A Performance and Usability Comparison of Automated Planners for IT Change Planning

Sebastian Hagen
Technische Universität München
Department of Computer Science
85748 Garching, Germany
hagen@in.tum.de

Alfons Kemper
Technische Universität München
Department of Computer Science
85748 Garching, Germany
kemper@in.tum.de

***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 satisfy business goals and to minimize costly disruptions on the business. As part of Change Management, IT changes need to be planned for. Despite previous efforts to use planning algorithms to generate IT change plans, it remains unknown as to which automated planning algorithm does best solve the IT change planning problem. To answer this question, we compare four domain independent automated planners in the context of an Infrastructure as a Service deployment case study of a three-tier business application. We focus on two aspects: (1) The scalability of the planners to large Configuration Management Databases (CMDBs) comprising thousands of resources and (2) their usability by a change manager, in particular how easily a change manager can specify the planning domain and rules to guide the search. For the deployment case study we conclude that Hierarchical Task Network algorithms scale to significantly larger CMDBs than all other examined algorithms. Furthermore, we find that their way to formalize search control naturally matches to the domain of IT change planning compared to that of others who require change managers to have a profound knowledge of the planning algorithm or temporal logic.**

I. Introduction

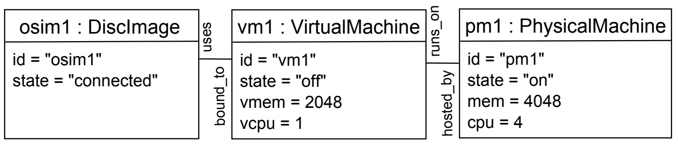
The Information *Technology Infrastructure Library* (ITIL) [1], a set of best practices on how to manage IT services and operations within a company, has become an important recommendation for the implementation of proper IT Service Management (ITSM). *Change Management* [2], part of ITIL, ensures that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

IT Change Planning, an important step of the process, is concerned with the generation of detailed IT change plans to accomplish high-level *Request for Changes* (RFCs) [2]. It has been subject to intensive research [3], [4], [5], [6], [7], [8], [9]. Some works [3], [5], [6], [9] on this topic propose planning approaches that are not based on algorithms previously discussed in the automated planning community [10]. Others [4], [7], [8] use already existing domain independent planning algorithms (sometimes with slight modifications) and apply them to the IT change planning problem. Despite these works it remains unknown as to how existing planning algorithms compare to each other when applied to IT change planning. In particular, the following two questions remain unanswered:

- **How well do the different algorithms scale with the number of Configuration Items (virtual machines (VMs), physical machines (PMs), disc images) in the Configuration Management Database to plan on?**
- **How difficult is it for a change manager to specify the IT changes and the planner's search control to work efficiently for IT changes? Which way to specify search control most naturally matches to the domain of IT change planning?**

To investigate which planners are most suitable for the IT change planning problem in respect to scalability and usability, we compare four different domain independent automated planners [11], [12], [13], [14] in a case study. The case study asks each planner to generate a high-level deployment plan of a three-tier application using an Infrastructure as a Service (IaaS) Cloud. We examine the planners scalability when it comes to plan on large *Configuration Management Databases* (CMDBs) comprising thousand of *Configuration Items* (CIs) under different resource utilizations (RUs). In addition to that, we discuss how well the different planners can be used by a logic and algorithm-agnostic IT change manager, in particular when it comes to specify search control knowledge to guide each planner. To the best of our knowledge this is the first work to examine different AI planning algorithms [10] in respect to their applicability to IT change planning.

We find that Hierarchical Task Network (HTN) planners are the ones to offer the best performance as they scale to much larger CMDB sizes (up to 20,000 CIs) compared to the other algorithms ($\leq$ 1000 CIs) in our case study. This extends previous works [7], [8] that only argue in favor of Hierarchical Task Network planning due to its natural fit to change planning but not show its performance benefits on large CMDBs. Similar to previous works [7], [8], we claim that HTN algorithms naturally match to IT change planning. However, this is the



```
(image osim1)              (vm vm1 2048 1)      (pm pm1 4048 4)
(state osim1 connected) (state vm1 off)         (state pm1 on)
        (bound-to osim1 vm1)     (runs-on vm1 pm1)
```

Fig. 1. Object-oriented model instance of a Configuration Management Database and isomorphic set of ground atoms used by automated planners.

```
1  (:action start-vm
2   :parameters  (?vmid ?pmid)
3   :precondition  (and
4    (vm ?vmid ?vmem ?vcpu) (state ?vmid off)           atoms of
5    (runs-on ?vmid ?pmid) (state ?pmid on)             precondition
6    (bound-to ?osid ?vmid) (state ?osid connected)
7   )
8   :effect (and                                         atoms added
9    (not (state ?vmid off)) (state ?vmid on)            and deleted
10  )
11 )
```

Fig. 2. Description of an IT-change / planning action to start a virtual machine written in the *Planning Domain Definition Language* (PDDL).

first work to support this claim by comparing alternatives to specify search control (among them, *linear temporal logic* (LTL) [13] and algorithmic specific control rules [12]) in a change planning case study with the goal to gain experience about their usability for change planning.

The remainder of this work is organized as follows: Section II sets the basic terminology and introduces the case study. Section III evaluates planning through planning graph analysis followed by an analysis of a forward chaining planner using temporal control knowledge in Section IV. Section V addresses planning through means-end analysis. Hierarchical Task Network Planning is evaluated in Section VI. We discuss related work in Section VII. Finally, Section VIII summarizes our findings and concludes the work.

II. Automated Planning and Case Study

A. Automated Planning

1) The problem of plan generation: Automated planning [10] has been an active research area within the *Artificial Intelligence* community for decades. The problem of plan generation (adapted to the IT change planning domain) is as follows: Given the current state of a data center (i.e., the current state of the infrastructure and software described in the CMDB), a description of the goal to achieve, a description of the preconditions and effects of IT changes[1] / planning actions, find a set of IT changes and a partial (sometimes total) order among the IT changes, that, when executed in any topological order of the partial order, the IT changes achieve the goal. We use actions and IT changes synonymously, depending on whether we want to highlight the IT change planning view or the view of the planner which uses actions to describe IT changes.

[1]If not noted otherwise, we consider an IT change to be one elementary, indivisible change operation on CIs of the CMDB.

Among other criteria, three different types of planners can be distinguished: (1) *Domain-specific planners* are made, tuned, and tested for a specific domain and constraints. They do not perform well or at all in a domain they were not engineered for. The previous works on change planning are domain-specific planners. (2) *Domain-independent planners* can be used in any planning domain. They take a definition of the preconditions and effects of actions for planning. Thus, if actions describe IT changes, domain independent planners can be applied to IT change planning. (3) *Configurable planners* use a domain independent planning engine (as in (2)) and additional domain specific search control which needs to be specified by a change manager to control the search. All planners evaluated in this work fall into category (2) and except Graphlan (see Section III) they allow the change manager to specify search control in one way or the other (category (2)). However, the planners differ significantly on how easily search control can be written by a change manager.

2) Transformation of object-oriented CMDBs: Traditionally object-oriented (OO) models have been used to describe the current state of infrastructure and software hosted in a data center, e.g., in commercial CMDBs [15] or in the *Common Information Model* (CIM) [16]. Automated planners [10] do not plan on object-oriented models, but the state of the world is described in a function-free first order language. Such a language comprises the following concepts: (1) *Constant symbols*: Describe objects of the planning domain. For example, CIs (`vm1`,...,`vm10`) or their states (`on`, `off`, `installed`) (2) *Predicate symbols*: For example `state`. (3) *Atom*: A predicate symbol together with variables. For instance, let `?ci` and `?st` be variables, then `(state ?ci ?st)` is an atom. An atom is called *ground* if all its variables are bound to constants/objects. For example, `(state vm1 off)` is a ground instance of the `state` atom. It describes the fact that virtual machine `vm1` is in state `off`. To be leveraged by automated planners, the information stored in an OO CMDB needs to be translated to a set of ground atoms. Figure 1 depicts a part of the object-oriented CMDB and the analogous ground atoms view used by all planners. To transform the OO CMDB, the attributes of an object are described as variables of a ground atom. For example, ground atoms `(vm vm1 2048 1)` and `(state vm1 off)` describe all attributes of object vm_1 in Fig. 1. References are translated to binary predicates (e.g., `(runs-on vm1 pm1)`). Maghraoui et al. [4] describe this transformation process in more detail.

3) Actions and IT changes:: An action is an atom together with a precondition and effects. Precondition and effects are described as a set of partially ground atoms. To apply an action all of its unbound variables need to be bound such that all atoms in the precondition become grounded and account in the current state of the *knowledge base* (KB). At each point in time there might be multiple consistent bindings among all variables of an action to instantiate it. For example, consider IT-change `start-vm` in Fig. 2. The action has two variables `?vmid` and `?pmid` that need to be bound in order to apply the action (Line 2). If we choose `?vmid:=vm1`, `?pmid:=pm1`, and

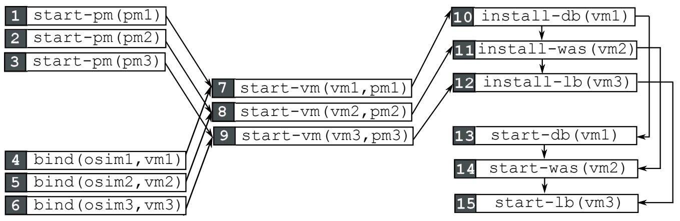


Fig. 3. Ideal partial order plan solving the planning case study.

`?osid:=osim1`, then `start-vm(vm1,pm1)` is an applicable action in the state depicted in Fig. 1 because all its ground precondition atoms in Lines 4-6 account in this state under the substitution. The effects of the action are the deletion of ground atom `(state vm1 off)` from the KB and the addition of `(state vm1 on)` to the KB (see Line 9).

B. Planning Case Study

1) Planning case study: The planning problem to solve by all planners is the generation[2] of a high-level deployment plan of a 3-tier business application (database (DB), web application server (WAS), and load balancer (LB)) making use of an IaaS Cloud. Figure 3 depicts a partial order plan that solves this problem on a particular instance of a CMDB. It consists of IT changes to turn on PMs (cr_{1-3}) and VMs (cr_{7-9}), to bind OS images to the VMs (cr_{4-6}), and to install (cr_{10-12}) and start (cr_{13-15}) software. Although this plan omits more complex configuration details inherent to applications and networks, it already causes serious trouble for several planners when CMDBs become large. All planners need to take the following constraints into account: (1) Images can only be bound to one VM, (2) VMs need to run dedicated on PMs, (3) DB, WAS, and LB are meant to run in their own VMs, and (4) software depending on another software, e.g., WAS on DB, cannot be installed/started without the other one being already installed/started due to configuration and runtime constraints.

2) Shape of CMDB used for evaluation: The algorithms are evaluated on differently sized and shaped configurations of the CMDBs. A CMDB, i.e., the knowledge base over which planning is done, always consists to $\frac{1}{4}$-th of PMs, VMs, OS images, and services no matter how large it is. Services consist to $\frac{1}{3}$-rd of DB, WAS, and LB services. For example, a CMDB containing 8,000 CIs comprises 2,000 PMs, VMs, OS images, and each 666 instances of DB, WAS and LB services. The goal to achieve is to deploy the three-tier application (see Fig. 3) in this environment.
During the experiments the resource utilization of CIs in the CMDB is varied. *Resource utilization* (RU) describes the fraction of resources (PMs, VMs, OS images, services) that cannot be used to properly instantiate an IT change that assigns resources. For example, if 100 out of 300 OS images have already been bound to a VM, then OS images have a RU of 33% because only 66% of all OS images qualify for an IT change to bind an unused OS image to a VM. Thus, RU is a metric describing the number of CIs that do not satisfy the precondition of IT changes that have to choose among resources (e.g., placement of PMs on VMs, placement of services on VMs, choice of OS images for VMs, etc.) because resources are already in use.

3) Reference characteristics of case study: Our case study is representative for IT changes because it comprises the following cases, typically observable in the workload of IT changes: (1) The need of IT changes to choose resources from large CMDBs to be properly instantiated, e.g., placement IT changes. This frequently happens for IT changes participating in the deployment of an application. (2) State related changes among dependent CIs [9], [7], e.g., the need to start a DB server to start a WAS server. State-related dependencies are typical for three tier applications when it comes to deployment, undeployment, and migration. (3) The refinement of abstract IT changes into more detailed IT changes [7], [17], [8]. As we will discuss later, some algorithms better cope with the potentially exponential increase in the search space that is caused by an increase in the number of actions that can be instantiated for a given CMDB configuration. It is due to this techniques, that we can expect our results to hold for a similar workload of IT changes that is dominated by IT changes that need to select resources over large CMDBs.

III. Planning Through Planning Graphs

A. The Graphlan Algorithm

The *Graphlan* algorithm [11] is a forward chaining search algorithm, i.e., it starts by searching from the initial state towards the goal state by applying actions. Graphlan takes as input a description of each action/IT change to consider for planning (see Fig. 2 for an example), a list of ground atom instances accounting in the initial state, and a list of ground atom instances that need to account in the goal state. Graphlan creates a planning graph which consists of several levels, i.e., disjunct sets of ground atoms (proposition level) and ground actions (action levels). Generally, proposition level i contains all the ground atoms that can be achieved by applying i or less IT changes to the initial state. Thus, proposition level 0 comprises all atoms of the initial state. To generate action level i, Graphlan creates all possible ground instances of all actions using all ground atoms in proposition level i. Finally, proposition level $i+1$ is created from proposition level i and action level i by adding all atoms of proposition level i to the $(i+1)^{st}$ proposition level (to account for doing nothing in the i^{th} step/action level) together with all positive and negated ground atoms mentioned in the effects of all actions in the i^{th} action level (to account for applying an IT change). Graphlan keeps on creating proposition and action levels until it reaches a proposition level that contains all atoms of the goal formula. It then starts a backward search to verify whether a valid plan exists. the backward search picks conflict-free actions from different action levels to form a plan. Actions from the same action level can be executed in parallel. For a more detailed introduction to the Graphlan planning algorithm see [11].

[2]All performance measurements are conducted on an *Intel Xeon* x86 CPU with 2.8Ghz and a maximum of 1024 MB RAM available to all planners. Planners run single threaded.

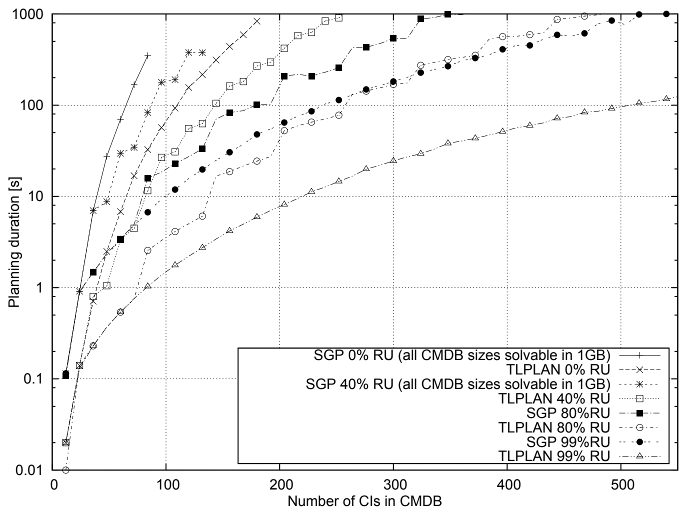


Fig. 4. Planning duration of Sensory Graphlan (SGP) and TLPlan (with LTL search control) depending on the size of the CMDB and the resource utilization.

B. *Evaluation*

1) Domain Description: With only 200 *lines of code* (LOCs) we found Graphlan's action descriptions to be easily writable because the change manager only needs to think about the preconditions and effects of single IT changes as shown in Fig. 2. Furthermore, we found the idea of atoms, which basically are tuples being added and deleted from a tuple store, to describe the current state of the world, intuitive. However, a change manager is likely to feel different about this logical representation. We do not consider this to be a big hurdle in the adoption of automated planning approaches because the object-oriented and predicate based representation are isomorphic which enables Domain Specific Languages for IT changes [7] which are closer to the change manager's domain.

2) Performance: Figure 4 depicts the planning time of *Sensory Graphlan* (SGP) [18], a LISP extension of Blum's original algorithm [11], for the planning case study. SGP is preferred over Blum's original implementation because in our experiments it solved larger problems while being memory constrained. The planning time increases exponentially with the size of the CMDB. For example, for an upper bound of 100s on the planning duration and a RU of 0%, a domain cannot be larger than 60 CIs, i.e., 15 PMs, VMs, OS images, and services to remain solvable (see Fig. 4). For a domain consisting of more than 84 CIs and 0% RU SGP runs out of memory. However, if resource utilization increases to 99%, a planning domain comprising 240 CIs becomes feasible within 100s. We conclude that the higher the RU and the smaller the CMDB, the faster SGP becomes and the more likely it is that SGP can solve the problem when constrained to 1024 MB of memory. All in all, the Graphlan algorithm shows an unsatisfactory performance for our case study and can at best be used to solve small problem instances. This is the case because the algorithm tries to apply all instances of all IT changes, independently of their contribution to the goal, in a breadth-first manner at each action level. With increased CMDB size and small RUs a larger number of CIs qualify as valid bindings to instantiate the actions inflating the action and proposition levels. Increased RUs and decreased CMDB sizes lead to less ways to instantiate an action and can make the difference between solvable and unsolvable problems.

IV. Planning Through Forward Chaining and Temporal Control Knowledge

A. *The TLPlan Algorithm*

Similar to Graphlan, *TLPlan* [13] is a forward chaining planner but it uses domain specific search control knowledge specified in a first-order version of linear temporal logic to prune plans causing undesired sequences of intermediate configurations of the CMDB. Algorithmically, TLPlan starts with the initial CMDB and determines all applicable ground instances of actions/IT changes that can be applied to the current CMDB. Based on these actions all successor CMDBs are created. TLPlan then continues in a depth-first search with the exploration of a successor CMDB. However, TLPlan does only further explore a CMDB configuration if the last recently added IT change to the plan does not violate the temporal logic control formula. Nevertheless, all successor worlds are created. TLPlan also offers the option to directly discard CMDB states violating the formula (leading to less memory consumption) but we were unable, due to stability issues of the planner, to derive a working LTL formula for this early pruning option.

1) Domain Description: For TLPlan to solve the IT change planning case-study, the change manager has to specify a formula in linear temporal logic to guide the search. Let's assume cr_i are IT changes, such that the sequence of IT changes $< cr_1, cr_2, ..., cr_n >$ is a plan to satisfy the deployment of a 3-tier application. Then, let $< s_0, ..., s_n >$ be the sequence of corresponding configurations s_i of the CMDB induced by the execution of the plan (s_0 initial state, s_n valid goal state, s_i state after the execution of cr_i, $i \in \{1, ..., n\}$). A first-order LTL formula, evaluated over this sequence of intermediate CMDB states, can be specified by the change manager to prevent undesired plans. Two out of the four LTL operators [13] used in our control rules are: (1) $\Box f$ means that f holds for the current and all future configurations (states) of the CMDB. (2) $\bigcirc f$ means, that f holds in the next state of the CMDB. To focus TLPlan's forward chaining planning algorithm, IT changes that do not progress toward the goal state need to be disallowed by a LTL formula. For example, it makes sense to only turn on VMs (action `start-vm`, Fig. 2) that are meant to be on in the goal state. As a LTL formula:

```
(□  (∀ ?vmid : (vm ?vmid ?mem ?cpu)
        (  (state ?vmid off) ∧
           (○ (state ?vmid on))
        ) → (goal (state ?vmid ?on))
)    )
```

The formula only permits sequences of intermediate CMDB configurations ($\Box$) such that among subsequent configurations ($\bigcirc$) only VMs are allowed to change their state from *off* to *on* (left side of implication) if the VM is explicitly specified as *on* in the goal state (goal expression). There are several problems with LTL formulas as a mean to formalize effective search control for IT changes: (1) It cannot be decided any more whether an IT change contributes to the goal state if the goal state is partially specified. (2) The LTL formula used in the change planning case study comprises 170 LOCs – too large to be practically used and derived by an IT change manager. (3) It is very unlikely that a change manager is willing and capable to cope with LTL and its subtle semantics. (4) We found it very difficult to write LTL formulas over a sequence of intermediate CMDB states. Instead, given our experience, it seems more natural to specify LTL formulas over valid and invalid sequences of IT changes because intermediate sates are more difficult to grasp than the constraints on sequences of IT changes.

2) Performance: Without LTL search control and a RU of 0%, TLPlan takes 10ms to solve the change planning case study for 12 CIs and 833s for a CMDB comprising 24 CIs. The problem becomes unsolvable within 12h for larger CMDBs and 1024 MB of RAM. Thus, TLPlan is impractible to solve even small problems without LTL search control. Figure 4 depicts the performance of TLPlan (with LTL search control) in comparison to SGP/Graphlan depending on the size of the CMDB and the resource utilization. TLPlan performs better than SGP for every resource utilization and every size of the CMDB . While SGP can only solve very small instances of the case study when constrained to 1 GB of main memory at RUs betweeen 0 and 60%, TLPlan does not show this limitation. Similar to SGP, TLPlan struggles to maintain performance with larger CMDBs. The time to derive a plan increases exponentially with the size of the CMDB and is slightly subdued by higher resource utilizations because less successor CMDBs need to be managed by the planner. Different to Graphlan, TLPlan produces totally ordered plans instead of partially ordered plans. Note, that the performance improvement is achieved through complex LTL formulas that are difficult to specify.

V. Planning through means-end analysis

A. The Prodigy Algorithm

Different to Graphlan's and TLPlan's forward chaining approach, *Prodigy* [12] searches backwards from the goal (backward chaining). Prodigy uses means-end analysis, i.e., it chooses a ground atom of the goal that has not yet been achieved and attempts to instantiate an action in such a way that its effects produce that atom. The atoms in the action's precondition that do not yet account become goal atoms that need to be subsequently achieved by applying another action. For example, consider the ground goal atom `(state vm1 on)`. Action `start-vm(?vmid, ?pmid)` in Fig. 2 can be partially instantiated (`?vmid:=vm1`) such that it produces this atom. If atoms in the precondition of the action should not yet account, Prodigy keeps on working on these open goals. During planning Prodigy has several choices: (1) Which goal to achieve first, (2) which action to use to achieve the goal and how to instantiate the action, e.g., how to instantiate variable `?pmid` in `start-vm(vm1, ?pmid)`, and (3) when to apply an action. To make the right choices, an IT change manager can specify *control rules* which tell the planner on how to make these decisions.

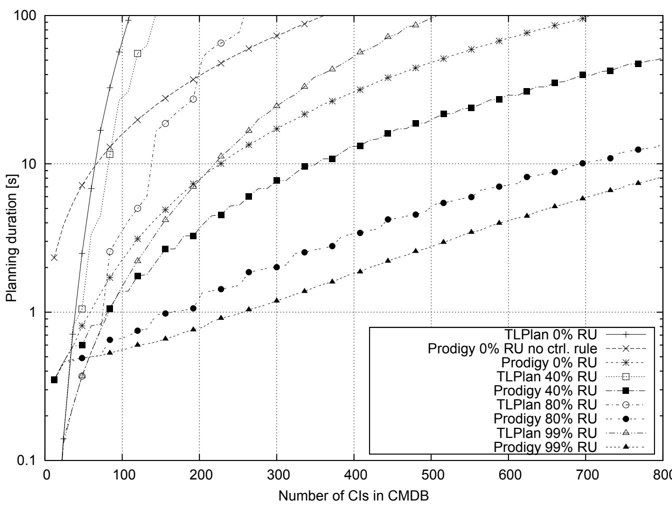


Fig. 5. Planning duration of Prodigy (with ctrl. rules) depending on (1) size of the CMDB, (2) the resource utilization, and (3) the use of control rules compared with TLPlan (with LTL search control).

B. Evaluation

1) Domain Description: Prodigy takes the same domain description (besides syntactical differences) as SGP and TLPlan for input. Thus, Prodigy's actions can be as easily engineered as Graphlan's and TLPlan's. For Prodigy to work efficiently, control rules need to be specified. Control rules are *if-then* rules. For example, one of four control rules used in the case study reads as follows: If the planner is working on goal `(state ?vmid on)` and tries to achieve it by instantiating action `start-vm(?vmid,?pmid)` (consult Fig. 2 to see why this achieves the goal), then bind `?pmid` to a PM, that has not a VM running on it. The reason for this rule is that in a subsequent action this constraint is checked leading to costly backtracking if the wrong decision is made when instantiating action `start-vm(?vmid,?pmid)` with the wrong PM. For change managers to write effective control rules, they are forced to look at the debug output and need to have a precise understanding of how the algorithm works. With search control so closely related to the algorithm, i.e., the need to examine the debug output of plans, we doubt that this approach is of practical use to a change manager. However, if actions, problems, and control rules have been carefully tuned, Prodigy's control rules offer effective speedup.

2) Performance: Figure 5 depicts the planning performance of Prodigy and TLPlan (with LTL search control) depending on the size of the CMDB, the use of control rules, and the resource utilization. Prodigy performs better with control rules tuned to the planning case study. For example, with time set to 100s and 0% RU, Prodigy can solve the case study on a CMDB comprising 350 CIs without control rules compared to 700 CIs with control rules. For small CMDB sizes, TLPlan outperfroms Prodigy even with control rules. Beyond 24 - 36 CIs (depending on the RU), Prodigy becomes faster than TLPlan. Thus, for realistic CMDB sizes Prodigy always outperforms TLPlan and SGP/Graphlan. Without control rules TLPlan manages to stay ahead of Prodigy on slightly larger CMDBs. For example, at 0% RU the CMDB must be larger than 60 CIs for Prodigy (without ctrl. rules) to outperform TLPlan.

Similar to TLPlan and SGP, the higher the resource utilization, the faster the planner. For every not yet achieved goal atom that Prodigy decides to work on, it computes all ground instances of all actions that can achieve this atom before it continues planning with one instance. Thus, the larger the CMDB and the less the resource utilization, the more CIs qualify to instantiate an atom, increasing the computational effort to compute all bindings for applicable IT changes. Even without control rules Prodigy performs significantly better than TLPlan for the deployment case study (350 CIs vs. 100 CIs within 100s at 0% RU). Prodigy only considers the application of actions that contribute to the goal. Differently, Graphlan and TLPlan apply all applicable actions in each step. TLPlan then prunes parts of the newly generated worlds using LTL search control. Both is costly and degrades the performance.

Note, that these results differ to Blum et al. [11] who showed that Graphlan outperforms Prodigy in two artificial planning domains (2-Rockets domain and Link-repeat domain). However, for the IT change planning problem Prodigy wins because when the CMDB increases linear in size, it, though it has to potentially compute exponentially many bindings, does not materialize all these possible future states. Instead, the proper one is chosen and the control rules can prevent backtracking.

VI. Hierarchical Task Network Planning

A. The SHOP2 Algorithm

Hierarchical Task Network (HTN) planners differ from the previously examined planners in a significant way: The goal to achieve is not specified as a set of ground atoms that need to be satisfied by the goal state, but as an abstract task, e.g., to deploy a three-tier application, for which the planner needs to derive a plan. Similar to the previous approaches, elementary IT changes are described as actions that add and delete ground atoms to and from the knowledge base (see Fig. 2 for an example). The domain specific search control is described by HTN *methods*. *Methods* decompose an abstract task, i.e., a high-level IT change, into finer grained IT changes until IT changes are reached that can be directly mapped to actions. Fig. 6 depicts an exemplary decomposition tree created by an HTN planner for the abstract IT change to deploy a database on a specific CMDB configuration. To generate an IT change plan that solves the high-level change `deploy-db(db1)`, SHOP2 [14], a forward chaining partial order HTN planner, searches for a method to decompose the abstract IT change. A method specified by a change manager instructs the planner to decompose cr_1 by trying to achieve the IT changes `provision-vm(vm1)` (cr_2), `install-db(vm1)` (cr_6), and `start-db(vm1)` (cr_7) in sequence (see Fig. 6).

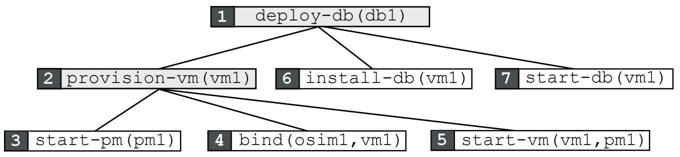


Fig. 6. Simplified HTN decomposition tree created by an HTN algorithm for the deployment of a database.

Methods capture best practice problem solving strategies inherent to the domain. Planning is done in a first depth search according to the order on the subtasks (see numbers of IT changes in Fig. 6). Note, that planning terminates at the leaf nodes of the decomposition tree, which cannot be further decomposed and directly map to the actions also used by the other algorithms. An HTN planner only considers to apply an action if it is directed to do so by the decomposition tree. For example, the planner will never try to start a VM (cr_5) before an image has been bound (cr_4) because the decomposition tree directs the planner to do it in the opposite order. The plan returned by an HTN planner only comprises the leaf nodes of the decomposition tree. Note, that the leaf nodes match to the changes in Fig. 3 and their order as explored in the depth first search is a topological order of the partial order of the plan in Fig. 3.

B. Evaluation

1) Domain Description: The SHOP2 domain description comprises around 700 LOCs (3.5-times the size of the other approaches). In addition to the specification of the preconditions and effects of IT changes (as needed for all other approaches as well, see Fig. 2), refinement rules need to be specified to tell the planner the order in which to explore IT changes during search. Thus, it takes more effort to write and debug such a domain. Once engineered, an HTN domain has several advantages: (1) It is very reusable because newly added changes can refer to already existing changes. For example, an IT change to deploy a three-tier architecture can rely on the change to deploy a database (see Fig. 6). (2) The idea of abstract task refinement matches precisely to the IT change planning problem, that, given an abstract Request for Change [2] asks the change manager to derive a plan to implement that change. The change manager stays within his/her domain of thinking (refinement of IT changes) without being distracted by LTL formulas (see Par. IV-A1) or algorithmic specific control rules (see Par. V-B1). All in all, HTN methods, i.e., the refinement of IT changes, seem to be the easiest way for a change manager to describe domain specific search control knowledge. These findings are consistent with Cordeiro et al. [6] who argue in favor of reusable plan templates that can be implemented using HTN methods. While the proximity of IT change planning to HTN planning has been noted before [7], [8] we argue in this work that alternatives such as LTL and algorithmic control rules are not a better solution to specify search control.

2) Performance: Figure 7 depicts the planning duration of SHOP2 for two different domain descriptions, one that

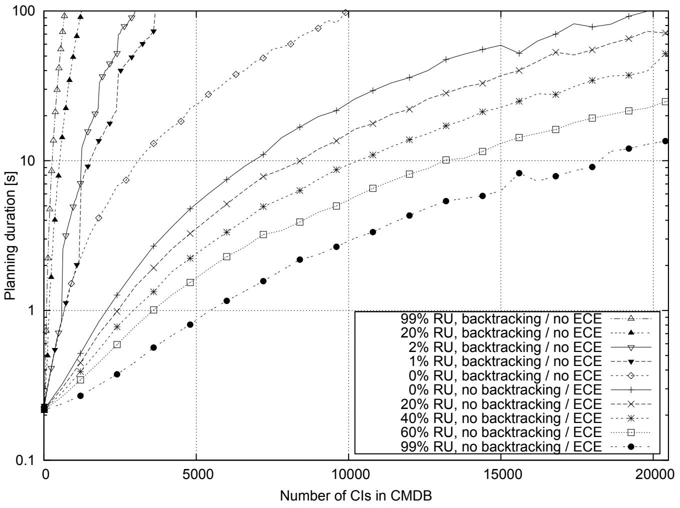


Fig. 7. Planning duration of SHOP2 depending on the size of the CMDB, the resource utilization, and the type of the domain description.

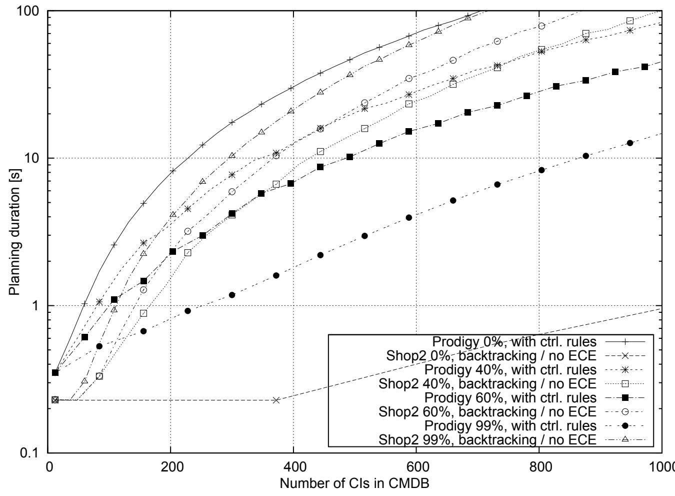


Fig. 8. Planning duration of SHOP2 (no ECE domain) and Prodigy (with ctrl. rules)

avoids backtracking by means of *early constraint enforcement* (ECE) in the decomposition tree and another one that only enforces constraints in the leaf nodes. To see the difference consider `bind(osim1,vm1)` (cr_4) in Fig. 6. When planning for the case study problem, a method is applied to decompose `provision-vm(vm1)` (cr_2) into cr_3, cr_4, and cr_5. This means, that appropriate OS images need to be determined to instantiate cr_4. For cr_4 to be executable, its precondition checks whether the image is unbound. If the method binds parameter `?osid` in cr_4 to already bound images, then the planner will backtrack over cr_4 because it is infeasible (backtracking / no ECE in Fig. 7). Instead, we can add the unbound image constraint to the method to avoid backtracking because then the method only choose images that do not lead to backtracking. This yields the ECE / no backtracking measurements in Fig. 7.

Similar to all other algorithms, the planning time increases exponentially with the size of the CMDB. However, the exponential increase is much more modest than for the other planners if early constraint enforcement is used. Thus, problems of up to 20,000 CIs − 28 times the size limit of Prodigy and 333 times that of SGP − can be solved in 100s at 0% RU (worst case). The performance is superior because on decomposing a change, SHOP2 computes all bindings of variables in the subtasks but only applies one instance of an action or method during planning. Thus, SHOP2 does not blow up its search space by applying all unifications of an IT change as Graphlan and TLPlan do. In addition to that, the decomposition tree tells the planner precisely when to try an IT change pruning large portions of the search space.

Using ECE, the runtime decreases with increased RU (similar to all other planners). In this case the runtime is dominated by computing all ground instances of actions and methods which is lower for higher RUs.

If backtracking is not avoided, planning takes significantly longer than with ECE because backtracking is more costly than enforcing binding constraints early. In this case, the runtime increases with the resource utilization (exactly opposite to the ECE case) because the higher the RU, the more frequent the planner has to backtrack as it more often commits to unsuitable CIs to instantiate an action or method. This yields the question whether a Prodigy domain with control rules outperforms a quickly written HTN domain not avoiding backtracking by means of ECE?

Yes, but only on large CMDBs. On small CMDBs a naively written SHOP2 domain (not avoiding backtracking) performs better than a carefully tuned Prodigy domain with control rules for the case study (see Fig. 8). However, with increasing CMDB size Prodigy starts to outperform SHOP2 without ECE. This happens the earlier the higher the resource utilization grows (for 90% RU from 90 CIs onwards, for 60% RU from 200 CIs onwards, for 40% RU from 750 CIs onwards, see Fig. 8) because the lower the RU, the less SHOP2 has to backtrack making it more difficult for Prodigy to catch up. For 0% RU, i.e., no backtracking for SHOP2, Prodigy cannot catch up with SHOP2 any more.

VII. Related Work

Several solutions [3], [4], [5], [6], [7], [8], [9], [17] have been proposed for the (semi-)automated generation of IT change plans. All works propose domain specific algorithms for IT change planning and do not examine them in the context of already existing automated planning algorithms [10].
Keller et al. [3] propose *CHAMPS*, a system for the planning and scheduling of IT changes which enables a high degree of parallelism among IT changes. The authors propose a planning algorithm which is not based on automated planning algorithms [10]. A performance and usability comparison to other planning algorithms is out of the scope of that paper. Maghraoui et al. [4] are the first to apply an automated planner, UCPOP, to generate IT change plans. Their highly customized version of the UCPOP algorithm is applied to a case study in the size of a few hundred resources. Our experiences with UCPOP for our case study are that it is even slower than Graphlan without control knowledge. Furthermore, control

knowledge is extremely difficult to specify because it is too close to the algorithm. A comparison to other algorithms than UCPOP is out of the scope of Maghraoui's work. Cordeiro et al. [6], [17] propose *ChangeLedge* a system for the generation of change plans by capturing best practices in IT change design using change templates. Similar to CHAMPS [3], the proposed algorithm does not reason about the preconditions and effects of IT changes on a logical level. It remains unknown as to how the algorithm scales and how it compares to automated planners [10]. In another work, Cordeiro et al. [5] propose a runtime constraint aware solution for the automated refinement of IT changes. The proposed algorithm captures the basic idea of preconditions and effects of IT changes as known from automated planning [10]. It remains open, as to how their approach relates to automated planners in terms of performance and logical soundness. In a previous work [7] we propose to apply a hybrid HTN - state based planning algorithm to the IT change planning problem. Similarly, Trastour et al. [8] propose a pure HTN algorithm. Both works do not compare the proposed HTN solutions to other already existing domain independent planners in respect to performance and usability as done in this work. The work herein extends these works by providing evidence that HTN planning algorithms are the fastest and most usable algorithms for IT change planning.
Change plan generation is closely related to plan execution, which can be troublesome due to unpredictable failures. To proactively avoid failures when IT changes are executed, we present in [19] an approach to render IT change plans feasible again if the CMDB changes between planning and execution. The proposed solution is independent of the change planning algorithm and we found it to be applicable to smaller IT change plans. To avoid the adaptation of change plans and to detect conflicting IT changes, we propose in [20] an approach to proactively detect conflicting IT changes. These works extend the work herein, in that all examined planning algorithms cannot prevent conflicts among IT changes created by different operators or runs of a planner. Others argue in favor of risk assessment [21], [22], [23] for IT change plans to proactively treat risks during deployment. Similar to risk assessment, the logical soundness and completeness of the algorithms evaluated in this work contributes to the proactive treatment of problems because they guarantee the feasibility of the generated plan on a logical level. If proactive solutions should fail, Machado et al. [24], [25] propose a rollback solution to deal with failures during change implementation in a reactive way by undoing partially executed change plans.

VIII. Conclusions

So what is the best algorithm in terms of usability? In terms of applicability by a change manager, planners without any search control are easiest to use (see Graphlan, Par. III-B1) because only preconditions and effects of IT changes need to be specified. Unfortunately, runtimes become very bad in this case (Par. III-B2). Among the different ways to specify search control for IT changes, we found that algorithm specific control rules (see Prodigy, Par. V-B1) cannot be readily used by a change manager because profound algorithmic understanding and plan debugging is necessary. Although, TLPlan's approach using linear temporal logic is independent of algorithmic knowledge, it needs the change manager to undergo initial training in linear temporal logic. Furthermore, we found LTL formulas over state sequences to be a unnatural way to specify search control for IT changes (see Par. IV-A1). Instead, we think, that the refinement ideas inherent in HTN methods very well match to the refinement of Request for Changes [2] into change plans. Such refinement rules can be written without knowledge about the algorithm but nevertheless require additional specification effort compared to domains without search control.
And what is the best algorithm in terms of performance? We found that planners without search control, e.g., Graphlan and TLPlan, can only solve problem instances of a few CIs (Par. III-B2). TLPlan is slightly better, than Graphlan when search control is specified. Means-end analysis works better for the change planning case study (see Prodigy, Par. V-B2) than an undirected forward chaining planner (Graphlan) and a temporally controlled one (TLPlan) because it does not instantiate all applicable actions at once and prunes larger portions of the search space. HTN planning delivers the best performance for the case study because its rigorous refinement concept prunes large portions of the search space by considering the appropriate IT changes at the right time. A naively (algorithm agnostic) written HTN domain for the case study is faster than Prodigy for smaller CMDBs (the extend depends on the RU). However, for larger CMDBs Prodigy becomes better. Nevertheless, a carefully engineered HTN domain with early constraint enforcement still outperforms all other algorithms by a factor of 28 to 333 at 0% RU.

For automated planning approaches to emerge from research prototypes to commercial service management products, performance on large CMDBs and usability by a change manager are key factors of success. Given the results of the experiments in this work, we believe that HTN algorithms possess both characteristics.
But all that glistens is not gold. For any algorithm to scale to large CMDBs, the domain description and search control has to be carefully tuned. This is no show-stopper but a great opportunity for data center management products to distinguish themselves from each other: They could provide a carefully tuned set of IT changes that can be efficiently planned for on large CMDBs by means of an HTN planner. − Something that does not seem to be farfetched considering the results of our case study.

Acknowledgment

This work has been generously supported by a research grant of the *International Graduate School of Science and Engineering* (IGSSE) and the *Munich Center of Advanced Computing* (MAC) of Technische Universität München.

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 *et al.*, "The CHAMPS System: Change Management with Planning and Scheduling," in *Proc. of 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, Seoul, South Korea, Aug. 2004, pp. 395–408.

[4] K. E. Maghraoui, A. Meghranjani, T. Eilam, M. H. Kalantar *et al.*, "Model Driven Provisioning: Bridging the Gap Between Declarative Object Models and Procedural Provisioning Tools," in *Proc. of 7th ACM/IFIP/USENIX International Middleware Conference*, Melbourne, Australia, Dec. 2006, pp. 404–423.

[5] W. L. da Costa Cordeiro, G. S. Machado, F. G. Andreis, A. D. Santos *et al.*, "A Runtime Constraint-Aware Solution for Automated Refinement of IT Change Plans," in *Proc. of 19th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2008)*, Samos Island, Greece, Sep. 2008, pp. 69–82.

[6] W. L. da Costa Cordeiro, G. S. Machado *et al.*, "ChangeLedge: Change Design and Planning in Networked Systems based on Reuse of Knowledge and Automation," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 53, pp. 2782–2799, 2009.

[7] S. Hagen, N. Edwards, L. Wilcock, J. Kirschnick *et al.*, "One Is Not Enough: A Hybrid Approach for IT Change Planning," in *Proc. of 20th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2009)*, Venice, Italy, Oct. 2009, pp. 56–70.

[8] D. Trastour, R. Fink, and F. Liu, "ChangeRefinery: Assisted Refinement of High-Level IT Change Requests," in *Proc. of 10th IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY 2009)*, London, UK, Jul. 2009, pp. 68–75.

[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. of 3rd IEEE International Conference on Cloud Computing (CLOUD 2010)*, Miami, USA, Jul. 2010, pp. 11–18.

[10] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Morgen Kaufmann, 2004.

[11] A. Blum and M. Furst, "Fast Planning Through Planning Graph Analysis," *Artificial Intelligence*, vol. 90, pp. 281–300, 1997.

[12] M. Veloso, J. Carbonell, A. Perez, D. Borrajo *et al.*, "Integrating Planning and Learning: The PRODIGY Architecture," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 7, no. 1, pp. 81–120, 1995.

[13] F. Bacchus and F. Kabanza, "Using temporal logics to express search control knowledge for planning," *Artificial Intelligence*, vol. 116, pp. 123–191, 2000.

[14] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter *et al.*, "SHOP2: An HTN Planning System," *Journal of Artificial Intelligence Research (JAIR)*, vol. 20, pp. 379–404, 2003.

[15] A. Keller and S. Subramanian, "Best Practices for Deploying a CMDB in large-scale Environments," in *Proc. of 11th IFIP/IEEE International Symposium on Integrated Network Management (IM 2009)*, Long Island, New York, Jun. 2009, pp. 732–745.

[16] W. Bumpus, J. W. Sweitzer, P. Thompson, A. R. Westerinen *et al.*, *Common Information Model: Implementing the Object Model for Enterprise Management*. John Wiley & Sons, 2000.

[17] W. L. da Costa Cordeiro, G. Machado, F. Daitx, C. Both *et al.*, "A Template-based Solution to Support Knowledge Reuse in IT Change Design," in *Proc. of 11th IEEE/IFIP Network Operations and Management Symposium (NOMS 2008)*, Salvador, Bahia, Brazil, Apr. 2008, pp. 355–362.

[18] D. S. Weld, C. R. Anderson, and D. E. Smith, "Extending Graphlan to Handle Uncertainty and Sensing Actions," *Proceedings of AAAI*, pp. 897–904, 1998.

[19] S. Hagen and A. Kemper, "Facing the Unpredictable: Automated Adaption of IT Change Plans for Unpredictable Management Domains," in *Proc. of the 2010 International Conference on Network and Services Management (CNSM 2010)*, Niagara Falls, Canada, Oct. 2010.

[20] S. Hagen and A. Kemper, "Towards Solid IT Change Management: Automated Detection of Conflicting IT Change Plans," in *Proc. of 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*, Dublin, Ireland, May 2011, pp. 265–272.

[21] J. A. Wickboldt, L. A. Bianchin, R. C. Lunardi, F. G. Andreis *et al.*, "Improving IT Change Management Processes with Automated Risk Assessment," in *Proc. of 20th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2009)*, Venice, Italy, Oct. 2009, pp. 71–84.

[22] J. A. Wickboldt, G. S. Machado, W. L. da Costa Cordeiro, R. C. Lunardi *et al.*, "A Solution to Support Risk Analysis on IT Change Management," in *Proc. of 11th IFIP/IEEE International Symposium on Integrated Network Management (IM 2009)*, Long Island, New York, Jun. 2009, pp. 445–452.

[23] L. A. Bianchin, J. A. Wickboldt, L. Z. Granville, L. P. Gaspary *et al.*, "Similarity Metric for Risk Assessment in IT Change Plans," in *Proc. of the 2010 International Conference on Network and Services Management (CNSM 2010)*, Niagara Falls, Canada, Oct. 2010, pp. 25–32.

[24] G. S. Machado, F. F. Daitx, W. L. da Costa Cordeiro, C. B. Both *et al.*, "Enabling Rollback Support in IT Change Management Systems," in *Proc. of 11th IEEE/IFIP Network Operations and Management Symposium (NOMS 2008)*, Salvador, Bahia, Brazil, Apr. 2008, pp. 347–354.

[25] G. S. Mechado, W. L. da Costa Cordeiro, A. D. dos Santos, J. A. Wickboldt *et al.*, "Refined Failure Remediation for IT Change Management Systems," in *Proc. of 11th IFIP/IEEE International Symposium on Integrated Network Management (IM 2009)*, Long Island, New York, Jun. 2009, pp. 638–645.