UQ	No	50%	**const**	**polyn.**
(6)	Q-UQ	No	100%	**const**	**linear**
(7)	UQ-Q	Yes	0%	**linear**	**polyn.**
(8)	UQ-Q	Yes	50%	**linear**	**polyn.**
(9)	UQ-Q	Yes	100%	**const**	**polyn.**
(10)	UQ-Q	No	0%	**linear**	**polyn.**
(11)	UQ-Q	No	50%	**linear**	**polyn.**
(12)	UQ-Q	No	100%	**const**	**linear**

(a) Complexity results for three-tier application deployment

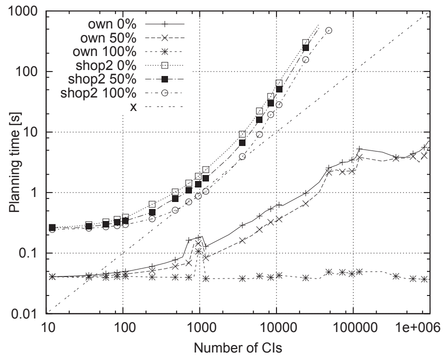
	Case study parameters		Runtime complexity	
	CMDB	Selectivity	OWN	SHOP2
(1)	Q-UQ	0%, 50%, 100%	**const**	**polyn.**
(2)	UQ-Q	0%, 50%	**polyn.**	**polyn.**
(3)	UQ-Q	100%	**const**	**polyn.**

(b) Complexity results virtual network configuration case study

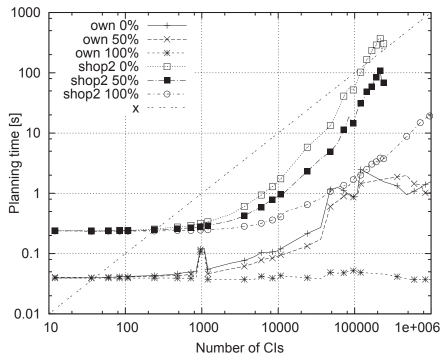
	Case study parameters		Runtime complexity	
	CMDB	Selectivity	OWN	SHOP2
(1)	NW-PMS	0%, 50%, 100%	**const**	**linear**
(2)	PMS-NW	0%, 50%, 100%	**const**	**linear**

(c) Complexity results virtual network unconfiguration case study

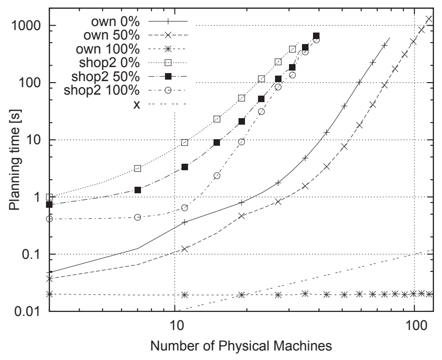
TABLE II
INFLUENCE OF CHARACTERISTICS OF IT CHANGES AND THE CMDB ON THE RUNTIME OF THE CASE STUDIES



(a) Deployment case study : UQ-Q, BT possible



(b) Deployment case study : UQ-Q, no BT possible



(c) Network configuration: UQ-Q CMDB

Fig. 3. Selected measurements of the Three-tier application deployment and the virtual network configuration case studies

when applying an effect. In contrast, SHOP2's performance to apply an effect to a CI depends on its location and the size of the CMDB: The more to the front of the CMDB the CI is located or the larger the CMDB, the more expensive the application of the effects. This is caused by the different costs in unification and adding / removing predicates to the CMDB. For both planners the planning time increases with the number of CIs affected by the activity. All in all, our implementation outperforms SHOP2 in all cases and is more robust regarding the position of a CI in the CMDB - a benefit as order of CIs in the CMDB is not controlled by a change manager.

V. Change Planning Case Studies

In this section we evaluate the performance of both planners using case studies instead of isolated characteristics.

A. Deployment of a Three-tier Application

1) Description of case study: In this case study we evaluate the planning time to deploy a Three-tier application (*Database* (DB), *Web Application Server* (WAS), and *Loadbalancer* (LB)). The case study comprises 28 abstract IT activities, 48 different refinement rules, and 16 atomic IT activities. We describe next the constraints according to which a plan is created: (1) Placement constraints: place VMs on suitable PMs, connect available OS images to VMs, place DB, WAS, and LB on VMs. The concrete placement constraint does not influence runtime because the time to evaluate a constraint over a CI can be neglected. However, the CMDB layout and the selectivity of CIs for a placement constraint influence the planning duration. (2) State-based constraints: a PM needs to be on to start a VM on it, an OS image needs to be mounted to start a VM, a DB needs to be installed/running to install/start a WAS, a WAS needs to be installed/running to install/start a LB. The final plan for the deployment comprises atomic activities to bind resources (according to the constraints in (1)), activities to mount OS images, change the state of PMs and VMs, and install and start DB, WAS, and LB according to the precedence constraints in (2).

2) Evaluation: CMDB layout and selectivity have a performance influence on the deployment case study because they influence how fast suitable CIs can be found during deployment. In addition to that, we examine different planning domains, i.e., problem solving strategies, to solve the case study. One does not prevent backtracking (see BT possible column in Table IIa) during planning because it commits to parameter bindings of IT changes high up in the decomposition tree that later need to be backtracked over because they do not satisfy the precondition of lower-level atomic activities. Another domain solves the problem without backtracking because placement decisions are made at the last possible moment. We consider these two domain descriptions because it mainly depends on the skills and experience of an IT operator how efficiently a planning domain is written: the one without backtracking requires comparatively more advanced knowledge and experience in formalizing refinement-based planning domains.

Table IIa summarizes the complexity results obtained for the Three-tier deployment case study. Our optimizations outperform SHOP2's runtime complexity in all cases. For qualified-unqualified (Q-UQ) CMDBs our optimizations have constant runtime as the initial choice made for parameters is correct and backtracking cannot occur due to the CMDB layout. In turn, SHOP2 shows polynomial runtime behavior on qualified-unqualified (Q-UQ) CMDBs despite the case where selectivity is 100% and the domain is engineered to solve the problem without backtracking. Similar to the microbenchmarks, the runtime of SHOP2 is dominated by the time to perform first-order unification.

Figure 3 depicts the runtime on UQ-Q CMDBs for the domain that does not prevent backtracking (Fig. 3a) and the one that avoids backtracking (Fig. 3b). On UQ-Q CMDBs our optimizations degrade to linear runtime when selectivity is not 100% because our planner has to backtrack / search through a prefix of unqualified CIs at the front of the CMDB. For 100% selectivity the first choice made for parameters works, so we can observe constant runtime. In turn, SHOP2 can only derive a plan in polynomial time for UQ-Q CMDBs despite for 100% selectivity. When backtracking occurs (Figure 3a) our planner performs much better than SHOP2. For example, for 0% selectivity SHOP2 can solve a CMDB comprising 3,600 CIs within 10s while our optimizations easily solve problem instances of 1.2 million CIs within 10s (333x times faster). For the optimized domain without backtracking (Fig. 3b) both planners are closer together (7,200 CIs for SHOP2 vs. 108,000, within 1s, 15x faster).

B. *(Un)configuration of a Virtual Network*

1) Description of Case Study: In this case study we evaluate the complexity to configure virtual routers and links on top of a physical network substrate. More specifically, a virtual network is to be configured among three physical machines (pm_1, pm_2, pm_3) such that network connectivity exists between the pairs (pm_1, pm_2) and (pm_2, pm_3). The constraints taken into account during planning are those that occur when configuring a *Logical Router Service* on Juniper routers as imposed by the JUNOS operating system [23]: (1) a physical router cannot host more than 16 virtual routers and (2) every physical or logical interface can only be used by one virtual router. The planner needs to choose three PMs such that the interfaces and routers on the network path connecting the PMs can be properly configured according to constraints (1) and (2). Backtracking occurs if the configuration fails among the path. In this case a different choice of PMs is made and a new attempt to reconfigure (a different part of) the network is made. In a second RFC we evaluate the complexity to unconfigure the virtual network. To plan for both RFCs our planning domain description comprises 14 different abstract IT activities, 22 refinement rules, and 4 atomic activities (configure/unconfigure router/interface).

2) Evaluation: Table IIb depicts the runtime complexity to configure the virtual network. On qualified-unqualified (Q-UQ) CMDBs our optimizations outperform SHOP2 by constant instead of polynomial runtime complexity because the first choice made for CIs yields a valid plan. Again, SHOP2 spends polynomial time in performing first-order unification. For unqualified-qualified (UQ-Q) CMDBs our approach shows the same runtime complexity as SHOP2 for selectivities below 100%. In this case SHOP2 and our optimizations start backtracking as network configuration fails among all triples of PMs chosen from the UQ prefix until the first triple of PMs is chosen from the postfix of qualified CIs for the first time. Nevertheless, Fig. 3c shows that our approach searches the state space about twice as fast as SHOP2 (for the worst-case at 0% selectivity). All in all, our approach outperforms SHOP2 in all cases.

When planning for the RFC to undo a virtual network configuration (Table IIc), our approach always outperforms SHOP2 by constant instead of linear runtime. Undoing a network configuration is easier to handle by both approaches because more parameters of IT changes are already bound to CIs. For example, the physical machines between which to unconfigure the network are already given avoiding the polynomial complexity inherent to the configuration case-study. For the unconfiguration case study Nilsson's unification algorithm causes linear runtime for SHOP2. In turn, our optimizations solve the problem in constant time because we do not use unification but references to quickly derive the parameters of IT activities during decomposition.

VI. Conclusions

Planning in the large is an imperative feature required of IT change planners in order to cope with the increasing size and complexity of IT infrastructures in most modern organizations. In spite of this, investigations carried out in the context of IT change planning have either left this requirement out of scope, or simply do not deal satisfactorily with CMDBs that contain up to millions of configurations items.

To bridge this gap, in this paper we proposed and discussed optimization techniques for refinement-based IT change planning over object-oriented CMDBs. The proposed techniques enabled a substantial reduction in the runtime complexity of the refinement process: from polynomial to linear or even constant complexity, as evidenced in the various case studies and microbenchmarks presented in the paper.

Our evaluation has evidenced two important aspects regarding IT change planning. First, object-oriented planning and our optimizations have shown to be the most promising approach to deal with planning over very large CMDBs. This is of vital important to most modern organizations, which have to manage up to several change requests a day over large infrastructures in a timely manner. And second, first-order unification suffers from scalability issues. Consequently, typical AI planners cannot be satisfactorily used for IT change planning in the large, as they do not scale for very large CMDBs.

As prospective directions for future research, we suggest research towards algorithms that integrate planning and scheduling while maintaining scalability on large Configuration Management Databases.

References

[1] Cabinet Office, *ITIL Lifecycle Suite 2011 Edition*. The Stationery Office, 2011.

[2] S. Lacy and I. Macfarlane, *ITIL Service Transition*. The Stationery Office, 2007.

[3] A. Keller, J. Hellerstein, J. Wolf, K.-L. Wu, and V. Krishnan, "The CHAMPS System: Change Management with Planning and Scheduling," in *Proc. IEEE/IFIP Network Operations and Management Symp. (NOMS'04)*, Seoul, South Korea, Aug. 2004, pp. 395–408.

[4] W. L. da Costa Cordeiro, G. S. Machado, F. G. Andreis, A. D. dos Santos, C. B. Both, L. P. Gaspary, L. Z. Granville, C. Bartolini, and D. Trastour, "ChangeLedge: Change Design and Planning in Networked Systems based on Reuse of Knowledge and Automation," *Computer Networks: The Int. J. of Computer and Telecommunications Networking*, vol. 53, pp. 2782–2799, 2009.

[5] S. Hagen, N. Edwards, L. Wilcock, J. Kirschnick, and J. Rolia, "One Is Not Enough: A Hybrid Approach for IT Change Planning," in *Proc. IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM'09)*, Venice, Italy, Oct. 2009, pp. 56–70.

[6] D. Trastour, R. Fink, and F. Liu, "ChangeRefinery: Assisted Refinement of High-Level IT Change Requests," in *Proc. IEEE Int. Symp. on Policies for Distributed Systems and Networks (POLICY'09)*, London, UK, Jul. 2009, pp. 68–75.

[7] T. Eilam, M. H. Kalantar, A. V. Konstantinou, G. Pacifici, and J. Pershing, "Managing the Configuration Complexity of Distributed Applications in Internet Data Centers," *IEEE Communications Magazine*, vol. 44, pp. 166–177, 2006.

[8] K. E. Maghraoui, A. Meghranjani, T. Eilam, M. H. Kalantar, and A. V. Konstantinou, "Model Driven Provisioning: Bridging the Gap Between Declarative Object Models and Procedural Provisioning Tools," in *Proc. CM/IFIP/USENIX Int. Middleware Conf.*, Melbourne, Australia, Dec. 2006, pp. 404–423.

[9] S. Hagen and A. Kemper, "Model-based Planning for State-related Changes to Infrastructure and Software as a Service Instances in Large Data Centers," in *Proc. IEEE Int. Conf. on Cloud Computing (CLOUD'10)*, Miami, USA, Jul. 2010.

[10] S. Hagen and A. Kemper, "A Performance and Usability Comparison of Automated Planners for IT Change Planning," in *Proc. of 7th IEEE/IFIP Int. Conference on Network and Services Management (CNSM'2011)*, Paris, France, Oct. 2011.

[11] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN Planning System," *Journal of Artificial Intelligence Research (JAIR)*, vol. 20, pp. 379–404, 2003.

[12] W. L. da Costa Cordeiro, G. S. Machado, F. G. Andreis, A. D. Santos, C. B. Both, L. P. Gaspary, L. Z. Granville, C. Bartolini, and D. Trastour, "A Runtime Constraint-Aware Solution for Automated Refinement of IT Change Plans," in *Proc. IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM'08)*, Samos Island, Greece, Sep. 2008, pp. 69–82.

[13] S. Hagen and A. Kemper, "Facing the Unpredictable: Automated Adaption of IT Change Plans for Unpredictable Management Domains," in *Proc. of 6th IEEE/IFIP Int. Conference on Network and Services Management (CNSM'2010)*, Niagara Falls, Canada, Oct. 2010.

[14] R. Rebouças, J. Sauvé, A. Moura, C. Bartolini, and D. Trastour, "A decision support tool to optimize scheduling of IT changes," in *Proc. of 10th IFIP/IEEE Int. Symposium on Integrated Network Management (IM'2007)*, Munich, Germany, May 2007, pp. 343–352.

[15] C. Bartolini, C. Stefanelli, and M. Tortonesi, "SYMIAN: Analysis and performance improvement of the IT incident management process," *IEEE Transactions on Network and Service Management*, vol. 7, pp. 132–144, 2010.

[16] Yixin Diao and A. Heching, "Staffing optimization in complex service delivery systems," in *Proc. of 7th IEEE/IFIP Int. Conference on Network and Services Management (CNSM'2011)*, Paris, France, Oct. 2011.

[17] J. Sauvé, R. Santos, R. Rebouças, A. Moura, and C. Bartolini, "Change Priority Determination in IT Service Management Based on Risk Exposure," *IEEE Transactions on Network and Service Management*, vol. 5, pp. 178–187, 2008.

[18] N. Loutas, E. Kamateri, and K. Tarabanis, "A Semantic Interoperability Framework for Cloud Platform as a Service," in *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom 2011)*, Athens, Greece, Nov. 2011, pp. 280–287.

[19] T. Somasundaram, K. Govindarajan, M. Rajagopalan, and S. Rao, "An architectural framework to solve the interoperability issue between private clouds using semantic technology," in *2012 International Conference on Recent Trends In Information Technology (ICRTIT 2012)*, Chennai, Tamil Nadu, India, Apr. 2012, p. 162.

[20] F. Baader and J. H. Siekmann, in *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Eds. Oxford, UK: Oxford University Press, 1994, ch. Unification theory, pp. 41–125.

[21] J.-P. Jouannaud and C. Kirchner, in *Computational Logic: Essays in Honor of A. Robinson*, J.-L. Lassez and G. Plotkin, Eds. Cambridge, MA, USA: MIT Press, 1991, ch. Solving equations in abstract algebras: A rule-based survey of unification.

[22] N. J. Nilsson, *Principles of Artificial Intelligence*. Springer, 1982.

[23] M. Kolon, "Intelligent logical router service," online, 2004. [Online]. Available: http://support.neoteris.com/solutions/literature/white_papers/200097.pdf