Automatic Plan Generation and Adaptation by Observation

Supporting complex human planning

Hugo Ferreira

Advisor: Rui Carlos Camacho de Sousa Ferreira da Silva

Co-advisor: João José da Cunha e Silva Pinto Ferreira

A Thesis presented for the degree of
Doctor of Philosophy

Department of Informatics Engineering
Faculty of Engineering of the University of Porto
Portugal

June 2011

Dedicated to

My wife Paula, for here boundless love and unending patience.

Abstract

With the advent of automated management of wokflow processes, the need to support ever more flexible processes arises. Theses processes, also referred to as ad-hoc, adaptive, or intelligent processes, are difficult to manage. This is because they need to be formalized in order to be successfully automated and require continuous adaptation due to a constantly changing environment.

Many techniques have been employed in order to support such ad-hoc processes. The construction of processes via automated planning techniques, as used in Artificial Intelligence, is one of them. The problem with this approach, however, is the difficulty in identifying and designing the operators definitions that describe a planning domain. This thesis proposes a Machine Learning algorithm that acquires these operators definitions from the non-obstructive observation of execution traces of such ad-hoc processes. Unlike the related work, this machine learning algorithm extends the state-of-the art because it is not provided with information on either the number of operators that exist nor the operators' names and parameters.

We formalize the learning problem as an instance of the Minimum Consistent Subset Cover problem. A set of constraints are identified that determine which covering rules are generated that may represent operator definitions. These constraints, together with the use of a greedy covering algorithm, are used to determine the correct number of the operators and their definition when the data sets have no noise.

The basic algorithm depends only on the co-occurrence of covering literals. The basic algorithm was then extended to deal with overlap operators (operators that consist of combinations of co-occurring literals). Dealing with noisy data is a fundamental requirement for the practical application of any machine learning algorithm.

The proposed algorithm is able to deal with errors of omission and commission.

Experiments were performed using four planning domains. Three of these were obtained from an International Planning Competition. The fourth domain describes a workflow process that was specifically designed for testing the identification of overlap operators. Two main sets of experiments were made. The first used noiseless data only. The second used three different settings of noisy data. These noisy experiments used data sets that contained errors of commission only, errors of omission only and combination of these.

The results using noiseless data show that the algorithm is capable of identifying all of the operators without any errors, irrespective of whether or not overlap operators exist. When only errors of commission exist, all of the oracle's operators are also correctly identified. However a number of superfluous definitions are sometimes generated. Errors of omission have proven to be the most difficult to deal with. Whenever such errors are injected into the data sets, not all of the domain's operators can be correctly identified. As with the case of errors of commission, additional incorrect operator definitions may also be generated. Even though determining the correct solutions in the presence of noise is not possible, the algorithm is robust, producing results with low error rates.

Resumo

Com o advento da gestão automatizada de processos de wokflow, surge a necessidade de apoiar processos cada vez mais flexíveis. Estes processos, também conhecidos como processos ad-hoc, adaptáveis, ou inteligentes, são difíceis de gerir. Isso porque precisam de ser formalizados para que sejam automatizados com êxito e também exigem uma contínua adaptação devido ao meio que está em constante mudança.

Muitas técnicas têm sido empregues no apoio a processos ad-hoc. A construção dos processos através de técnicas de planeamento, como usado na Inteligência Artificial, é um deles. O problema com esta abordagem, porém, é a dificuldade em identificar e construir as definições de operadores que descrevem um domínio de planeamento. Esta tese propõe um algoritmo de Aprendizagem Computacional que adquire as definições de operadores a partir da observação dos registos da execução dos processos ad-hoc. Ao contário de outros trabalhos nesta área, este algoritmo de Aprendizagem Computacional estende o estado-da-arte porque não necessita de informação sobre o número de operadores que existem nem os nomes e parâmetros dos operadores.

O problema de aprendizagem foi formalizado como uma instância do problema de cobertura de subconjunto mínimo consistente. Um conjunto de restrições é identificado permitindo determinar quais as regras que representam as definições do operador. Essas restrições, juntamente com o uso de um algoritmo ganancioso de cobertura, são utilizados para determinar o número correcto dos operadores e suas definições, quando os conjuntos de dados não têm qualquer ruído.

O algoritmo básico depende somente da co-ocorrência de literais de cobertura. O algoritmo básico foi então estendido para lidar com operadores de sobreposição (operadores que consistem em combinações de literais em co-ocorrência). Lidar com

dados com ruído é um requisito fundamental para a aplicação prática de qualquer algoritmo de Aprendizado Computacional. O algoritmo proposto é capaz de lidar tanto com os erros de omissão como comissão.

Experiências foram realizadas utilizando quatro domínios de planeamento. Três destes foram obtidos de um Concurso Internacional de Planeamento. O quarto domínio descreve um processo de workflow, que foi concebido especificamente para testes de identificação dos operadores com sobreposição. Dois grandes grupos de experiências foram realizados. O primeiro grupo de dados não possuía ruído. O segundo grupo de dados continha três configurações diferentes de dados com ruído. Estas experiências com os dados com ruído utilizaram conjuntos de dados que continha só erros de comissão, só erros de omissão e uma combinação de ambos.

Os resultados utilizando dados sem ruído mostram que o algoritmo é capaz de identificar todos os operadores, sem qualquer erro, independentemente de existirem ou não operadores de sobreposição. Quando existem apenas os erros de comissão, todos os operadores do oráculo também são identificadas correctamente. No entanto, uma série de definições supérfluas são por vezes geradas. Erros de omissão mostraram ser os mais difíceis de tratar. Sempre que tais erros são introduzido nos dados, nem todos os operadores do domínio podem ser correctamente identificados. Tal como no caso de erros de comissão, definições adicionais incorrectas de operadores também podem ser geradas. Apesar de não ser possível determinar as soluções correctas na presença de ruído, o algoritmo é no entanto robusto, produzindo resultados com baixas taxas de erro.

Résumé

Avec l'apparition de la gestion automatisée des processus des flux de travail, la nécessité de supporter des processus toujours plus flexibles se fait sentir. Ces processus, que l'on appelle aussi processus ponctuels, adaptatifs ou intelligents, sont difficiles à gérer. Le raison en est qu'ils doivent être formalisés pour être automatisés avec succès et qu'ils exigent une adaptation continue à cause des changements constants de l'environnement.

De nombreuses techniques ont été employées pour supporter de tels processus ponctuels. La construction de processus via des techniques de planification automatique, comme celles qui sont utilisées en Intelligence Artificielle, est l'une d'entre elles. Toutefois, le problème de cette approche est la difficulté d'identifier et de concevoir les définitions d'opérateurs qui décrivent un domaine de planification. Cette thèse propose un algorithme d'apprentissage artificiel qui acquiert des définitions d'opérateur à partir de l'observation non obstructionniste des traces d'exécution de tels processus ponctuels. A la différence d'un travail similaire, cet algorithme d'apprentissage artificiel élargit la technologie de pointe parce qu'il n'est pas alimenté par une information basée sur l'un ou l'autre nombre d'opérateurs qui existent ni sur les noms et paramètres des opérateurs.

Nous formalisons le problème d'apprentissage comme un cas de problème de la Couverture Minimale d'un Sous-ensemble Cohérent. Une série de contraintes sont identifiées, elles déterminent quelles règles de couverture sont produites, qui peuvent représenter des définitions d'opérateur. Ces contraintes, jointes à l'utilisation d'un algorithme glouton de couverture, sont utilisées pour déterminer le nombre correct d'opérateurs et leur définition, quand la série de données n'a pas de corruption.

L'algorithme de base dépend seulement de la cooccurrence des littéraux de cou-

verture. L'algorithme de base était alors étendu pour faire face aux opérateurs de chevauchement (Opérateurs qui consistent en combinaisons de littéraux co-occurrents). Faire face aux données corrompues est une exigence fondamentale pour l'application pratique d'un algorithme d'apprentissage artificiel. L'algorithme proposé est capable de faire face aux erreurs d'omission et de commission.

Des expériences ont été exécutées en utilisant quatre domaines de planification. Trois de ceux-ci ont été obtenus à partir d'une Compétition Internationale de Planification. Le quatrième domaine décrit un processus de flux de travail qui a été spécifiquement conu pour tester l'identification d'opérateurs de chevauchement. Deux séries principales d'expériences ont été réalisées. La première a seulement utilisé des données non corrompues. La deuxième a utilisé trois différents paramètres de données corrompues. Ces expériences corrompues utilisent des séries de données qui contiennent seulement des erreurs de commission, seulement des erreurs d'omission et une combinaison des deux.

Les résultats obtenus en utilisant des données non corrompues montrent que l'algorithme est capable d'identifier tous les opérateurs sans aucune erreur, peu importe s'il existe ou non des opérateurs de chevauchement. Quand il existe seulement des erreurs de commission, tous les opérateurs d'oracle sont correctement identifiés. Toutefois, un certain nombre de définitions superflues sont parfois produites. Il est prouvé que le plus difficile est de faire face à des erreurs d'omission. Quand de telles erreurs sont injectées dans les séries de données, les opérateurs de domaine ne peuvent pas tous être correctement identifiés. Comme dans le cas d'erreurs de commission, des définitions additionnelles d'opérateur incorrect peuvent aussi être causées. Même si la détermination des solutions correctes en présence de corruption n'est pas possible, l'algorithme est bien élaboré et produit des résultats avec de faibles taux d'erreur.

Declaration

The work in this thesis is based on research carried out at the Faculty of Engineering of the University of Porto, in the Department Informatics Engineering, Portugal. No part of this thesis has been submitted elsewhere for any other degree or qualification and is all my own work unless acknowledged to the contrary in the text.

© Hugo Miguel Mendes Ferreira, 2011.

Acknowledgements

Before I undertook this project, I was careful to read up on others' experiences and advice. A common theme was that such a project was guaranteed to be a memorable journey, one among many others to be taken in life. The advice given also pointed out that the word memorable should not be confused with the word enjoyable. I attest to this wholeheartedly. It's been a trek that has made an indelible mark on me. However, it has also been a gratifying experience, albeit in ways I did not foresee. As with any trip, I owe my thanks to those who helped me prepare, execute and endure such a journey.

First and foremost my thanks to Prof. João José Pinto Ferreira, my co-advisor, who so diligently *nudged* me into doing a PhD in the first place. To my advisor Prof. Rui Carlos Camacho de Sousa Ferreira da Silva, whose rigorous attention to detail in the research area of machine learning, has set a higher standard to which I now aspire to. I also appreciate his time and effort in managing the computer cluster that has allowed me to execute the literally hundreds of experiments that were required.

A special thank you goes to Prof. José Manuel de Araújo Baptista Mendonça and Prof. João António Correia Lopes, whose letters of reference were undoubtedly responsible for the acceptance of my PhD grant, without which none of this work would have been possible. A note of thanks to Prof. Jorge Manuel Pinho de Sousa who took the time to review my curriculum vitae when I applied for the PhD grant.

I am in-debt to INESC Porto, which allowed me time off to pursue this work and provided me with and the necessary administrative support. My thanks to Luís Carneiro, the unit's director, who graciously accepted to act as host by providing me with a place to toil. I also owe my gratitude to many colleagues in INESC

Porto, some of whom have already moved on to other places. My thanks go to Diogo Ferreira for the initial research in this area, which resulted in two articles, one of which garnered a best paper award. To Ana Viana and Abdur Rais, whom I pestered with my questions on combinatorial optimization. To Nicolau Filipe Santos who suggested a greedy algorithm. To Pedro Souto, Iryna Yevseyva and Luís Lima for feedback on measuring and processing noise generated errors.

I have had the pleasure and privilege of receiving help from many people who I have never personally met. The list is much too long to enumerate. However I would like to acknowledge the help of those few who went *above and beyond the call of duty*. To the people in the SWI Prolog mailing list, especially Jan Wielemaker, Bart Demoen and Richard A. O'Keefe, who helped me think for myself. To that talented group in the Ocaml mailing lists, in particular Jean-Christophe Filliatre, Jon Harrop, Richard Jones and Martin Jambon whose provided example code and helped me overcome the, sometimes, unruly typing system. I also received aid from people of the Artificial Intelligence planning community. Thanks goes to Alessandro Cimatti, Piergiorgio Bertoli, Blai Bonet, Corin Anderson, Daniel Bryce, Haakan Younes, Jörg Hoffmann, Jussi Rintanen and Rune M. Jensen. A big thank you to José Santos for his perseverance in describing his subsumption algorithm and to Ondrej Kuzelka for also providing additional assistance in this area. To Geoff Sutcliffe for his help in the normalization algorithm of logic formula. To all those I have neglected or forgotten to mention, my sincere apologies.

Finally, a very special thank you for those closest to me. To my parents, brothers and in-laws whose help and company I dearly cherish. To my wife Paula who has endured this ordeal, I can now give your more of what is most precious to me, my time.

This work was wholly supported by the Portuguese Foundation for Science and Technology (FCT), which provided the PhD grant reference n° SFRH/BD/41983/2007.

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
ARMS	Action-Relation Modelling System
B2B	Business to Business e-Commerce
BDD	Binary Decision Diagram
CSP	Constraints Satisfaction Problem
CTL	Computational Tree Logic
DAG	Direct Acyclic Graph
DNF	Disjunctive Normal Form
EBL	Explanation Based learning
HTN	Hierarquical Task Network Planning
ILP	Inductive Logic Programming
LOPE	Learning by Observation in Planning Environments
MAX SAT	Maximum Propositional (Boolean) Satisfiability Problem
MBP	Model Based Planner
MCSC	Minimum Consistent Subset Cover Problem
ML	Machine Learning
MSDD	Multi-Stream Dependency Detection algorithm
PDDL	Planning Definition Description Language
POMDP	Partially Observable Markov Decision Process
POP	Partial Order Planning
RLTB	Relational Learning ToolBox
SAT	Propositional (Boolean) Satisfiability Problem
SLAFS	Simultaneous Learning and Filtering of Schemas
STRIPS	Stanford Research Institute Problem Solver

TRAIL	Teleo-Reactive Agent with Inductive Learning

Chapter 1

Introduction

Our work addresses the problem associated with *workflow* management. In the context of this document Workflow refers to the automated management of processes. This includes, amongst others, document management, office automation, administrative workflow, inter-enterprise Business to Business e-Commerce (B2B) processes orchestration and choreography and scientific workflows.

The study of process automation, especially in the context of human orientated processes, include a myriad of interesting and relevant issues in many areas such as knowledge[1] representation, knowledge management, process modelling, modelling languages, requirements elicitation, and many of the social sciences (Sociology, Ethnography and Language).

In addition to this, several authors have attempted to identify and use alternative theories and methodologies to deal with many of the softer problems related to process (re)engineering (organisational learning, institutional theory, organisation culture and structuration) [SHH95, BR96]. Specifically cultural issues such as changing behaviour and attitude and resolving conflicts associated with promoting commitment and employee empowerment are dealt with. Nevertheless in our study we focussed only on the technical aspects related specifically to process automation which includes process model representation, model construction and process

[1]We abuse the term *knowledge*. Knowledge as we use it refers to both the artefacts (text, figures, models, ontologies, dictionaries, etc.) and actions (dialogue between people, argumentation, critiquing, memorization, learning, (automated) inference, (automated) verification, etc.) that give rise to knowledge.

control.

Even though we do not deal with the softer issues associated with the more general and encompassing subject of process (re)engineering, we believe this work also contributes to research that generally advocates the use of (continuous) organisational learning [BR96] and knowledge management as an important means for supporting process design and automation [Ber00].

1.1 Problem Statement

Much attention has been given to *workflow systems*, both in academia and in industry. Research in this area goes as far back as the 80s with the development and deployment of office automation systems [BP84]. Even though a lot of effort has been put into these systems, they are still considered inflexible and brittle [JSM+99, Car97]. The rigid control of workflow systems does not provide adequate support for ad-hoc or collaborative processes. These processes are characterised as being unstructured, knowledge intensive and highly context-dependent processes [Jør04]. They depend on the creativity and adaptability of the human participants. Examples of such processes include, amongst others, consulting and engineering. As a result, research in these domains has focused on providing what is commonly known as *ad-hoc* workflow systems, also referred to as *adaptive*, *flexible* or *intelligent* workflow management systems. Some of these works specifically target ad-hoc processes [Car97, Jør04], while others implicitly attempt to enhance system adaptability (for example [BD00, JSM+99]).

Much of the inflexibility of the workflow management systems stems from the fact that process deployment is done in two phases. The first phase is concerned with *process synthesis* (modelling, testing and correction), while the second one is concerned with *process enactment* (process execution and process instance administration). This presents two important issues. First and foremost, process models by definition are rigorous, formal descriptions of the world amenable to automated computation. Therefore one cannot conceivably expect to foresee and to *plan for all contingencies* thereby creating the perfect model [Jør04].

Second, as the models will never be complete they must be *continuously adapted and corrected*. Naturally this was known since the inception of workflow management systems, so process model correction and redeployment have always been an integral part of the workflow management lifecycle [zM04,GT97]. However, the rigid process models and the separate modelling and enactment phase make it difficult to continuously incorporate corrections and quickly adapt to changes.

All research in this area has therefore attempted to: a) facilitate or enhance modelling; and b) ease process adaptation and correction (thereby compensating for modelling omissions and errors). Much of this work has given more importance to either one of the issues in detriment to the other. Solutions usually trade model completeness for system reactiveness or vice-versa. The investigation that concentrates on the *modelling aspects* are usually concerned with knowledge management in general, modelling language expressiveness and ease of use, formal and simulated model verification, modelling aids in the form of suggestions and critiquing, model reuse and model identification and extraction. Research directed at *process adaptability* and correction, in its turn, looks at dynamic process synthesis, dynamic and intelligent resource allocation, automatic fault detection and correction, process auditing and analysis and soliciting, promoting and incorporating changes based on process execution experience. The result is that the models become richer and more expressive, sophisticated methods of inference are adopted and ultimately the distinction between the modelling and execution phase becomes blurred [zM99, MB99a].

Current research in adaptive workflow has succeeded in automating several stages of the process management lifecycle, namely process composition (planing the sequence of tasks that constitute the process in order to attain a goal), process execution (controlling the execution of the tasks in order to complete the process) and process correction (identifying and correcting plan execution). Some of this work has also succeeded in automating the transition between these process management stages. However, to the best of our knowledge, the *automation of the full workflow process management lifecycle* has yet to be achieved. This is due to two main reasons: a) modelling still requires the active involvement of the users to enumerate the possible tasks; and b) fault detection and error correction do not attempt to

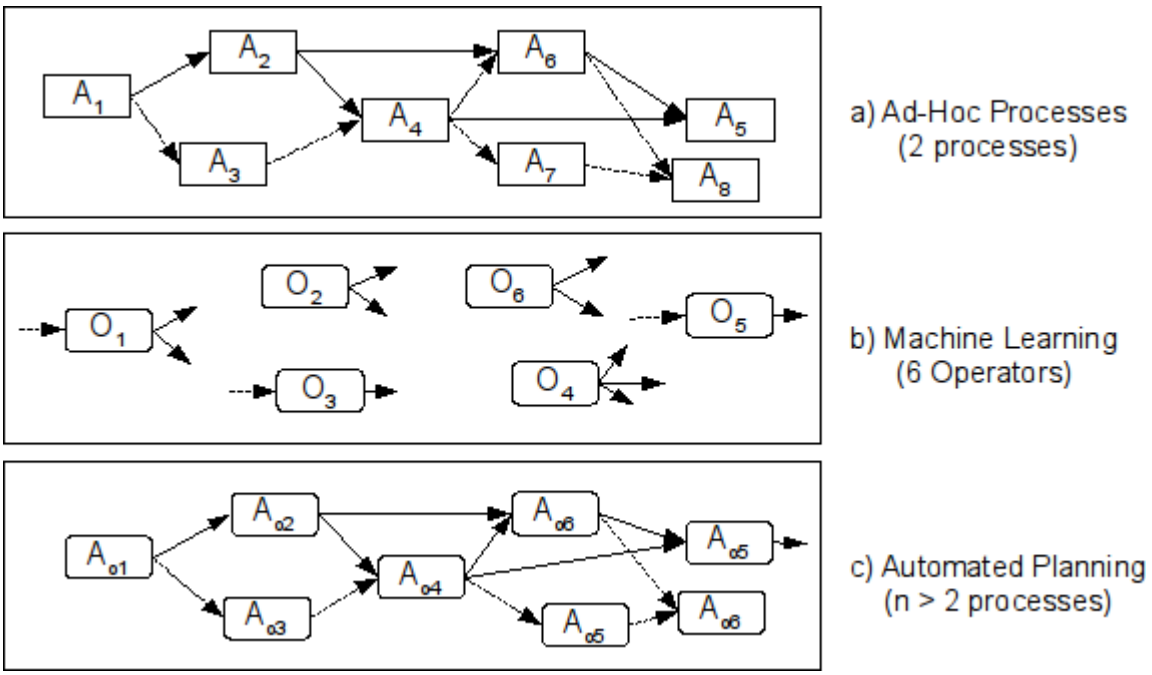
identify modelling omission and automatically incorporate these into the process models themselves.

This thesis contributes to the state-of-the-art by extending processes adaptability at the processes modelling level. It does this by analyzing the execution of actions of a set of processes, which will then allow for the automatic composition of workflows. More specifically we learn the definition of planning operators, whose instances formally describe workflow actions. The composition of workflows may then be accomplished automatically via the use of an AI planner that generates a plan. In this way we a) avoid much of the modelling effort by learning directly from users' behaviour and b) allow for the incorporation of changes into the models as soon as errors or omissions are detected. Learning task (or operators) definitions may therefore make it possible to automate all stages of the full lifecycle of the workflow process management by providing solutions to the two main issues referred to above.

The relationship between planning operator, workflow task, process and plan are shown in Figure 1.1. An ad-hoc process consists of various actions with many possible interactions (see the A_i in Figure 1.1 a)). The learning algorithm should be able to automatically determine the definition of the various actions (A_i) that are executed within one or more ad-hoc processes. Note that a single planning operator (O_i) may represent one or more instances of an ad-hoc process action. In other words there is not a one-to-one mapping between the A_i in Figure 1.1 a) and O_i in Figure 1.1 b). A planning system may then be used to automatically compose an ad-hoc process by instantiating and sequencing the appropriate actions in order to satisfy one or more goals.

It is important to note that a single ad-hoc process may have many possibly equivalent plans. In other words a set of different operator instantiations applied in varying sequences may attain the same goals. The plan (workflow process) that is composed depends on the AI planning algorithm used. This in turn requires that the Machine Learning algorithm acquire operator definitions that use a specific formalism. We must therefore, first select an appropriate planning algorithm that supports the composition of ad-hoc worklfows (see Section 2.3.3) and then opt for a Machine Learning algorithm that is able to learn planning operators that use the

Figure 1.1: Learning Planning Operators to Automated Processes Composition.



selected planning formalism (see Section 2.3.5).

Even though this research focus on these two problems, many other issues will remain open. Issues regarding security, privacy, accountability and process control will not be dealt with. These issues focus on process' rigid model, that in effect enforces control, but which may be inadvertently circumvented. Such control has the basic aim of imposing behaviour that is required to ensure process rationalisation (enumerated as one of workflows' major advantages) [Fis02] or even conformance to legal requirements. In addition to this we will not delve into issues regarding knowledge representation of the background knowledge (ontologies, controlled vocabularies), natural language processing (dictionaries, thesauri) nor protocols (normative behaviour, agent collaboration and competition), which are required for facilitating communication between the workflow participants themselves.

The ultimate goal of this work is to *facilitate knowledge acquisition and dissemination*. It should provide the means by which cooperating users may share and possibly create new process knowledge. We do not foresee, nor do we attempt, to remove the human element from workflow management. People are the autonomous

agents that will establish the goals and therefore set the system in motion. It is they that provide the necessary intelligence and context awareness to drive process creation, use and evolution.

The acquisition of (tacit) knowledge is often difficult and fails because: a) people are not necessarily aware of the expert knowledge they possess; b) they have no need to make such knowledge explicit in order to use it; and c) presents a dilemma because it means they loose a "valuable competitive advantage" [Ste01]. In order to circumvent such problems, knowledge elicitation may: a) acquire tacit knowledge indirectly (recommender system example: instead of enumerating concepts of interest, point out documents of interest to update a search profile); b) explicitly provide the user with added value so that he is motivated to share such knowledge (recommender example: letting the shared knowledge be used to find information of interest by recommending documents based on similar profiles); and c) not make the knowledge explicit but allow the user to retain it (recommender system: user keeps and changes his profile independently of others). We use these results as guidelines in this research.

Accordingly, we attempt to non-intrusively acquire expert knowledge by observation during problem solving episodes [RSDC99]. Learning by observation allows us to acquire tacit knowledge that is effectively used during problem solving and not depend on explicit, conformant and possibly irrelevant knowledge that is adopted in organizations (for example job descriptions do not adequately describe ones responsibilities and tasks). This should allow one to substantially reduce the costly and possibly undesired need of eliciting expert knowledge.

1.2 Objectives

The problem statement described above can be detailed further and expressed in the following questions to be addressed:

> Can we automatically identify and learn what the constituent parts of a process are by the simple and non-obstructive observation of its execution?

The objective is to ascertain if in effect it is possible to learn enough from the executions of a process in order to automatically construct a model for it. To give a better understanding of how this will promote the dissemination of knowledge amongst workflow participants, consider a user that has a set of goals to achieve but does not know (or does not have a clear idea of) how to do go about it. Chances are that others have had similar experiences and have already successfully achieved these goals. The user therefore queries the system in order to obtain the list of actions necessary to achieve the goals. A process is then generated based on the other participants' previous experiences. The process may now be executed, however any new information gained from this new experience is also incorporated into the system so that someone else may later on take advantage of it. As the workflow participants execute their tasks so the knowledge base should increase providing a medium for recording and sharing process information.

In the above scenario several additional practical problems exist. These include for example: sampling rates, incomplete and erroneous observation of actions, dynamic and possibly inconsistent changes in the background knowledge base, the identification and setting of goals using a common semantics shared by all agents, non-deterministic and probabilistic actions and the partial satisfaction of goals.

Some of these represent important, complex problems in themselves that require in-depth research and much effort to solve. In order to make the current work feasible these issues have been identified and pointed out, but we will nevertheless not endeavour to solve them all. Instead, several simplifying assumptions are made. However, care has been taken to make assumptions that will, not only, allow for the practical implementation of a system based on our results, but also be useful. Examples of such assumptions include: full but possibly erroneous observation, sampling triggered by events only, all dynamic changes to the background will be consistent, all information that is exchanged is based on a common, previously agreed upon syntax and semantics and all actions are instantaneous.

1.3 Research Approach

1.3.1 Research Areas

According to the problem statement (Section 1.1) and objectives (Section 1.2) research was conducted in order to identify the current repertoire of techniques and tools used in (adaptable) workflow management. Emphasis was placed on flexibility and automation. This has provided the basis for identifying promising areas to work on. Apart from the research material related to workflow management, we have also done some preliminary work [Fer06] on Automated Planning and Machine Learning. Subsequently these areas of AI were analysed and studied with the aim of enabling the acquisition of process knowledge (task identification, task description, consistency checking) and its use in the automated synthesis of processes (composition, error detection, error correction). This thesis however focuses only on the very specific area of learning planning operators.

1.3.2 Evaluation and Validation

We formulate the hypothesis that current workflow management systems, in order to provide flexible and adaptable systems that support ad-hoc processes, may be further enhanced by combining the acquisition of information via the observation of agents in the execution of their tasks and the automated synthesis and correction of workflow process. To be able to answer this fully, one would require qualitative and quantitative evaluation of experimental results using a workflow management system that implements the proposed solutions. This is much too ambitious for the following reasons:

- The implementation (or adaptation of) a (robust, stable and bug free) worklow system would require a fully fledged development project;
- Qualitative and quantitative verification would require the installation and use of the workflow management system by various enterprises for prolonged periods of time;

- Qualitative and quantitative verification require lengthy comparative studies that depend on surveys, interviews and detailed analysis of auditing data;
- The core algorithms and methods are experimental in nature and therefore require further work before being used in the development of robust and usable workflow management systems.

On the other hand the methods proposed in this thesis are empirically evaluated based on several experimental settings using simulation. This has several important advantages:

- Provides a ready source of data for processing and analysis;
- Allows the control of simulation parameters that facilitate basic algorithm testing and evaluation;
- Facilitates quantitative (albeit empirical) analysis via simple well known metrics;
- Reduces the dependency on external factors for testing;
- Reduces experiment set-up and evaluation times;
- Does not require solving related issues not directly related this work (see Section 1.2).

It also presents some disadvantages which include:

- It does not provide a base for comparative analysis with existing solutions;
- Fails to provide a realistic test-bed that enables the identification of new problems in this context;
- Does not allow for qualitative analysis based on user feedback.

Because of this, experimental design takes some precautions in order to use realistic data and obtain useful and realistic results. This includes, but is not limited to, simulating sensing and sampling errors, considering multiple agents concurrently executing processes and task execution failure. Such caution should provide this study with a realistic and useful test-bed in which to work.

1.4 Contributions

The aim of this work is to improve the state-of-the-art in workflow management. Specifically it should allow for the future implementation of intelligent workflow management systems that not only adequately support ad-hoc processes but also, to a certain extent, provides a platform that facilitates knowledge acquisition and dissemination. Specifically the resulting artifacts of this work include: a simple simulation framework, test data (not included in simulation) and a Machine Learning algorithm that acquires process action definitions that may be used by a planning system.

1.5 Outline of the Thesis

The remaining chapters of this thesis are organized as follows. In Chapter 2 we review the related work. We start off by looking at *workflow mining*, which specifically deals with the identification of workflow processes based on the execution logs of workflow engines. We point out several problems associated with these approaches and explain why we have chosen automated planning as means of generating workflow process definitions. Next we give a general presentation on automated planning, its use in the construction of processes definitions and how this results in the need to learn operator definitions.

In Chapter 3 we describe the Machine Learning algorithm that was used to acquire the planning operators. We first provide a set of formal definitions, which includes descriptions of the data sets used and the type of operator definitions that were learned. We then express the learning problem as a specific instance of the general Minimal Consistent Subset Cover Problem. The description of the algorithm is done in three parts. The first deals with learning standard operator definitions that are encountered in the planning domains used by the automated planning research community. The second describes how the algorithm may acquire a type of operator definition found only in a planing domain that describes a workflow process. Finally we show how the algorithm can learn using noisy data sets.

In Chapter 4 we detail the design and execution of the experiments. We first

describe the planning domains that were used to generate the data sets. We then explain how the generation of this data is parametrized in order to create several test scenarios. A set of error measures are then described that were used in evaluating the results. Experimental results are presented in two main groups: those experiments that used noiseless data sets and those experiments wherein noise has been injected into the data sets. Finally we analyse these results.

Chapter 5 presents the conclusion of this work. It reviews the contributions that were made based on the experimental results and ends with a description of the work that has been planned for the future.

Two appendices are also included. Appendix A contains formal descriptions of all of the domains used in the experiments. Appendix B contains all of the results that were obtained. Each result is represented as graph split into three parts, each part representing an error measure that was used in evaluating the results.

Chapter 2

Related Work

2.1 Introduction

Flexible workflow support for ad-hoc processes should combine long-term process evolution with short term adaptation. Very few studies have attempted to support both short and long term process adaptation. To the best of our knowledge all research on *long-term process evolution* is based on a process level analysis (workflow mining). None of these solutions allow for task based adaptation. Neither do they take advantage of the possibility of automatically composing processes in order to support both short and long-term changes. In our work we have addressed the combination of the support for long-term process evolution of processes (based on task adaptation) and short-term plan changes (based on automated process composition).

Much research has been done in the area of AI planning providing us with a wealth of information and an extensive range of solutions. In order to focus our attention on the more relevant material and facilitate the description of AI related techniques, we first characterize the planning requirements that enable us to compose and execute workflow processes successfully. This will allow us to select and use the appropriate planning system in the domain of automated workflow planning. The objective is to use AI planning to automatically synthesize processes and thereby support short-term changes. The emphasis here is on system reactiveness.

We will look next at research in Machine Learning specifically directed at AI planning. This provides us with the means of evolving plans and thereby supporting

the long-term evolution of workflow processes. The objective here is to describe what planning techniques have been employed and what problems have been encountered and solved. The focus is on the learning techniques that allow us to automatically acquire a planning model, specifically to learn what planning operators exist and what their definitions are in order to be able to plan.

Not all issues related AI planning will be dealt with. Specifically, we will not delve into issues related to: re-planning, plan repair, planning with resources, mixed initiative planning and planning aids (consistency and critiquing). This will allow us to focus on the contributions of this thesis, which is the support of process evolution based on Machine Learning to acquire operator definitions.

2.2 Workflow Mining

The area of workflow research that attempts to identify process models is known as process mining (see [vdA10, vdAvDH+03, AGL98, ZGLL03, HK04, GABG05], for a short introduction and case study see [IG06]). Much work has been done on this subject (see [vdAvDH+03] for a survey). Process mining's main motivation concerns process discovery and optimization. The emphasis here is on identifying and correcting inefficiency (performance indicators such as wait-time and utilization can be used) in process executions. Most of the work attempts to identify a process model by analysing the sequences in which actions are executed during the processing of equivalent cases. The resulting process model consist of a set of activities that are ordered according to their dependencies. This research area has also influences from other areas of research such as identifying the interaction between ad-hoc web-services from the control-flow perspective ([GBG08]) and modelling the behaviour of multi-agent robotic systems ([RZV+09]).

Process mining poses many challenges that have yet to be adequately solved. Such challenges include basic issues in identifying duplicate tasks, loops, ensuring model completeness while minimizing model complexity and handling noisy data ([WWvdA+10]). Many of the studies attempt to solve these problems by directly evaluating and optimizing the quality of the process ([dMWvdA07]), fa-

cilitating selective data mining and process model creation ([vdARV+09]), using well known relation mining techniques to deal with noise and background data ([GMVB09], [GMB+07]), using time information to better establish dependencies ([WWA+09]), allowing for the interactive analyses of processes based on temporal mining ([BPNG09]) or limiting the generation of the models to the case of block structured processes ([Sch04]).

In most of the approaches the resulting process model is obtained in two phases: first the data is mined and then a process model is generated. A popular control model that is used is Petri Nets ([Rei92]), which itself introduces some problems namely: the need to identify invisible[1] tasks ([WWvdA+10]) and non-free-choice [2] constructs.

First, we believe that all the issues mentioned above stem from a common root cause. The main problem is that the effects of an action and therefore the flow control dependencies cannot be fully determined solely by their ordering. This is true for the case where datasets that are complete (in the sense that all of the possible ordering of all actions for a given case are available) do not exist. This problem also manifests itself in the identification of multiple occurrences of the same action in loops. In order to cope with this, some work establish a horizon of time for analysing global effects ([GMVB09]), others attempt to determine this global horizon dynamically ([GBG08]) or simply consider only direct dependencies ([Sch04]). These strategies however do not fully solve the problem and we find, for example, that only short loops (loops with a limited number of repeated actions) can be identified. This problem is exacerbated when the data is noisy.

The second issue is that some of the problems listed above are due to the use of Petri Nets as a formalism for the flow control model. More specifically places represent actions (events) and not the conditions under which the actions are executed. This means, for example, that when an action is skipped, an invisible task must be identified and added to the model ([WWvdA+10]). In other words, a clear

[1]Petri Net places that do not represent concrete actions but are used for flow control such as skipping another action.

[2]This represents a Petri Net transition that is both used for synchronization and flow control such as a choice.

distinction between the actions that are executed and the effects they have on the running process are not made.

For the reasons above, we have adopted an alternative formalism that is used in automated planning ([Tra04]). Automated planning clearly differentiates between an action, its pre-conditions and effects. Such a distinction also avoids the Petri Net's problem of non-free-choice transitions because the pre-conditions establish how synchronization will be attained and the effects determine flow control. Automated planning also has a large repertoire of algorithms that can be used for the automated generation of plans. Such plans essentially express a control flow model. We will focus particularly on learning planning operators that can be used by Model based planners, which allow for the generation of state machine like plans that support the execution and control of actions with non-deterministic effects and loops (iteration) ([CPRT03]).

2.3 Automated Planning

2.3.1 Concepts and Definitions

AI planning is concerned with action planning in general. Many types of action planning exist such as motion and path planning (control trajectory of mobile robots), perception planning (sensing actions), navigation planning (combination of path and perception planning), manipulation planning (building assemblies) and communications planning (collaboration) [Tra04]. Planning attempts to build a predictive model of the system's possible states and what actions need to be executed in order to transition between these states. AI planning is concerned with the automated synthesis of these predictive action models.

AI planners may either be *domain-specific* (inference procedures specifically designed for a given domain), *domain-independent* (planning algorithms that use general planning domain descriptions) or *domain-configurable* (planning domain also includes domain-specific knowledge that constraints the planners search) [Nau07]. The most efficient planning systems are domain-specific and is the reason why most of today's applications use such planners [Tra04]. Domain-configurable planners also

provide an efficient viable solution, nevertheless we will focus our attention on the use of domain-independent planners. The reason for this is our aim to support the automated synthesis of workflow processes. Because workflow processes cover a wide range of domains, it is not reasonable to limit ourselves to a given domain-specific planning algorithm.

Planning is concerned with the selection and sequencing of actions in order to change the state of a system. The model of *state-transition* (or discrete-event) systems provide a general conceptual framework that describes such a dynamic system. Formally, a state-transition system (in [Tra04], pages 5-6) is a 4-tuple system $\Sigma = (S, A, E, \gamma)$ where:

- $S = \{s_1, s_2, ...\}$ is a finite[3] or recursively enumerable set of states,
- $A = \{a_1, a_2, ...\}$ is a finite[4] or recursively enumerable set of actions,
- $E = \{e_1, e_2, ...\}$ is a finite or recursively enumerable set of events,
- $\gamma : S \times A \times E \rightarrow 2^S$ is a state transition function,

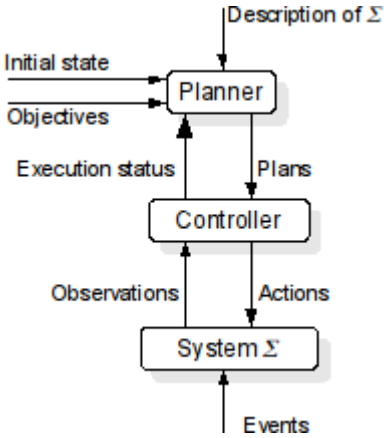
The state-transition system can be represented as a directed graph whose nodes are states in S and the edges are the state transition between nodes. State transitions may occur either due to the execution of an action ($\gamma(s, a)$[5] where $a \in A$) or the occurrence of an event ($\gamma(s, e)$[5] where $e \in E$). The events $e \in E$ represent contingent transitions which cannot be controlled via plan execution but describe the intrinsic internal behaviour of the system. We say that an action a is *applicable* to state s when $\gamma(s, a) \neq \varnothing$; in other words when the action a is applied to state s it will transition into a new state $s' = \gamma(s, a)$. Note that the state transition function γ may partition state transitions into separate action and event state transitions by considering actions and events separately. Planning, however, may also support the combined action-event transitions $\gamma(s, a, e)$ (for example via the use of temporal effects).

[3]We will see later, when dealing with probabilistic planning, that such a set may be infinite.

[4]We assume that we will only be dealing with planning problems where a finite number of objects are used, otherwise this set could be infinite.

[5]For convenience we introduce a *no-op* action and a neutral event ϵ so that we write $\gamma(s, \textit{no-op}, e)$ as $\gamma(s, e)$ and $\gamma(s, a, \epsilon)$ as $\gamma(s, a)$.

Figure 2.1: Conceptual model for dynamic planning [Tra04].



The objective of AI planning, given the a state transition system Σ, is to determine the sequence of actions that will result in an ordered set of transitions from an initial state s_0 or initial set of states S_0 to a goal state s_g or a set of goal states S_g. The sequence of (sets of) actions is called a plan denoted by π. An initial set of states S_0 is used to model uncertainty in the initial state of a planning problem. We will use the term belief state to refer to the set of states S_0. In the case of set of goals S_g, the objective is to reach any of the states $s_g \in S_g$. Note that planning objectives need not be so restricted. Planning paradigms (such as probabilistic planning based on Partially Observable Markov Decision Processes (POMDP)[6] and symbolic model-checking based planning [CPRT03]) allow for the use of extended goals. Various types of extended goals can be defined: states to avoid, states the system should maintain and state that must eventually be reached. A planning problem is defined as the triplet $\mathcal{P} = (\Sigma, s_0, S_g)$. Planning problems that use these concepts will be formally and more rigorously defined in Section 2.3.2. Both, the set of goals and planning paradigm determine the type of plan (π) generated. In order to better understand the role that AI planning plays, we may illustrate the relationships between the following three main components that constitute a planning system (see Figure 2.1) [Tra04]:

- A *state-transition system* Σ that evolves according to the events that occur

[6] http://www.pomdp.org

and actions that are applied to it;

- A *controller*, that at a given state s, executes the action a according to the plan π;
- A *planner*, that given a state-transition system Σ, an initial (s_0 or S_0) and final goal (s_g or S_g) synthesises a plan π.

Two important elements must be added to this model. The first is that it is necessary to model uncertainty. Later we will see that state transition systems may be used to model uncertainty by including a set of additional states that represent discrete albeit non deterministic outcomes or associating probabilities to several state transitions. Second, not all state information is available to the controller. This is modelled by an observation function $\eta : S \rightarrow \omega$ where ω is the set of discrete observations $\omega = \{o_1, o_2, ..\}$. The input to the controller is therefore the set $o = \eta(s)$, which represents the current state.

Classical planning problems have several possible representations, which have the same expressive power because each one may be translated into any other. Classical planning may use one of the following types of representations:

Set-theoretic representation The states S in Σ are expressed as a set of propositional formulae. The actions also consist of a set of propositions. These propositions specify when an action is applicable (pre-conditions) and the resulting state when such an action is applied (post-conditions). The transition to state $s' = \gamma(s, a)$ is defined by a set of propositions that are added to and removed from s.

Classical representation The states and actions are described in the same way as the set-theoretic representation, however first order logic and logical connectives are used. This is the most widely used planning representation because it is compact and clear.

State variable representation The state is represented by a set of n state variables $X = \{x_1, .., x_n\}$ assigned certain values. The applicability of the action is determined by a subset of state variables and their corresponding values,

which must hold in the world state s. The state transition is defined as a partial function that maps the tuple of values s to a another tuple of values s'.

These basic representations can be extended in various ways. This includes the use of axioms, quantified expression (universal and existential quantification in both the pre-conditions and post-conditions), conditional planning (post-conditions depend on the pre-conditions), disjunction in pre-conditions, functional symbols (for example incrementing a counter) and attached procedures (for example the comparison of numerical values). In additional to this, the use of extended goals used in non-classical planning can also be mapped into classical planning by, for example, altering the definitions of the actions themselves. This may include for example: excluding certain states from the search, enforcing a specific order in action execution and enforcing the execution of a minimum number of actions. These types of extensions are nevertheless limited.

2.3.2 AI Planning Paradigms

Research in this area has evolved with the growing capabilities of the planning algorithms. Initial emphasis was on efficiently generating plans for deterministic environments, referred to as classical planning. Currently work is now focused on extending the planning to cover more complex environments dealing with ever more realistic scenarios [Nau07]. As a result, many types of planning paradigms exist. Each type makes a different set of assumptions, uses specific algorithms to generate a plan and yield possibly more than one type of plan to solve the specific problem. A very large body of research with a diverse set of goals exists within the AI planning community. This work could be classified according to various criteria which include the planning algorithms used [Tra04,Nau07], planning principles [RH01], the types of plans generated, the assumptions under which the planning is defined or even based on a view of its most significant developments [Wel99]. In addition to this, research in this area covers many other topics, which include but are not limited to: replanning and plan repair (for example [vdKdW04,vdKdWW03,FGLS06,AKYG07]), resource aware planning and scheduling (for example [HS07, RKM06]), learning in planning

and scheduling ([LD06, YFG06, NLFL07, MOJ+07]), mixed initiative planning and scheduling ([ACBC+04, CC06]) and the interleaving of planning and execution for example in [WSK+05, LAP06]). Naturally these subjects are not independent.

Because our objective is to identify and use a planning paradigm that will allow us to adequately support the automated composition and execution of workflows, we will classify these according to the characteristics of the planning problem and the appropriateness of the plan type to solve it. Planning problems may be categorized according to [7]:

Plan action *determinism*: a planning domain may model actions whose effects are deterministic. In other words, once an action is executed, its effects are guaranteed. For example if the action of picking up an object takes place, then the planning state will always reflect the fact that the object has been moved. Alternatively the effects of an action may be *non-deterministic*. In this case, picking up an object may fail. This possible failure may be encoded as two belief states: one belief state maintains the object's previous position (failure) and a second belief state mirrors the fact that the block has been moved (success). In addition to this, non-deterministic outcomes may be assigned a probability. For example we may say that the picking up an object may fail 20% of the time. Such actions are said to be non-deterministic with a *stochastic* outcome. In such cases we refer to the belief states as probabilistic belief states.

Plan state *observability*: describes the ability to monitor all modelled states during plan execution. This implicitly determines the types of plans that are generated. For a given planning domain we may have *full, partial or no observability*. *Partial observability* requires that a sensing action be explicitly executed in order to acquire the state. Note that the lack of complete observability is not limited to the period of plan execution. The initial state may

[7] This classification and the formal description used are based on the 6^{th} International Planning Competition Rules drawn up by Daniel Bryce and Olivier Buffet for the uncertainty track, see http://www.icaps-conference.org/. A alternate and more complete characterization of planning problems may be found in [Tra04] in Sections 1.5 and 1.6 and [Nau07].

also include some measure of uncertainty.

Depending on the combination of the actions' *determinism* and the states' *observability*, several classes of planning systems may exists. The various possibilities are enumerate in Table 2.1.

Table 2.1: Plan types.

Action Determinism	**State Observability**	**Planning type**
Yes	Full	Classical (optimize) Classical (satisfice)
	None	-
	Partial	-
No	Full	Conditional Probabilistic
	None	Conformant
	Partial	Probabilistic

The planning types above can be described via a high level language such as the Planning Definition Description Language (PDDL) or any of its variants, which are based on propositional logic [FL03]. The initial and goal states are represented by logical formulae. The set of applicable actions (operators) and transition functions are described by action schemata. This schemata is based on STRIPS[8]-like operators [Tra04]. Various levels of expressiveness are required to describe these models adequately. This includes modelling constructs such as: variable typing, equality, negation, disjunction and existential quantification. Additional expressiveness may also aid the efficient execution of automated inference procedures by including domain-specific knowledge (control-rules) that allows search space pruning [Nau07]. Note that the various planning techniques are not limited to propositional logic and may also be encoded and solved as (mixed integer) linear programming, non-monotonic logic programming and CSP [RH01,Nau07]. In addition to this the basic operator schema may be augmented with information such as cost[9] and probability of occurrence in order to support various types of planning problems.

[8]Stanford Technology Research Institute Problem Solver.

[9]We will use cost in all our descriptions of planning problems, but could alternatively use reward, for example in relation to probabilistic planning.

Classical and Neoclassical Planning

In this section we describe all planning types wherein the effects of an action are not stochastic in nature. *Classical* planning systems assume full observability and action determinism. The corresponding planning algorithms may generate *partial-order* (non-linear) [Wel94, PW92, NK01], *optimal* [BF97] or *linear* plans. Both *partial-order* and *optimal* planning minimise plan length (or make-span) by allowing for the parallel execution of actions. Classical *linear* planning, on the other hand, only allows for the sequential execution of actions. Note that the notion of optimality is not limited to plan length. Other criteria can be used to evaluate plan quality. Such criteria can be generically modelled via a cost for executing an action. In the case of make-span this cost is a single constant value that is attributed to all actions.

Although neoclassical planning algorithms based on *Graphplan* [BF97, KS99] generate minimum length plans, these plans are less flexible than those generated by *partial-order* planners because concurrently executing actions are sequenced at specific (fixed) intervals. So, even though several algorithms may solve the same type of planning problem, the plans that are generated have different characteristics.

Note that all of classical planning is optimal in the sense that only the minimum number of tasks required to reach the goal are used. However for larger problems the computing time required to guarantee optimality is excessive. In such cases a combination of non-admissible pruning strategies and heuristics may be used to reduce the search space, which will result in acceptable, albeit non-optimal, plans. We refer to classical planning of this kind as satisficing.

A more realistic setting however assumes that actions are *non-deterministic*. In other words, task execution is not guaranteed to succeed. In AI planning such failures are modelled as a set of possible outcomes of an action or as an initial set of unknown facts in the planing problem. The possible consequences of non-deterministic actions are described in a operator's post-conditions.

In the case of *non-deterministic* planning with *no observability*, plan generation must guarantee that the goal is reachable irrespective of the actual initial state and the actions' true effects during execution [SW98]. This is known as *conformant* planning. Note that not all problems may have a conformant solution. Conformant

planning caters for problems with no observability and non-deterministic actions.

In the case of *non-deterministic* actions with *full observability*, *conditional*[10] planning is used [Rin04, WAS98, PS92, BCRT06]. Conditional planning attempts to generate robust plans that consider all possible non-stochastic consequences of the actions and select the next appropriate action based on these outcomes. The set of states that describe all these possible outcomes are also referred to as (discrete) belief states. The resulting plan is in fact a Direct Acyclic Graph (DAG), each path representing a trajectory. Conditional plans consider all possible trajectories thereby guaranteeing that the goal is reached. Several planning systems model non-determinism via the use of sensing actions only and observables.

Probabilistic Planning

In the case of *stochastic* operators effects, *probabilistic planning* is used [BKS08, BG01]. Probabilistic planning is a *general approach to planning with uncertainty* that attempts to solve the stochastic shortest-path by specifying action policies that minimize the cost of plan execution. Unlike conditional planning however, minimization of the plan cost also depends on the probability of the actions' outcome. In other words probabilistic planning attempts to generate a policy that maximizes plan success depending on the probability of the actions' effects.

Probabilistic planning under *partial observability* is also possible. In this case not all state values are visible. Actions may now non-deterministically change both state and state observability. The planning objective is therefore to find a policy that increases the chances of success by: a) avoiding actions with high cost and b) opting for actions that increase state's observability thereby allowing for more informed choices. Note that unlike the case of full observability, we now have a notion of probabilistic belief state, which is a probability distribution over all of the states. When an action takes place, the system does not transition between state, but between sets of states. Each state is associated with probability that reflects the degree to which it is believed it reflects the true system.

[10] Also referred to as contingency planning.

The cases where deterministic actions occur in contexts where either partial or no observability is possible, is not contemplated. This is because plans using deterministic actions do not need not consider alternate execution paths due to non-deterministic outcomes or unobservable state.

Planning Complexity

In order to select an appropriate planning system to support the automated acquisition, composition and execution of workflows, it is necessary we not only understand the type of planning problems that can be solved, but also be aware of the difficulty that solving such problems poses. In this very brief discussion we will provide some general information on planning decidability and (worst-case) computational complexity of classical planning. Some comments will also be made in relation to non-classical planning.

Planning problems complexity falls into two categories: a) verifying that a plan exists (*plan-existence*) and and identifying (or generating) solutions that are bound by a maximum number of tasks (*plan length*). All of classical planning is decidable for both plan existence and plan length. This is true whether classical, set-theoretic or state-variable representations are used to encode the planning problem [Tra04]. However, if function symbols are used, then plan-existence is semidecidable[11]. Note that for obvious reasons plan generation is generally of greater importance, however the ability to verify the existence of a plan does permit interesting solutions (for example sound and incremental planner for a given class of problems [JB98]). In our case, tractable plan-existence is a crucial requirement that makes the automated acquisition of planning operators feasible. It stands to reason then that the planning representation used through this work must be function free.

The planning complexity of unrestricted problems for both plan-existence and plan-length is very high (EXPSPACE-complete and NEXPTIME-complete). EXPSPACE is the set of decision problems that can be solved by a deterministic Turing

[11] A planning procedure is semidecidable if: a) it always terminates and returns an answer **yes** if a plan exists; b) never returns **yes** if no plan exists; and c) is not guaranteed to terminate if a plan exists. [Tra04]

machine using $O(2^p(n))$ space, where $p(n)$ is a polynomial function of n ([Tra04]). NEXPTIME represents the set of all problems that are solved by non-deterministic Turing machine in $O(2^{p(n)})$ time, where $p(n)$ is a polynomial function of n ([Tra04]). Several restrictions, such as disallowing the use of negative pre-conditions[12], disallowing the use of negative effects, disallowing the use of functional symbols and fixing the operators instances prior to planning, may be used to reduce this complexity. In such cases planning complexity may vary from constant time to EXPSPACE-complete [Tra04,ENS95]. However, save for the use of functional symbols, employing these restrictions in the general context of workflow planning is not possible.

The classical and state-variable representations have greater expressiveness than that of the set-theoretic representation in the sense that they allow for a more direct and succinct description of the planning domain and problem. Nevertheless, all representations can be translated from one form into another. The study in complexity shows that the unrestricted use of the set-theoretic representation of classical planning has the lowest complexity (both plan existence and plan length are PSPACE-complete). We must nevertheless be aware that the set-theoretic representation may potentially take exponentially more space than the equivalent classical and state-variable encodings (in [Tra04], page 48). It may therefore be advantageous to use a planning algorithm based on the set-theoretic representation, although this is not guaranteed.

An interesting result is that conditional effects used to describe the non-deterministic outcome of actions, do not influence planning complexity [Tra04, ENS95]. This is significant because workflow planning needs to deal with uncertainty. It is important to note that the results of planning complexity presented here are for a worst-case scenario. Many practical planning problems have lower complexity. The modelling of uncertainty nevertheless ensures that planning complexity remains high, which explains why initial work (such as [PS92]) in this area did not present experimental results until the introduction of Graphplan [WAS98]. Conditional planning still remains a challenge. Recent planners have therefore resorted to the use of heuris-

[12]This restriction is dominated by the operator's negative effects and is therefore of limited use.

tics [DBS06] and symbolic model-checking [BCP+01] to deal with the very large search space.

Probabilistic planning has unmatched expressiveness when it comes to defining goals as competing criteria (see [BKS08, BG01] for examples of probabilistic planners). It allows for the generation of optimal policies based on several criteria, which is otherwise difficult to do in classical planning. The complexity results for unrestricted[13] probabilistic planning with full observability are essentially the same as those for classical planning (EXPSPACE-complete, see [LGM98][14]). However, in the case of probabilistic planning with partial observability, the state space is infinite (in [Tra04], page 395). This is because state transitions are made between stochastic belief states that are defined by a continuous probability distribution. Even though we are unaware of a formal analysis of its complexity we assume this is the highest. Probabilistic planning with partial observability therefore presents the greatest planning challenge of all.

2.3.3 Workflow Planning

Several examples that use AI planning in order to support flexible workflow exist [RMBCO07,RMKM00,RMBM00b,RMBM00a,RMBM01,RMK01,RMK02,RMOB+04, RMBO+05,MB99a,BD99,BD00,MB99b,AGF06,GDB+04,ESBPDWJHDN04,BJ03, LADM07, MB99a]. Much of the work in automated process composition has been done in the specific area of scientific workflow. Additional work, related to workflow management, has also been done. According to this research, the automated synthesis of workflow process definitions have several advantages: increased efficiency (reduces the amount of effort used in modelling processes), ease of use (many of the details are now taken care of automatically via inference) and greater adaptability (replanning for example can be used to correct or adapt processes according to changes) [MLR06]. However none of this research presents an analysis of the re-

[13]Restrictions include bounding plan length, disallowing cycles and allowing optimal, average or pessimistic partial ordering.

[14]Some results are also presented for the case of restricted stochastic planning with partially observability.

quirements for automated workflow planning, nor is the selection and use of a given planning system justified.

The aim of this section is to identify the modelling requirements for workflow planning and to use this information to select an adequate planning system. This selection considers both functional criteria (such as type of problem that can be solved (see Section 2.3.2)) and non-functional criteria (such as the planning algorithm's completeness).

In their essence workflow process present task-based operational plans. First and foremost these plans need to be robust in order allow for their successful completion. To do this, several basic routing (or flow-control) constructs are used in the modelling and execution of the process instances. These constructs allow for the selection and ordering of task execution according to the observed state. Basic routing constructs include conditional selection and iteration. This means that workflow planning is clearly a *non-deterministic* AI planning problem. In other words conformant, conditional, or probabilistic planning may provide the necessary problem solving capabilities.

The objective of an operational plan is, in general, to reach a set of goals as efficiently as possible. In other words both the number of tasks and the selected task's execution costs should be minimized (for example by reducing resource consumption)[15]. Conformant planning enforces the execution of possibly needless actions in order to attain a desired set of goals because it assumes no observability. In addition to this, conformant planning may yield no solution in cases where the use of partial or total observability would (see Section 2.3.2). We may therefore conclude that conformant planning is not appropriate for workflow planning.

In order to use *probabilistic* planning it is necessary to have statistical data regarding the non-deterministic outcome of actions (see Section 2.3.2). If we consider the case of probabilistic planning with *partial observability* then statistical data regarding the observations must also be used. In addition to this we need to include the costs related to actions (or alternatively, and more problematically, the rewards

[15]Additional criteria such as process make-span can also be considered.

associated with plan states). This information is hard to come by. In fact the aim of this work is to identify the tasks themselves, so we do not even know what the outcomes are, let alone their probability distribution. Using all this information to model a planning problem correctly is difficult. In fact modelling planning domains in a probabilistic setting is generally recognized as a difficult undertaking (in [Tra04], page 399). If we consider that solving probabilistic planning problems, specially in the case of partial observability where the state space is infinite, is computationally a hard problem, then we can conclude that probabilistic planning is not a promising solution.

We are left with *conditional* planning as a means to automatically compose a workflow process. As was mentioned before we need to support loops (cycles) in order to deal with uncertainty. Plans that allow for iterative, repetitive behaviour can successfully deal with short-term errors during plan execution (for example communications errors that occur during the execution of data gathering web-services). Planning must also ensure that any such iterative behaviour also contribute to attaining the goal. Needless dead-ends must not be allowed (for example after successfully executing a data gathering task we should not repeat it again). Although conditional planning allows for task selection, it cannot generate plans with loops. This is because conditional planning guarantees success under all circumstances (see details in Section 2.3.2). We are therefore forced to conclude that conditional planning is not a viable solution.

An alternative planning system, based on symbolic model-checking (known as the Model Based Planner (MBP)[16] planner) is available. It is based on a general framework that provides supports for different classes of discrete, non-deterministic planning problems [BCP+01]. It can deal with uncertainty in the actions' effects and any of the planning states. It supports planning under full, partial and null observability. It also allows for the use of extended goals based on temporal conditions (Computational Tree Logic - CTL). Depending on the planning problem it can generate sequential plans, conditional plans, plans that use an iterative trial-and-error strategy and

[16] http://sra.itc.it/tools/mbp/

plans whose execution depends on its execution history [CPRT03, BCRT06, CRT98]. We are therefore interested in its ability to generate *iterative conditional plans* that repeatedly sense the world (non-determinism with full observability). These plans are state-action tables (partial functions that map actions to reachable states), which also facilitates process monitoring and execution (simple interpretation of a finite state machine is enough to implement a controller). In addition to this, these plans may be *weak* (at least one execution path will reach the goal states), *strong* (all execution paths are guaranteed to reach the goal states) or *strong cyclic* (an iterative trial an error strategy ensures that all execution paths are guaranteed to reach the goal states). We therefore opt for the use of the MBP planner in order to generate *strong cyclic plans* under full observability (possibly using temporal extended goals).

2.3.4 Plan Learning Scenario

In order to better grasp the issues that have an impact on the automated learning of planning, we will first describe the scenario where such learning will be applied. This allows us to: a) identify the basic requirements for Machine Learning; b) provide a basis for comparison with existing work in this research area; and c) determine the set of goals and assumptions that can be made in order to successfully support the automated composition of workflow processes.

The general setting consists of a group of agents interacting with each other in a shared environment. Agent is a generic term that is used here to represent people or information systems that consume and/or generate data. The shared environment represents a common medium by which agents interact. It is also the repository that holds the information that is generated and consumed by these agents. We will assume that agents may concurrently access this medium in order to read and write data. The read and write operations are atomic, so agents may obtain a consistent state of the world. The agents actions are taken in accordance with this consistent view of the world state.

As far as the data format and information semantics are concerned, we will also make several additional simplifying assumptions. First we assume all data is exchanged in a common format that is known to all agents. Second we will work

under the premise that all agents share a common domain ontology. In other words only one interpretation is possible. This allows agents to classify objects according to a class hierarchy, exchange information and understand how to express goals using a single vocabulary. We also admit that agents just exchange extensional information (facts). Such data will be expressed as fully ground predicate in our own representation. Note that these assumptions do not, in any way, restrict the intentional information represented by the learning and planning system.

The agents that interact with each other have different levels of knowledge. Some of these agents may be experts and therefore know how to attain a given set of goals via the execution of a sequence of actions. Others are not as proficient and may therefore attempt to execute invalid, erroneous and inefficient sequences of actions. The difference between these agents is their knowledge base of actions. We work under the important premises that: a) these agents are not adversarial; on the contrary they have a vested interest in collaborating[17]; b) all tasks that are executed belong to a domain of actions that adequately described the common domain ontology, which is shared by all agents and the planning and learning systems c) their is no way we can relate a set of actions to a given plan execution (after all some of these agents might not have the necessary knowledge to plan correctly); and d) goals cannot be associated with actions or plans (we assume that novice agents do not have the necessary knowledge to correctly express actions nor compose plans[18]).

Recall that the objective of this work is to support the automated management of ad-hoc workflow management. In this setting two types of actions exist: actions that are planned by the workflow management system and actions that are pro-actively executed by the agents. The first case represents tasks that have been sequenced by the AI planning system in response to an agent's request to achieve a set of goals. The second case represents actions that are executed because a previously planned process failed to complete correctly or simply because a new and previously unknown action needs to be executed.

[17]Strictly speaking, even in environments where agents supposedly do collaborate, this may not always be true [Jør04].

[18]Here we assume that novice agents may experimentally execute actions.

Several assumptions regarding the execution of these actions are also made. First of all the execution of the actions are signalled via an action execution signature that contains information on its name, parameters and the states in which it initiates and terminates execution. This data is made available to the Machine Learning algorithm and plan controller. The action signature may not contain all of the above information, for example its name and/or parameters may not be known. Second, the execution of the actions need not be instantaneous, however it is guaranteed that: a) during an action's execution none of its pre-conditions are altered by any concurrently executing action; b) at the end of the action's execution all effects are guaranteed to be true. Third, actions will only initiate execution if all their pre-conditions are valid and if it does not alter other concurrently executing actions pre or post-conditions[19]. Finally action execution fails when the expected outcome is not observed. This may occur when actions are pro-actively executed by the agents which inadvertently alter the world state or when unknown (for example non-deterministic) outcomes of an action occur. This set of restrictions allows us to use of classical planning algorithms, although errors caused by the execution of *new actions* must be dealt with.

The aim of this work is to non-obstructively observe the interaction between agents and identify what actions take place and the conditions under which this occurs. These actions are then encoded as planning operators, which can later be used to compose sequences of actions. These plans are generated on the behalf of agents requests. New actions or alterations to existing actions are learned when execution errors occur. These changes are incorporated into the planning system, thereby allowing for the dissemination of information amongst agents. Because agents may attempt the execution of new unknown and possibly invalid actions or because of sensing errors, the learning algorithm must be able to deal with noise.

In this work we are not concerned with process optimization. In other words, the objective is to discover and use planning operators that emulate the agents actions. No care will be taken in identifying or re-engineering optimal operators. In addition

[19]Note that we assume here that the concurrent action may belong to other plans in simultaneous execution.

to this, AI planning used here will not deal with issues regarding resource allocation or task scheduling.

2.3.5 Learning Operator Definitions

In this section we review research work in AI planning that uses Machine Learning (ML). We are specifically interested in work that uses ML to identify and refine operator schema so that these may subsequently be used by a planner. In order to facilitate the description and classification of this work, a set of characteristics, which are described in Table 2.2, will be used.

Table 2.2: Characterisation of research in learning AI planning models.

Sampling	
Characteristic	Values
Observability	full, partial
Causal Relationship	image, pre.lag, post.lag.
Trace	plan, act.
Noise	yes, no
Concurrency	yes, no
Actions	
Characteristic	Values
Action Duration	inst., durative
Action Effects	det., cond., stoch., cond.stoch., cond.non-det.
Disjoint Effects	yes, no
Action Pre-condition	conj., disj., 1.lit., $^{-}$conj., $^{-}$disj.
Operator	STRIPS, rule, quant.
Representation	prop., rel.
Background Knowledge	yes, no
Learning	
Characteristic	Values
Incremental	yes, no
Examples	$\oplus$, $\ominus$, $\odot$
Induction	$\uparrow$, $\downarrow$, $\uparrow^{\downarrow}$
Integrated	yes, no
Evaluate Planning	yes, no

Sampling refers to the set of features related to the data used for learning. This data concerns the examples from which operator definitions are acquired. The first of these features is the *observability* of the world-state, which may be *full* or *partial*. Partial observability means that when an action is taken not all of the world-state

can be sensed. The notion of *causal relationship* refers to knowledge of an action's cause and effects. Knowledge of the causal relationship is complete when both an action's pre-image and post-image are known (simply referred to as *image*). If a set of candidate pre-images exist, then we say that we have a state where the true pre-image lags (*pre.lag*). In this case we assume that no post-image is known. If however the pre-image is known but a set of candidate post-images exist, then we say that we have a state where the true post-image lags (*post.lag*). Note that the last two cases naturally attempt to describe durative actions. The data *trace* refers to the context under which the samples are collected. In the case of a *plan*, the samples not only include the actions' signatures (action name and parameter list) but also information on the plan's pre-image (initial state), intended goals and its action's ordering. Alternatively, only the actions' signatures (*act.*) may be known. The sampled data may or may not contain *noise*. Several sources of noise exist such as extraneous actions and faulty sensors. Lastly, the sampled data may be used to collect information on a single action or various actions that are taking place simultaneously (*concurrency*).

The *actions* sub-group characterises the actions' execution and corresponding operator definition. *Action duration* refer to whether or not operators model instantaneous (*inst.*) or *durative* actions. Durative actions may execute over a period of time. Their pre-conditions may not all be satisfied at the same instance in time nor their effects take place simultaneously. The *action's effects* may be deterministic (*det.*), conditional (*cond.*), stochastic (non-deterministic) (*stoch.*) or a composition of conditional and non-deterministic effects. The conditional operators' non-deterministic effects may be modelled as a set of either stochastic (*cond.stoch.*) or discrete (*cond.non-det.*) outcomes. When we presuppose the concurrent execution of actions, we may or may not make the simplifying assumption that these action's effects overlap (*disjoint effects*). An *action's pre-conditions* may be as simple as a single positive literal (*1.lit.*), a conjunction of positive literals (*conj.*), or a disjunction of positive conjuncts (*disj.*). When either positive or negative literals are used, we indicate this with explicitly ($^{-}$*conj.,* $^{-}$*disj.*). An *operator* may be modelled via an action schema using *STRIPS*-like notation (set of pre-condition, and add and

delete lists) or may consist of a set of planning *rules*. The operators using *STRIPS*-like notation may also include quantification (*quant.*). It is possible to translate between either form. The operators, whether expressed via STRIPS-like schema or rules, may be *represented* in propositional (*prop.*) or predicate logic (*rel.*). The ML algorithm may or may not use *background knowledge* in order to infer additional relationships. This background knowledge may be both extensional and intentional in nature.

Learning refers to those key elements that distinguish current ML algorithms and systems used in acquiring planning operators. Several of these ML algorithms can continuously and iteratively refine operator definitions. In this case we say that they are *incremental*. The data samples constitute the positive ($\oplus$) and negative examples ($\ominus$) used by the ML algorithms. Most algorithms require only positive examples or a combination of both positive and negative examples ($\odot$). The majority of the algorithms are based on induction. Induction may be based on the generalization ($\uparrow$) or specialization ($\downarrow$) of the examples. A combination of these two strategies is also possible (for example version space learning, $\uparrow^{\downarrow}$). Many of the existing systems *integrate* learning, planning and execution in a single system. The corresponding algorithms are interleaved in order to generate and process action execution examples. The ML algorithm must then be assessed. This can be done for example by: evaluating operator classification quality or using empirical measures to measure operator error. Research may also realistically evaluates planning operators via AI planning (*evaluate planning*).

The ARMS System

One of the most recent work on learning action descriptions is presented in [YWJ07]. In this work a ML algorithm (ARMS - Action-Relation Modelling System) acquires operator schema from a set of successful plan execution traces (positive examples only). These traces also include the plan's initial state and goals. This data is modelled as an initial action's post-conditions and final action's pre-conditions respectively. Although not required, ARMS may also take advantage of state-images (called intermediate states) that may be obtained during plan execution. The sam-

pled data, which basically consists of sequences of action signatures and the domain description (non-ground predicates with possibly typed parameters), are then used to construct a set of action (existence of supporting pre-conditions and supported post-conditions), information (initial and intermediate states supporting actions, supported goal and intermediate states) and plan constraints (supporting preconditions within a sequence of actions). In addition to this, a frequent-set mining algorithm is used to obtain a frequent set of action-relation pairs. The constraints (propositional formulae representing relationships) and frequent set action-relations pairs (weights of relationships) are then encoded as a weighted maximum satisfiability problem (weighted MAX SAT). The solution of this satisfiability problem results in an action-model that describes the action and its most frequent (pre and post-condition) relations. The satisfiability problem is parametrised according to a probability threshold (which determines when a relation-pairs is considered frequent) and an upper bound[20] on the number of relationships in the pre-conditions and post-conditions of each action (this actually determines when a action schema is learned or not). It is important to note that this work did not evaluate plan generation quality but used two quality measures (error and redundancy rates) to evaluate its experimental results. The authors have considered extending this work to include, amongst other things: accepting partial and possibly incorrect operator schema, dealing with noise in the traces, evaluating plan generation quality, extending support for operator schema that include quantification for example, learning HTN schema and evaluating how best to determine the upper bound on the number of relationships in action schema.

Recently this work has been extended to learn operators from an environment wherein various agents can execute actions [ZnAY11]. The Lammas ML algorithm (Learning Action Models for Multi-Agent Systems) learns MA-STRIPS operators, which are standard STRIP-like operators whose action parameters are augmented with the indication of which parameter represents the agent that executes the action.

[20]This value can be obtained empirically by solving the problem for various bounds. For experimental purposes in the article this value was directly obtained from the planning domains that were tested.

It uses this information in order to establish a set of additional constraints that encodes the possible interactions amongst agents. These constraints expand on the already existing ARMS action constraints. The evaluation criteria used counts the number of incorrect pre-condition, add and delete literals in the operators. Both false positives and false negatives are counted. Equal weight is given the error counts of the pre-conditions add and delete errors. An error rate measures the total number of counts for all actions and agents. The effectiveness of Lammas is shown by comparing its performance with ARMS. Essentially the learning algorithm, which is based on weighted MAX-SAT, remains unchanged.

The LAMP System

The LAMP algorithm (Learning Action Models from Plan Traces) attempts to learn STRIP-like operators that use both quantification and logical implications. It is very similar to the ARMS and Lamma ML algorithm in that it also attempts to learn action models based on action traces that contain action labels, the initial state, final state and possibly empty intermediate states (subsets of world states) [ZYHL10]. As with those systems, LAMP first encodes each plan trace as a set of propositions, generates a set of candidate formulae according to a number of constraints and learns the weights of these formulae. However there are two major differences. First it extends the set of constraints of the ARMS algorithm (action consistency, plan consistency and non-empty constraints) with additional pre-conditions, positive and negative effects, and positive and negative conditional effects restrictions. All possibilities of groundings of these new constraints are enumerated. All potential quantifications of the enumerated formula are then generated. The second difference is that the weights of the formula are learned using Markov Logic Networks and not weighted MAX SAT. Once the set of formula that maximize the weights is obtained, they are combined to form the final operator definitions using the Planning Domain Definition language (PDDL). The evaluation criteria is also based on the number of missing literals. However in this case the counts are more sophisticates because now an operator may not only have an incorrect (sub-)set of propositions but the quantifications learned may also be wrong. A comparison based on the

grounding of the formulae is used. Additionally the errors regarding add and delete propositions are not computed separately but as a single effects error. The pre and post-conditions errors are given equal weight. An error rate base on the average error counts of all actions is used to evaluate the experiments. Unlike many of the other research efforts the authors of LAMP experimentally show that the algorithm can successfully help humans in reducing the modelling time and error.

The SLAFS Algorithm

A ML technique called Simultaneous Learning and Filtering of Schemas (SLAFS) is used in [SA06, SCA06, AC08, Ami04] to generate *exact* actions schema. In other words, unlike the ARMS system [YWJ07], all action schema reflect every sample that has been observed. It cannot therefore deal with noise. The trace is based on actions, so no notion of plan is available. The sampled data consists of a set of state images and action signatures. In addition to this the system can also use information indicating whether or not the action has been successful applied (the assumption being that any failed action will not generate any effects). Unlike [YWJ07], this work is able to produce schema for conditional operators in the form $G \xrightarrow{a} f$, where $a \in A$ is an action, G a set of formulae (conjuncts) that represent the action's pre-conditions and f represents a fluent that is altered by a. The actions' effects (or fluents) are limited to a set of literals (not formulae). This means that some post-processing may be required to allow for the use of these operators in automated planning. In addition to this, the ML algorithm explicitly caters for partial observations thereby yielding operators that may establish relationships between actions and unobservable effects. The SLAFS works by establishing all possible relationship between every fluent and action via a set of logical formula. These formulae are referred to as explanations (one for each fluent). Whenever a sample is processed (a new explanation is obtained), any of the existing formula that are made inconsistent, are removed (filtered) from the knowledge base. Unlike many other solution, this means that the ML algorithm is incremental. Additional constraints are used to ensure consistency between actions, pre-conditions and effects: $\bigwedge_{a,F,G} \neg(G \xrightarrow{a} F) \wedge (G \xrightarrow{a} \neg F)$ (no action results in incompatible effects) and $\bigwedge_{a,F,G \to G'} \neg(G \xrightarrow{a} G') \to (G \xrightarrow{a} \neg F)$

(the set of more specific operator pre-conditions are always used). The knowledge base is therefore a set of consistent formulae. The explanations are maintained in a compact DAG that permits the efficient addition of new formulae and reuse of existing (sub)formulae. The action schema is a valid assignment of the model (set of formulae). It is obtained via the use of a Boolean Satisfiability Problem (SAT) solver. Note that this ML requires that all samples be consistent, which explains why it cannot handle noise. The result is a set of propositional rules whose pre-conditions consist of possibly several conjuncts. The fluents are limited to a single literal however. Because several rules may refer to the same action, these rule can effectively represent (ground) STRIPS operators with conditional effects and disjunct pre-conditions.

The OBSERVER System

One of the earlier research efforts in the automated acquisition of domain knowledge in AI Planning which did not require prior planning operator knowledge, is [Wan95, Wan96b, Wan96a, WC94]. It is based on the *learning-by-doing* paradigm and combines interleaved AI planning, plan execution and learning within an integrated learning system called OBSERVER. This framework assumes that a basic set of predicates that describe object types and relationships are available. Its' traces consists of a sequence of action signatures and their respective state-image (pre and post-images), which are referred to as the expert solution traces. Most importantly, and unlike [YWJ07] and [SA06], it also uses a number of (random) practice problems, that allow it to incrementally refine its operator definitions when planning. It assumes the full error-free observability of deterministic actions only. Although it is limited to learning STRIPS-like operators with conjunctive pre-conditions, it is still able to learn negated pre-conditions and conditional effects.[21] In addition to this, it also assumes that when action execution fails it is due to unsatisfied pre-conditions. In such cases no effects are observed.

Learning and planning uses two simultaneous representations of the planning

[21] Note however that is does not support conditional planning. It uses the non-linear deterministic planning system PRODIGY (http://www.cs.cmu.edu/~prodigy/).

operator *Op*: a most specific *S(Op)* and most general representation *G(Op)*. Learning consist of three algorithms that infer the operator's most specific pre-conditions, most general pre-conditions and effects respectively. The first algorithm generalises the pre-conditions of the most specific operator definition based on the successive observation of successful action traces. If however an action trace fails then it assumes that this is due to missing negated pre-conditions, which are then added to the *S(Op)*'s pre-conditions[22]. The initial version of $S(Op)$ consists of all of ground predicates in the pre-image that are related to action. The second algorithm determines the *necessary* and *critical pre-conditions* of *G(Op)*, which is initially empty. These pre-conditions are obtained when action execution either fails due to near misses (*necessary* pre-condition is a single condition in S(Op) that is not satisfied in the pre-image) or succeeds in very similar conditions (*critical* pre-conditions is a single condition in S(Op) that is satisfied in one pre-image but not another). Finally, the effects of the operator are learned by generalising the *delta-states*, which represent elements that are added to the post-image or removed from the pre-image. In this algorithm, conditional effects are also detected and further refined using the two previously described algorithms.

Note that the learning algorithms initially use a set of expert traces to obtain an initial version of the operators. Practice problems are then used to refine these operators. The author reports that learning during problem solving is crucial in order to allow for the generation of effective plans. So in addition to learning, the issues related to planning with incomplete and inconsistent operators is also tackled [Wan96b]. The planning module first attempts to solve the planning problem using the *G(Op)*, thereby increasing the chance for planning success. If planning is possible, plan execution is attempted. During successful (state change is observed) plan execution the operator's *S(Op)* and effects are learned. If plan execution fails (no state change is observed), failed plan traces provide opportunity for learning the *G(Op)*. Note that in the case of plan execution failure, a attempt is first made to solve any pending goals. If this is possible plan execution and learning continues,

[22]Any state $\{p \mid p \in \text{pre-image}(i) \wedge p \notin S(Op)\}$ is added so that $S(Op) = S(Op) \cup \neg p$ where pre-image(i) is the state prior to action's execution failure.

otherwise full replanning is attempted. If this is successful then plan execution and learning continues. If planning is not possible, the system falls back on expert traces.

The Three Level Learning Algorithm

A recent effort to learn probabilistic planning rules is presented in [PZK04]. This work is interesting for two reasons. First it shows that it is possible to learn operators with non-deterministic, albeit stochastic, effects. Second, it demonstrates how induction can be used to efficiently learn these (stochastic) relation planning rules, as opposed to propositional planning rules as in the case of the SLAFS algorithm [SA06]. This work assumes full observability of instantaneous actions. Only one action can take place at any point in time. The data samples consists of an action's signature and its state-image. No concept of plan is used. In addition to this all examples are covered by the induced rules, so this work cannot deal with sampling noise. From the description provided it seems that only positive examples are required to learning the planning rules (no information on action execution failure is used). Although part of the ML algorithm is based on ILP, it is not evident that a rich set of intentional background knowledge based on Horn clauses is supported.

The learning algorithm consists of three phases wherein each performs a greedy search. The first phase (*learn rules*) is based on ILP and searches the hypothesis space in order to determine the action's pre-conditions. In the article, both generalisation and specialisation are described, however it is unclear if both types of searches are performed simultaneously and if so, how. Nevertheless it is clear that the initial rule set consists of a number of the most specific rules that are necessary to cover all state-action pair examples. Whenever a new hypothesis is generated by *learn rules*, a second phase (*induce rules*) is initiated in order to determine the action's outcomes. During this phase, the algorithm first determines the change in state (equivalent to delta-state in [Wan95]) in order to identify the action's effects (possible outcomes). It then tries to generalise these effects in order to ensure that: a) all examples are consistently covered; and b) no effects overlap (i.e.: all effects are mutually exclusive). The final phase (*learn parameters*) is used to determine the probability that must be assigned to each of the rules' possible outcomes. There

is no closed-form expression to do this, so a conditional gradient method is used to determine these probabilities. As with standard ILP, a bias is also used to rank and select which hypothesis are further refined. This bias ensures that the algorithm favours the simplest (Occam's razor) rules that assign high likelihood to the data. The scoring metric that represents the bias depends on a weighing factor that is empirically determined. The second phase also uses a similar scoring metric which is maximised. Future work aims at providing an incremental learning algorithm and allowing for the induction of planning rules based on partial observations.

The MSDD Algorithm based Learner

Another solution that can generate stochastic planning rules in presented in [OC96b, OC96a]. It is similar to [PZK04] in that it generates stochastic planning rules. It also assumes full observability of state and the execution of a single action at any point in time. No notion of plan is available. However this work differs markedly from the previous ones in that its actions are not instantaneous (scenario consists a robot that must execute various tasks wherein sensor readings are sampled). An action's causal relationship is determined by a single pre-image and a set of post-images that represent world states sampled at different instances after an action has been initiated (history of multitokens). In other words, an action's effects may be delayed (this lag however is limited to a fixed value, in this case to a single sample). The action signature is also simpler in that it consists of an action name only. However, the sample consists of a set of all propositions (tokens). This set of tokens (multitoken) reflects all of the robot's sensors' categorical values at any given time. Multitokens are not only used to represent the pre-image and a set of post-images, but also indicate the action taken. The planning rules generated here also differ from those of [PZK04]. In the case of [PZK04], each planning rule represents a single action. Such an action contains a single set of conjuncts that represent an action's pre-conditions. Each of these rules however contains one or more outcomes with their corresponding probability of occurrence. All rules are presented using literals consisting of predicates (relations). In this case however, an action may be represented by one or more rules. Each rule contains two set of

propositional conjuncts (the pre-conditions and post-conditions respectively) and the action name.

Automated learning is based on the Multi-Stream Dependency Detection (MSDD) algorithm. The input of this algorithm is a sequence of streams (history of multi-tokens). It determines frequent co-occurrences of propositions (subset of multi-tokens) that occur between fixed intervals of time (experiments limited to a single sample). MSDD performs a general to specific best-first search. Initially all propositions (tokens) are assigned a wild card. Two multi-tokens, representing the pre-conditions and effects, have wild cards assigned to all their tokens (wild cards represent "don't care" values). Whenever the systematic search identifies a correlation, the corresponding token is assigned a value (specialised). The first token represents an action. As with the other tokens, when a dependency is detected, an action token is assigned an action name. During the search a tree (trie-like structure) is generated. All nodes in this tree are assigned a probability of occurrence. Any such node with a action name assigned to the first token represents a planning operator. The algorithm is parametrised so that one may set a threshold to determine frequent occurrences. It can therefore deal with noise. In addition to this, statistical measures are also used to determine if the probabilistic effects that occurs in a more specific context are significantly different or not. If not the algorithm retains the more general definition. The MSDD algorithm does not take advantage of any background knowledge.

Learning Teleo-Reactive Operators

Another research effort in the general area of robotics is presented in [Ben95a, Ben95b]. As with the [OC96b], actions are durative. However, the emphasis here is in providing reactive agents with a means to continually observe and respond to changes in the environment. Like [Wan95], the TRAIL (Teleo-Reactive Agent with Inductive Learning) system integrates and interleaves planning, execution and learning. Planning is based on a plan structure known as a teleo-reactive tree. Each node in the tree represents a teleo-reactive operator that is linked to a parent teleo-reactive operator. Each operator can only be executed according to a set

pre-conditions (pre-image) and supports the execution of the parent operator by altering the world state (one deterministic effect and several stochastic outcomes known as side-effects). The root of the tree contains the goal of the plan. Paths is the teleo-reactive trees are akin to plans generated by regression based planners. Each path represents a plan starting at a given initial state. Execution therefore starts by analysing the lower teleo-reactive operators of a given tree and executing one of those whose pre-image is satisfied by the current world-state. This process is repeated until either the goal is reached, planning fails or execution fails. During execution both positive and negative (when a failure is detected) examples are collected. During planning failure, the system requests examples from an oracle and records those as positive examples. These examples are then used by the learning module in order to generate new operators or refine existing operators. This work assumes full observability of a single action. As with [OC96b], the causal relationship is determined solely by the pre-image. However the traces consist of a sequence of several pre-image only. This is because the learning algorithm only learns operator pre-conditions.

Whenever a failure is detected the system first attempts to execute some simple experiments (repeat the action for at least the mean observed time of previous operator executions) in order to further diagnose the problem (detect possible side-effects). If the error persists, then a standard ILP system (GOLEM [MF90]), is provided with the execution examples of a single specific operator in order to learn its pre-conditions. GOLEM is a bottom-up inductive logic programming system that generates a determinate[23] least general generalisation of a single operator. It then generalises this operator in order to cover all positive, but no negative, examples. Note that GOLEM does not generate the least general generalisation relative to the background knowledge, so no background knowledge is used (more efficient). GOLEM is also parametrised in order to handle noise in the examples. Because the GOLEM does a systematic search, it may be used to incrementally refine existing

[23]This means that all pre-condition literals' variables must be either directly or indirectly bound to a variable in the operator's parameter list. All literals variables are bound if at least one if its variables are bound.

operator definitions.

The WISER System

Unlike the work that has been surveyed until now, [TCH99] identifies and attempts to improve on the results presented in previous research [OC96b, Wan95]. The focus is on improving the efficiency of the ML algorithm and the correctness of the operator definitions used in classical planning via experimentation. It assumes that the world is fully observable and deterministic. It assumes that noise may be present in the samples. As with [OC96b, Wan95] the traces are based on the actions' execution signature and pre and post-images. Like [Wan95] the post-image is processed as a delta list and the operators' negated pre-conditions are also learned. The world is represented via predicate logic, but as in the case of [OC96b] a three valued logic (true, false and do not care) in order to represent a state value. Unlike [OC96b] however, the "don't care value" is implicitly managed by first explicitly expressing and learning operators using both true and false values of a literal. If neither are applicable, then the ML algorithm implicitly assumes a "don't care value".

WISER's algorithm is split into three phases: a) learning the initial set of pre-condition of an operator, b) refining the operators pre-conditions and c) acquiring the operator's effects. The initial pre-conditions of an operator are learned using a search that generates all possible transition states (pre and delta lists) much like [OC96b]. It such state constitutes an experiment to be evaluated based on the examples. Note that unlike [Wan95, Ben95a] these experiments are not generate by a planner. Unlike [OC96b], which used all sensor values during learning, states are iteratively and systematically increased in order to reduce the search space of the following phases. To increase efficiency further, several other improvements where made. First the *naive domain* assumption is used. It presumes that the modelled world is regular in that any set of pre-conditions learned from a set of objects with a given type signature will be the same for any other set of object with an equivalent type signature. This allows sampling of action signatures to be directly proportional to the types of the objects and not the number of objects themselves. Heuristics are also used to order the literals that are added to the explored state space. These

heuristics are based on user feedback or ordering literals using the *highest-frequency-first heuristic* (here it is assumed that literals that are important for one operator are also important for another). This effectively changes a random search into a biased search.

The refinement phase of the operator's pre-conditions is done in four steps: a) generalising the operator's pre-conditions, b) acquiring negative pre-conditions, c) generalising object type and d) type specialisation of objects. The generalisation of pre-conditions is done by negating the existing literals and testing these against the examples. If the operator remains valid then these literals are removed. Negated pre-conditions allow for more robust operator definitions by detecting and making explicit several relationships (for example in the robot world (*arm-empty*) and $\neg$(*holding x*) may have the same meaning but may not be explicitly modelled). WISER uses its three valued logic to test the operator's pre-condition validity. If the negated literal is consistent with the examples then it is retained and effectively acquired. The ML algorithm assumes that a type hierarchy of classes is available. Whenever objects of the same class are encountered their types are generalised using simple disjunction (for example in the blocks world where *keys*, *boxes* and *cars* exist, the *pick-up* operator may have its parameters generalise to (*box* $\vee$ *keys*)). Lastly, type specialisation uses a information-theoretic system (C4.5 [Qui87]) to induce decision trees that classify objects according to additional criteria (for example box is *portable* if it is light and small). This criteria is used to further specialised the operator's parameters. Note however that both the background knowledge and an initial definition of the target class must be provided. The use of the induction algorithm allows the system to support partial observability and handle noise.

Similar to [Wan95], delta lists are used to acquire the operator's post-conditions. Several improvements are also made here. The first is related to the use of unbound variables in the operator definition. Unlike [Wan95], [TCH99] this algorithm does not assume that all the literals' variables, which represent the operator's effects, are bound to the operator's parameter. For example, in the blocks world we may define the *move(X,Y)* operator with two parameters X and Y that represent any object of type *portable*. A valid effect is then to set the place, where X was initially

placed, to clear (in other the operator should have a pre-condition *on(X,Y)* and the effects $\neg$*on(X,Z)* $\wedge$ *clear(Z)* where Z is not bound to the operator's parameter list). In addition to this, symmetrical relations such as *next-to(X,Y)* in robot world are dealt with by adopting human modelling conventions. In this case the parameter X is always assumed to be of the type *robot*. Finally, the use of *naive domain* does not allow acquiring the operator's effects correctly. This assumption is therefore not used in this learning phase.

The LOPE Integrated Architecture

Another research effort that integrates and interleaves learning, planning and execution is [GMB00]. Although this work uses reinforcement learning and its planning algorithm is not based on a (neo)classical planner, it does nevertheless generate propositional STRIPS-like operators. What is most interesting about this work is that it is one of the few research efforts able to generate planning operators based on traces from several concurrently executing agents. This work assumes partial observability of instantaneous actions. Samples (referred to as observations) include the actions' signature (name only) and pre and post-images. These samples are also assumed to be noisy. The underlying representation is propositional logic, but operator definitions are augmented with additional information in order to describe the stochastic outcomes of the actions and the utility values used for reinforcement learning. It is nevertheless assumed that each operator have only one set of effects.

A single ML algorithm is used to acquire both the operators' pre and post-conditions. It does this by checking if any previously recorded operator's conditions are similar to the new observation. If it is not, a new (most specific) operator is introduced, which includes the observation's pre and post-images. In addition to this a counter P containing the number of times the operator has been successfully applied (expected outcomes are satisfied) and a counter K containing the number of times the action was applied under the same pre-conditions (pre-conditions are satisfied) are initiated to one. If however an equal (and therefore similar) operator is encountered, then it is rewarded (P is incremented) and all other similar operators are penalised (K is incremented). If however no equal operator is not found, but

a similar one is available, then this observation is added as a new operator. Its P value is initiated to one. However its K is set to the same values as any other similar operator that was previously found. This ensures that all similar operators (operators that represent the same actions and share a common subset of the pre-conditions) will have the same K and that the sum of the ratio $\frac{P}{K}$ over all similar operators is one. All these similar operators are then penalised. At this point several heuristics are used to generalise all similar operators irrespective of the last observation's action (One of these heuristics is for example parametrised by setting a threshold. Any operators whose $\frac{P}{K}$ value is below this threshold is generalised further). As a result a new set of more general operators are obtained and the process is then repeated with any new observations that have yet to be processed. Because this algorithm successively generalises all operators it supports incremental learning.

Several additional characteristics of this learning framework should be pointed out. First, unlike [Wan95] and in a similar vein to [TCH99] and [Ben95a] it is assumed that actions may be executed in a given world-state even if not applicable. This means that failed actions may result in the generalisation of operators pre-conditions when these are not satisfied but a subset of its post-conditions are (Hayes-Roth inclusion heuristic). In addition to this and like [Wan95], [OC96b] and [TCH99] negated pre-conditions are learned (Hayes-Roth exclusion heuristic). Note however that both here and in [OC96b] negative post-conditions are also learned. Like [Wan95] and [Ben95a] a planner is used to generate a new sequences of actions and therefore promote the creation of new examples. Here however, a very simple stochastic planner is used to create experiments, in the same spirit as [TCH99]. In addition to this, learned operator may be shared amongst agents that concurrently in the same or different worlds (experiments where done using a simple robot world). The authors report that the number of successful plans that are generate, significantly increases with the sharing of operator definitions amongst agents.

The Two Phase Stochastic Local Search Algorithm

This research's assumptions and objectives most resemble the characteristics and requirements we have previously identified as necessary for supporting the automated composition and execution of workflows ([JV07]). This work is specifically aimed at acquiring planning operators in environments where actions are concurrently executed. It uses propositional logic to represent operators with conditional and non-deterministic effects. To the best of our knowledge it is the only work that generates operators with discrete non-deterministic effects. All other research efforts model non-deterministic operator effects as stochastic outcomes. As with [PZK04, SA06, OC96b], an operator is represented via planning rules. In this case one operator may have several planning rules. Each rule consists of one or more pre-conditions (a pre-condition is a conjunction of literals). Each pre-condition may be associated with a set of one or more effects (effects consist of a conjunction of literals). Non-deterministic outcomes are modelled by rules whose pre-conditions are associated with more than one effect.

Samples are assumed to be noise-free and consist of an action's signature and its pre and post-image. The authors of this paper point out that learning conditional non-deterministic operators based on samples of concurrently executing actions is equivalent to learning disjoint formula in Disjunctive Normal Form (DNF). The learnability of this problem has not been solved and they therefore resort to the use of heuristics. In addition to this, and contrary to [PZK04], in order to avoid the NP-hard (sub)problem of learning overlapping effects, they also assume that all concurrently executing actions have no overlapping effects. Unfortunately the basic premise of this work is that, not only is an initial version of the domain knowledge available, but it is also fairly complete and correct. The learning algorithm, which is inspired on the three phase learning algorithm of [PZK04], is therefore designed to refine existing operators with a minimum number of differences.

Very succinctly, the algorithm consists of two phases. The first phase identifies the operators' modification set for an action. A modification set is a set of literals that include all propositions that are used in the operators pre-conditions or effects. A second phase then uses the identified modification set and the positive and nega-

tive execution examples (which consist of pre and post-images) to obtain the actions' pre and post-conditions. During this second phase it assumes that: a) all positive examples are covered (no noise); b) the pre-conditions do not cover any negative examples; and c) no operator effects overlap. The first phase uses a generate and test cycle in order to produce the possible modification sets used by the second phase. The algorithm greedily searches for an assignment that allots all fluents to one or more actions in such a way that it minimises the Hamming distance between the new modification set and the modification set of the initial domain. In the second phase, the modification sets are used to specialise or generalise the operator, depending on whether a positive or negative example is processed. Note that specialisation of pre-conditions allows for insertion of negated literals. It is also used to ensure that no negative examples are covered and no overlapping of effects occurs. Care is also taken to split rules so that a negated pre-condition can discriminate between one effect and another. Generalisation on the other hand ensures that all positive examples are covered, which may result in a new most specific rule. Finally during this phase all operators are ranked in order to bias the search. Ranking selects those operators that are simplest (smallest rules) and penalises those rules with a greater number of non-deterministic effects. Experimental results seem to indicate that very large number of experiments are required in order to attain reduced error rates. Error rates were measured by counting the number of classification errors and are not based on planning success.

A summary of the research that has been reviewed is presented in the table below (see Table 2.3). It has been divided into three areas in order to facilitate its interpretation: a) sampling of information used for learning the operator definitions; b) the characteristics of the action themselves; and c) some of the more relevant features of the learning algorithms.

Several interesting conclusions may be drawn from the summary. The first is that all the work that has been reviewed requires that both pre and post-images be available and associated (*causal relationships*) with an action (*full action signature*). The only work that does not use both images is [Ben95a] because only the pre-conditions are learned. In additional to this, all but two research efforts assume

Table 2.3: Comparison of research in learning AI planning models.

Reference	Sampling					Actions						Learning				
	Observability	Causal Relationship	Trace	Noise	Concurrency	Action Duration	Action Effects	Disjoint Effects	Action Pre-condition	Operator	Representation	Background Knowledge	Incremental	Examples	Induction	Integrated	Evaluate Planning
[YWJ07]	partial[a]	image	plan[b]	no	no	inst.	det.	-	conj.	STRIPS	prop.	no	no	$\oplus$	-[c]	no	no
[ZYHL10]	partial[a]	image	plan[b]	no	no	inst.	det.	yes	conj.	quant.	prop.	no	no	$\oplus$	-[d]	no	no
[SA06]	partial	image	act.	no	no	inst.	cond.	-	$\bar{}$disj.	rule	prop.	no	yes	$\odot$	$\leftarrow^{e}$	no	no
[Wan95]	full	image	act.	no	no	inst.	cond.	-	$\bar{}$conj.	STRIPS	rel.	no	yes	$\odot$	$\leftrightarrows$	yes	yes
[PZK04]	full	image	act.	no	no	inst.	stoch.	-	conj.	rule	prop.	no	no	$\oplus$	$\leftrightarrows$	no	no
[OC96b]	full	post.lag	act.	yes	no	dur.	cond. stoch.	-	$\bar{}$disj.	rule	prop.	no	yes	$\oplus$	$\rightarrow$	no	no
[Ben95a]	full	pre.lag	act.	yes	no	dur.	-	-	1.lit.	rule	prop.	no	yes	$\odot$	$\leftarrow$	yes	no
[TCH99]	partial	image	act.	yes[f]	no	inst.	cond.	-	$\bar{}$conj.	STRIPS	rel.	yes[g]	no	$\odot$	$\leftrightarrows$	no	no
[GMB00]	partial[h]	image	act.	yes	yes	inst.	stoch.	no	conj.	rule	prop.	no	yes	$\odot$	$\leftarrow$	yes	yes
[JV07]	full	image	act.	no	yes	inst.	cond. non-det.	yes	$\bar{}$disj.	rule	prop.	no	no	$\odot$	$\leftrightarrows$	no	no

[a]Partial observability is supported because not all the plan's intermediate states need to be made available. No new relationships are inferred.
[b]State images is used. Can be limited to the plan's initial state and goal only.
[c]Based on weighted MAX SAT
[d]Based on Markov Logic Networks
[e]Algorithm based on filtering propositional formula.
[f]During inference of objects' characteristics.
[g]Limited to the induction of objects' characteristics based on existing type hierarchies
[h]Uses probabilistic planning, so it can deal with hidden states. No new relationships are inferred.

that an action's pre-conditions or effects are not instantaneous [Ben95a, OC96b]. No one has attempted to learn pre-conditions and effects when both pre and post-images and the causal relationships are not instantaneous and known (if this were not so then the time-series segmentation problem would have to be solved). All assume that the actions' signatures, which include the actions' name and respective parameter list, are available.

The ML algorithms that are able to acquire the more complete operator definitions learn *action rules* and not STRIPS-like operators. These rules have the same expressiveness as STRIPS-like operators with: a) disjunct pre-conditions; b) conditional effects; and c) non-deterministic effects (discrete or stochastic). Cases (a) and (b) requires that the ML algorithm be able to learn several rules that refer to the same action. Case (c) requires that the ML algorithm be able to learn several rules with a single pre-conditions and several sets of effects. No one has attempted to learn planning operator definitions that contain resource constraints.

All but two research efforts use the *propositional representation* of the planning domain [Wan95, TCH99]. This may be an indication that the use of ML algorithms based on propositional logic exhibit better performance or facilitates the implementation and testing of the ML algorithms.

Very few ML learning algorithms have been evaluated by verifying operator quality using AI planning directly (planning success) [Wan95, GMB00]. Those that were evaluated, have integrated learning, planning and execution. We suspect that this is because the integration of these procedures requires much effort, which includes the development and use of a simulator. Note that even though [Ben95a] also integrates learning, planning and execution, a classical AI planning algorithm is not used. In fact only one research effort does test operator quality using AI classical planning [Wan95].

From the summary we can see that only two ML algorithms are not based on induction [YWJ07, SA06]. Of those that are based on induction, only one uses specialisation exclusively. All others either generalise the operator definitions which are based on a single example, or use a combination of both generalisation and specialisation. In addition to this we have noticed that those ML algorithms that

execute a systematic search of the search space can be used to learn the planning operators in an incremental fashion. This is true regardless of whether specialisation, generalisation or a combination of these, is used.

Only [GMB00] and [JV07] can learn operators from sampled data whose effects may be produce by concurrently executing actions. In the case of [GMB00] it is unclear whether the inherent difficulties of learning operators in these conditions is actually tested (tests assume that agents simultaneously acting in the same environment, but it is not explicitly stated whether actions execute concurrently or not). As for [JV07], several important simplifying assumptions are made (no overlapping effects of concurrently executing actions, initial domain knowledge is assumed to be close to the target model and no noise). This means that learning operators of concurrently executing actions with overlapping effects in a noisy environment has not been solved. We believe that these requirements currently represents the most difficult problem to be solved in this research area.

No one uses background knowledge to infer additional relationships that can be used to complete or correct the planning operator definitions, save for [TCH99]. It is nevertheless limited to type specialisation of objects based on existing type hierarchies and object properties. However, [SA06] is able to establish relationships between unobserved effects based solely on the sampled information. It is therefore interesting to know if more complex background knowledge can be used, in conjunction with the examples, to infer useful relationships in order to complete the operators definition.

Finally, only two ML algorithms deal with durative actions [Ben95a, OC96b]. This is explained by the fact that both systems where developed for the robotics domains, where this is the usual assumption. Our main interest however is in (neo)classical where actions are assumed to be instantaneous. This is because workflow processes have tasks whose execution time varies widely (from milliseconds to months). It is therefore not feasible that we treat actions as durative.

2.4 Concluding Remarks

We have seen that unrestricted planning's computational complexity is very high. Several constraints may be applied to the planning representation and planning domain in order to improve performance. However, because our aim is to support workflow planning generically, it is neither possible to foresee nor to implement domain specific constraints. In order to reduce such complexity, our best option is therefore to use a proportional representation of the planning domain (PSPACE-complete).

Fortunately the complexity results represent an upper bound on planning efficiency. When solving many practical problems, planning algorithms usually exhibit better behaviour. The simple fact that not all planning operators require the planning language's full expressiveness, allows for more efficient planning. However, supporting workflow planning, also requires that we be able to generate planning types that are more difficult to solve due to the larger search space they represent. Specifically we need to allow for the composition of plans with parallel actions, discrete non-deterministic effects and iterative conditional behaviour[24]. In order to deal with this, we will forego with the requirement that planning be optimal (minimise plan make-span using parallel actions).

Recall that our main objective is the support of flexible ad-hoc workflows via the automated generation and execution of plans. The analysis of the workflow-patterns demonstrates that in order to do this, we need to be able to generate plans with conditional iterative behaviour. Further investigation shows that AI planning via model-checking can compose plans with non-deterministic conditional planning actions. Planning via model checking however has several limitations: a) assumes full observability; b) has limited support for dealing with resources; and c) generates linear plans. Full observability does not present any real hindrance the to adoption of this type of planning paradigm because automate workflow management inherently

[24]Planning complexity depends on the planning languages' expressiveness because such expressiveness directly determines the planning problems' search space. However, generating plans with parallel actions and iterative behaviour inherently increase the planning search space without requiring this to be explicitly expressed in the planning language.

assumes full observability. Both the lack of support for reasoning about resources and non-optimal planning means that the usefulness of the plans is limited to a single agent. It nevertheless allows for efficient plan composition and the use of extended goals may facilitate plan querying and generation.

Note that not all workflow-patterns can be expressed via AI planning. This is because a lot of run-time behaviour is implicitly encoded into the patterns that allow for the activation and cancellation of actions that are not strictly necessary in order to attain the plan's goals. The aim of this dynamic behaviour is to increase plan responsiveness and flexibility, which allows workflow management systems to deal with certain plan execution failures. In this work we will use replanning as means to increase planning flexibility and correct execution failure. Another alternative is to attempt plan repair prior to full replanning.

It is important to note that additional research into AI planning is required in order to fully support workflow management. This includes, but is not limited to: reasoning about resources, model-lite planning (planning with incomplete and evolving domain models) [Kam07], plan repair and planning with extended goals [LPT02, BK98]. Although interesting and of great relevance to our work, it will not be pursued here because our main focus is on automatically acquiring the domain knowledge via ML.

In order to fully appreciate the problem of learning the planning domain model, an extensive yet very focused survey was done on this subject. Our interest lies specifically in research that learns and refines operators for classical planning when no initial partial operator definitions exist. We have observed that initial work in this area attempted to solve the simpler problem of acquiring planning operators in deterministic, fully observable environments where single instantaneous actions are executed. The (chronologically) first algorithms are characterized by: a) being specially conceived for the problem at hand; b) using several separate steps in order to identify the operators' pre and post-conditions; c) integrate learning, planning, and execution; and d) use experimentation as a means to generate examples and therefore induce results. The more recent work attempts to learn operators with conditional and non-deterministic (discrete and stochastic) effects. It also relaxes

several other restrictions imposed on the environment by: a) dealing with noise in the samples (based on statistics); b) use only the observations and not the problem solving episodes as a means to learning and; c) allowing for the concurrent execution of actions. We have detected a trend towards: a) simpler more cohesive learning strategies (single phase) b) the use of generic algorithms for learning (SAT solvers for example) and c) learning without the integration with and interleaving of planning and execution.

The survey has also allowed us to identify the more challenging problems that need to be solved in order to learn planning operators effectively. These issues are obviously closely related to the planning problem itself (plan type, plan goals and assumptions made about the environment). We have seen that the hardest problem to solve is that of acquiring planning operators with *disjunct pre-conditions and non-deterministic effects*. This is especially challenging when several concurrently executing operators may have overlapping effects [JV07]. The *concurrency* of actions significantly increases the learning problem because it make it more difficult to attribute an effect to any given action. However, being able to deal with concurrently executing actions also means that the data set used by the ML algorithm have less restrictions imposed on them.

Another issue that makes acquiring planning operators difficult is that of *noise* in the sampled data. These errors may have several sources: a) faulty sensors; b) exogenous events; and c) perceptual aliasing[25] [Ben95b]. We have noticed that all prior research has used statistical analysis to deal with sampling noise. This usually implies a threshold that is empirically determined. In addition to this, statistical data provide a general framework that can also be used to solve the problem of learning non-deterministic effects (independently of whether or not concurrent actions occur). Note that we can think of exogenous events and perceptual aliasing as representing the concurrent execution of actions whose signature is unknown or whose effects have yet to assigned to the appropriate action, respectively.

[25]Perceptual aliasing is the problem that occurs when two identical inputs should lead to different outputs. In other words two or more actions may be applicable, but only a subset of these actions is actually responsible for the effects. Alternatively, the same action may occur but the effects are non-deterministic.

Research in this area seems to support the conclusion that the use of negative pre-conditions result in more robust set of operators [TCH99, Wan95]. Acquiring negated pre-conditions however increases the difficulty of learning because it enlarges the state space that must be analysed. In addition to this learning may also attempt to establish and use relationships that are themselves acquired automatically. In other words, background knowledge can provide a means to identify new and useful relationships that are not directly observable. For examples, in a robot world with two rooms (left and right), two switches may be placed in the left room and used to switch on lights in the left and right rooms respectively. This means that a robot in the left room may switch the right room's light on and off, but this effect is not directly observable. This is referred to as learning with partial observability[26] and has so far been limited to the operator's pre-conditions [SA06] and parameter types [TCH99]. Like negated pre-conditions, this substantially increases the learning complexity of the ML algorithms. To date only the sample data and object types have been used as background knowledge due in part to the use of propositional logic.

In addition to the above, we also see that some of the surveyed research assumes that experiments (random problem solving episodes and (biased) state based exploration) may be executed at any point in time in order to facilitate learning [Wan95, TCH99, Ben95a]. We will assume that in our case this is not possible because in a workflow setting: a) experiments may be impossible, b) experiments may be too costly and c) if the proposed actions do not solve an agents problem, it will not be executed and the execution's expected feedback (such as detecting an action failure) is not produced. We must therefore obtain all our sampled data from the agents. We make some simplifying assumptions here. First, we admit that any agent that does execute an action, which is suggested by the system, will know if it executed correctly or not. Second all agents that are requested to provide an example, are assumed to have perfect knowledge and will therefore attempt the execution

[26]Note that this does not have the same meaning as in AI planning. In AI Planning, partial observability means that plans may need to use knowledge acquisition actions (observations) in order to correctly discern between states within belief state and thereby select an appropriate state-effecting action.

of a valid action. We do nevertheless accept some errors from these agents. Some work has been done with durative actions [Ben95a, KL06]. We will nevertheless restrict ourselves to instantaneous action so that the resulting operators may be used by the model-checking based AI planner. The surveyed work differs in the use of negative examples. Most work assumes that actions will only be executed if all of operator's the pre-conditions are fulfilled. In this case, when an action does fails, no information on the effects is obtained. We will also make this assumption.

All of the surveyed work requires that sampled data explicitly contain the actions' (full) signature. This means that the ML algorithms are aware of the actions names and the parameters[27]. We will however assume that no action signature is available. Learning must therefore also identify what actions do exists and their possible parameter list consists of. Note that we do admit that action names are provided by the executing agents, but these need not be used consistently. Such name will be simply used to facilitate human readability[28]. This is intended to be our primary contribution to this research area.

A second contribution to this area is to acquire planning operator that adhere to the following restrictions, assumptions and requirements, which to the best of our knowledge, have yet to be supported simultaneously:

Sampling noise: assume that both faulty sensors and exogenous events may occur. The main issue here is in discerning between the valid effects produced by simultaneously executing actions and invalid data contained in inconsistent samples.

Concurrent execution of actions: expect two or more actions to be executed simultaneously. We admit that actions are instantaneous so the issue here is to be able to correctly identify what effects are generated by which actions.

Disjunct pre-conditions: We presuppose that an action may occur in a number situations that may only be adequately described via the use of disjunction.

[27]Note that some work assumes that all operators may depend on or affect any fluent. In this case an operators' parameter list implicitly consists of all possible fluents.

[28]The system is free to use any or all of these names so long as it consistently refers to the same action.

We must therefore be able to learn operator definitions whose pre-conditions could include the use of disjuncts.

Discrete non-deterministic effects: in order to adequately support workflow planning, we must be able to represent the non-deterministic outcomes of actions. This allows one to automate the composition of robust contingency plans, which are essential for supporting workflows' short-term adaptability.

Negated pre-conditions: allow for the modelling of robust operators by making certain pre-conditions explicit. It may be possible, although unlikely, that an operators pre-conditions solely use negated pre-condition (for example, the pre-condition of the *turn-on*(S) operator is $\neg on(S)$).

Negated effects: are a standard modelling requirement in AI planning. Negation describes the states values that are removed by an action. They are also represented as terms in a delete list. It is therefore necessary to acquire these conditions.

Conditional effects: describe an action's outcome that depends on the current world-sate in which it is applied. For example *turn-switch(S)* may be expressed as the conditional operator: $((on(S) \rightarrow \neg on(S)) \oplus (\neg(on(S) \rightarrow on(S))$. This modelling construct is not essential because it can be represented by a set of equivalent operators, each with a different set of pre-conditions and effects. For example the *turn-switch(S)* can be expressed by the operators *turn-on-switch(S)* and *turn-off-switch(S)* with their obvious definitions. Use of conditional planners however facilitate understanding of modelling and may improve planning efficiency. We therefore make this a soft requirement.

Partial observability of pre-conditions: it is advantageous to use all sampled and background knowledge in order to identify relationships that are not directly observable but may be required to adequately describe an action's pre-conditions. The ML algorithm should be able to learn these automatically.

Partial observability of effects: as in the case of *partial observability of pre-conditions*, relationships that are necessary to describe the action's effects,

should be acquired automatically.

No experimentation allowed: because, as was already explained, this could incur high costs or even be impossible to perform. It is therefore necessary to be able to learn from examples only. The ML algorithm should be able to use both positive and negative examples in order to further refine the planning operator. Note that negative examples do not provide any information on the action's effects.

Allow the generation conditional iterative plans: in order to support the execution of workflows it is necessary to generate and execute plans that have loop and branching control flow . This is required for workflows' short-term process adaptability. It is therefore necessary to encode the learned operators so that they may be used by a model-checking based planner in order to successfully compose iterative conditional plans.

The following sections will describe the selection, analysis and empirical testing of several Machine Learning algorithms. The assumptions, restrictions and goals described here, will be used throughout the rest of this work.

Chapter 3

Machine Learning Algorithm

3.1 Introduction

In this chapter we present the algorithm that allows the determination of the number of operators and their definitions. We first provide a set of formal definitions and a formal description of the learning problem. This includes the clarification of known terms such as *operator*, *world state*, *transaction*, *rule* and *support set*. We also introduce definitions that are specific to the problem that include terms such as *qualification*, *overlapping operators* and *co-occurrence*. With all these definitions in mind we proceed with the explanation of how this learning problem is a particular instantiation of the general Minimal Consistent Subset Cover Problem. We describe a greedy heuristic that solves this problem for the case of standard operators and later extend it to identify what are referred to as overlap operators. Finally we describe the types of errors that noisy data may have and how the algorithm must be adapted in order to deal with them.

3.2 A Formal Setting for Learning Planning Operators

Before proceeding with the description of the algorithm we first present the relevant definitions and adopted notation. We will use a first order predicate logic-like

notation to describe the planning operators and world states. We will use some general and basic definitions of first order predicate logic such as literals, ground terms, (well formed) formula, sentence and clause as defined in [Hog90]. Informally all sentences of first order predicate logic consist of constants, functions, variables, predicate symbols and logic connectives (conjunction, disjunction and negation). All symbols will start with a lower-case letter except for variables, which will start with an upper-case letter. A term can be a constant, function or variable symbol or a function symbol applied to a tuple of terms. For example: $function1(constant0, Var1, function2(Var2))$. An atomic formula is a predicate symbol applied to a tuple of terms. Unlike standard predicate logic however we also allow for the typing of constants and variables. A constant or variable may be typed with a constant, which appears after a colon. For example: $function1(constant1 : type1, Var2 : type2, function2(Var2 : type3))$.

Definition 3.2.1 (world state) A world state is a ground clause that consists only of conjunction of literals.

The world state represents a snapshot of the world at a given moment. We assume that this state is consistent in that it represents the state when all actions have terminated (no action is still in execution). It is also consistent in the logical sense in that it does not contain a literal that also appears negated (contradiction).

Definition 3.2.2 (rule) A rule is a clause consisting of an antecedent and consequent connected by a material implication. The antecedent and consequent each consists of a conjunction of literals.

A rule need not be a definite clause[1]. Here we assume that the antecedent and consequent may each have zero or more negative literals. For example $(p_1 \wedge (\neg p_2) \wedge p_3) \rightarrow (e_1 \wedge e_2 \wedge (\neg e_3))$[2].

Definition 3.2.3 (operator) An operator is a rule that has no constants (just variables or functions with variables) such that:

[1]Definite clauses contain exactly one positive literal and zero or more negative literals.
[2]For reasons of clarity we represent literals such as $p_1(X, Y)$ as simple proposition p_1.

- A negative literal can only be in the consequent part if and only if it is in the antecedent,
- A positive literal can only be in the consequent part if and only if it is not in the antecedent,
- The antecedent must have at least one literal,
- The consequent must have at least one literal,
- An antecedent must not have two literals that are a negation of each other (consistency),
- A consequent must not have two literals that are a negation of each other (consistency).

Example 3.2.1 shows the definition of the operator move_down_slow($Lift : elevator, F1 : count, F2 : count$).

Example 3.2.1

$$\begin{aligned}&\text{lift\_at}(Lift, F1) \wedge \ \text{above}(F1, F2) \wedge \\ &\text{reachable\_floor}(Lift, F2) \wedge \text{is\_slow}(Lift) \\ &\rightarrow \\ &\text{lift\_at}(Lift, F2) \wedge \neg\text{lift\_at}(Lift, F1) \wedge \\ &\text{increase}(Total\_cost : cost, travel\_slow(F1, F2))\end{aligned}$$

Definition 3.2.4 (action) An action is an instantiation of an operator. An operator is instantiated by binding the appropriate constants to the variables. An action is therefore a ground formula.

Example 3.2.2 shows the definition of the action move_down_slow($lift\_a : elevator, f1 : count, f2 : count$).

Example 3.2.2

$$\begin{aligned}
&\text{lift_at}(lift_a, f1) \wedge \ \text{above}(f1, f2) \wedge \\
&\text{reachable_floor}(lift_a, f2) \wedge \text{is_slow}(lift_a) \\
&\rightarrow \\
&\text{lift_at}(lift_a, f2) \wedge \neg\text{lift_a}(lift_at, f1) \wedge \\
&\text{increase}(cost_100 : cost, travel_slow(f1, f2))
\end{aligned}$$

A set of operators can be used by an automated planning system to generate plans. Such a set of operators is defined and used according to a domain, which we refer to as a planning domain. For example the operator and action described in Examples 3.2.1 and 3.2.2 belong to the Elevator domain. A planning problem consists of a set of goals (literals) and the respective planning domain. A plan is a description of the order in which non-dependent operators are instantiated into actions and (possibly) executed (simultaneously). An example of a plan is a state machine.

Definition 3.2.5 (qualified operator) A qualified operator is obtained from an operator (as defined in Definition 3.2.3) by doing the following:

- Literals in the antecedent of an operator remain the same in the qualified operator,
- Literals l_k that are in the consequent but not in the antecedent of the operator are qualified as $add(l_k)$,
- Literals l_k that are in the consequent and are negated in the antecedent of the operator are qualified as $del(l_k)$.

Note that any other combination of literals and qualification are not allowed. For example the same literal may not be qualified simultaneously with the `add/1` and `del/1` predicates. The restrictions that apply here are equivalent to those that apply to the operator (see Definition 3.2.3). For example the operator $(p_1 \wedge (\neg p_2) \wedge p_3) \rightarrow (e_1 \wedge e_2 \wedge p_2)$ is qualified as follows: $p_1 \wedge (\neg p_2) \wedge p_3 \wedge add(e_1) \wedge add(e_2) \wedge add(p_2))$.

Definition 3.2.6 (qualified conjunction) A qualified conjunction is a conjunction of literals in negation normal form such that all literals are either positive or

negative when not qualified or are positive only when qualified with the `add/1` and `del/1` predicates.

It is important to note that qualified literals are always positive. A transaction is a qualified conjunction. When we refer to a conjunction we assume it is qualified unless explicitly stated otherwise or it is obvious from the context. We represent a conjunction $\mathbf{p}$ consisting of literals l_k using the angle brackets. For example $\mathbf{p} = \langle l_1, l_2, \ldots l_n \rangle$.

Definition 3.2.7 (transaction) A transaction is a qualified conjunction that represents the change between two chronologically consecutive world states. Assume we have world states S_i and S_{i+1} that were recorded at instants i and $i+1$ respectively, then the transaction T_i is a qualified conjunction such that:

- All literals' constants are consistently substituted by variables,
- If a literal l_k is in S_i and remains in S_{i+1} then $l_k \in T_i$,
- If a literal l_k is in S_i but not in S_{i+1} then $del(l_k) \in T_i$,
- If a literal l_k is not in S_i but is in S_{i+1} then $add(l_k) \in T_i$.

Any subset of literals of a transaction is a qualified conjunction. A qualified operator is also a qualified conjunction. A transaction contains both the representation of state prior to the execution of a set of actions (pre-image) and the representation of the state after all these actions have completed execution (post-image)[3]. In order to differentiate between the pre and post-image literals, post-image literals are qualified with the predicates `add/1` and `del/1`[4], which respectively indicate which literals were added or removed from the pre-image. This format not only reduces the search space but also allows for a direct means of identifying the operator definitions. Note that the use of the qualification must also be consistent. In other words one cannot

[3]Note that invariant states are not represented. Any transaction must have at least one qualified operator.

[4]We assume that no literals that describe the world state use the `add/1` and `del/1` predicate symbols. Alternatively we may randomly assign any two unique symbols that do not exist in the world state representation.

qualify a literal and its negation with the `add/1` or `del/1` predicate. In addition to this the same literal cannot be qualified with both the `add/1` and `del/1` predicates.

When we say that conjunction $\mathbf{p}$ θ-subsumes conjunction $\mathbf{q}$ (see definitions of θ-subsumption in [NCdW95] and [RIAS97]) it means that $\mathbf{p}$ is more general than $\mathbf{q}$ (see definitions of generality in [NCdW95] and [RIAS97]). In other words whenever $\mathbf{p}$ is true $\mathbf{q}$ is also true ($\mathbf{p} \rightarrow \mathbf{q}$). Syntactically we say that $\mathbf{p}$ is a subset conjunction (variant) of $\mathbf{q}$ ($\mathbf{p} \preceq \mathbf{q}$). We will refer to θ-subsumption simply as subsumption.

Definition 3.2.8 (support set) The support set S of a conjunction $\mathbf{p}$ is the set of transactions in which $\mathbf{p}$ is a subset conjunction. Formally let $\mathbf{p}$ be a conjunction, T be a set of N transactions, $T(i)$ the transaction at index $i : i \in [1 \ldots N]$, then the support $S(\mathbf{p}) = \{T(i) \mid \mathbf{p} \preceq T(i)\}$.

Definition 3.2.9 (support value) The support value $|S|$ of a conjunction $\mathbf{p}$ is the number of transactions in which $\mathbf{p}$ is a subset. Formally let $\mathbf{p}$ be a conjunction, let $S(\mathbf{p})$ be the support set of $\mathbf{p}$, then the support value of $\mathbf{p}$ is $|S(\mathbf{p})|$

Definition 3.2.10 (location) The location (idx, pos) is a pair that represents the existence of a literal l_k at the position pos within a transaction $T(idx)$.

Definition 3.2.11 (cover set) The cover set C of a conjunction $\mathbf{p}$ is the set of *locations* of all l_k literals of $\mathbf{p}$. Formally let $\mathbf{p}$ be a conjunction, l_k be a literal of $\mathbf{p}$, T be the set of N transactions, $T(i)$ be the transaction at index $i : i \in [1 \ldots N]$, then the cover set of $\mathbf{p}$ is $C(\mathbf{p}) = \{(i, pos) \mid T(i) \in S(\mathbf{p}) \ \wedge \ l_k \in \mathbf{p} \ \wedge \ pos = \text{location of } l_k \text{ in } T(i)\}$

Definition 3.2.12 (co-occur) We say that a conjunction $\mathbf{p_j}$ co-occurs with a conjunction $\mathbf{p_k}$ if whenever the conjunction $\mathbf{p_j}$ is a subset of the transactions $T' \subseteq T$, then the conjunction $\langle \mathbf{p_j}, \mathbf{p_k} \rangle$ is also a subset of those transactions T'. Formally we write $(\mathbf{p_j} \preceq \langle \mathbf{p_j}, \mathbf{p_k} \rangle) \wedge (S(\langle \mathbf{p_j}, \mathbf{p_k} \rangle) \subseteq S(\mathbf{p_k})) \wedge (S(\langle \mathbf{p_j}, \mathbf{p_k} \rangle) \subseteq S(\mathbf{p_j}))$.

This is also true when either one or both of the conjunctions consist of a single literal. When we say that two conjunctions are not equal it means they are not variants. Two conjunctions are variants if they consist of the same literals and,

after being uniformly renamed, have the same variable bindings modulo renaming. We say that a conjunction $\mathbf{p}$ is a subset of another conjunction $\mathbf{q}$ if there exists a variant of $\mathbf{p}$ in $\mathbf{q}$ such that $\mathbf{p} \preceq \mathbf{q}$.

Definition 3.2.13 (anti-monotonic property) Let $\mathbf{p_j}$ and $\mathbf{p_k}$ be two conjunctions. For any two conjunctions such that $\mathbf{p_j}$ subsumes $\mathbf{p_k}$, it is always true that $S(\mathbf{p_k}) \subseteq S(\mathbf{p_j})$. Formally we write $\mathbf{p_j} \preceq \mathbf{p_k} \rightarrow S(\mathbf{p_k}) \subseteq S(\mathbf{p_j})$.

The anti-monotonic property also applies to the support values. Note that (in both cases) the anti-monotonic property is injective. In other words if $S(\mathbf{p_k}) \subseteq S(\mathbf{p_j})$ it does not necessarily imply that $\mathbf{p_j} \preceq \mathbf{p_k}$. Moreover if $|S(\mathbf{p_k})| \leq |S(\mathbf{p_j})|$ it also does not imply that $S(\mathbf{p_k}) \subseteq S(\mathbf{p_j})$.

We use the anti-monotonic property to prune the search space when looking for the co-occurrence of two conjunctions. Specifically we extend a conjunction $\mathbf{p_j}$ with all literals l_k by first checking that $|S(\mathbf{p_j})| \leq |S(\langle l_k \rangle)|$. We then check for the co-occurrence of the conjunction $\mathbf{p_j}$ with the literal l_k if $S(\mathbf{p_j}) \subseteq S(\langle l_j \rangle)$. Both the support value and support of the literal are used as upper bounds to quickly check for co-occurrence.

Definition 3.2.14 (non-variants set) The non-variants set U is the set of unique literals that are members of the set of literals in the transactions T. Unique means that no two literals in U are variants. More formally let l_k be a literal then $U = \{l_k \mid \exists_i \, l_k \in T(i) \wedge \forall_j l_j \in T, \text{ if } l_k \neq l_j \rightarrow \text{ non-variant } (l_k)\}$

Definition 3.2.15 (maximal) A maximal is a conjunction of linked literals that co-occur with the maximum support value such that there is no other subsumed conjunction consisting of more literals than the first. More formally if $\mathbf{p}$ and $\mathbf{p'}$ are conjunctions whose literals co-occur then $maximal(\mathbf{p}) \leftrightarrow \nexists p' \; : \; (\mathbf{p} \preceq \mathbf{p'}) \wedge (|\mathbf{p'}| > |\mathbf{p}|)$.

A maximal's literals must be either directly or indirectly linked. A literal is linked to another if they share a variable. Two literals are indirectly linked if a chain of common variables found in other literals are also linked. For example the literals

in the conjunction $\langle p(X), q(Y)\rangle$ [5] are not linked but $\langle p(X), q(Y), r(X,Y)\rangle$ are. The predicates `p/1` and `q/1` are indirectly linked by `r/2`.

Definition 3.2.16 (minimal cover set) The minimal cover set is the minimum set of maximals with the highest support values such that:

- Only these maximals are required to cover all literals in all transactions (set cover);
- The same conjunction of literals is covered by the same maximal in all transactions where they co-occur (inter-transaction consistency);
- Each maximal covers at least one literal in a transaction not covered by any other maximal (intra-transaction consistency).

The minimal cover set in effect represents the solution to the problem[6]. The objective of the algorithm is to identify this set.

Definition 3.2.17 (consistent conjunction) A consistent conjunction is a member of the minimal cover set.

Note that the consistent conjunctions are qualified so that each represents a qualified operator (assuming the structure of the maximal is in accordance with the definition of the operator, see Definitions 3.2.3 and 3.2.5). To obtain the final solution we therefore need only to transform these qualified operators into operators.

Definition 3.2.18 (composite) A composite is a maximal that consists of literals that are members of all literals of two or more consistent conjunctions. More formally let l_k be a literal and $\mathbf{C}$ be the minimal cover set, then the composite is a maximal conjunction $\mathbf{p}$ such that $\mathbf{C'} = \{\mathbf{c_i} \mid \mathbf{c_i} \in \mathbf{C} \wedge \mathbf{c_i} \preceq \mathbf{p}\} \wedge |\mathbf{C'}| \geq 2$.

Definition 3.2.19 (combination) A combination is a possibly non-maximal conjunction that consists of literals that are members of some (not all) literals of two or more consistent conjunctions. More formally let l_k be a literal and $\mathbf{C}$ be

[5] For clarity we will represent the connective AND as a comma.

[6] We will later see that this is not quite true in the case the transactions contain noise.

the minimal cover set, then the composite is a maximal conjunction $\mathbf{p}$ such that $\mathbf{C'} = \{\mathbf{c_i} \mid \mathbf{c_i} \in \mathbf{C} \wedge \{l_j \in \mathbf{p}\} \subseteq \{l_k \in \mathbf{C'}\}\} \wedge |\mathbf{C'}| \geq 2$.

Definition 3.2.20 (overlapping operator) An overlapping operator is an operator that consists of two or more consistent conjunctions such that:

- These consistent conjunctions do not (always) co-occur with each other,
- At least one of these consistent conjunctions must not have the structure of a (qualified) operator,
- These consistent conjunctions must combine to form at least two (overlapping) operators,
- The support of the original consistent conjunction that is extended into overlap operators must be covered by a combination of two or more other consistent conjunctions.

In order to make the above definition clear we provide an example. Assume we have the following set of consistent conjunctions: $\langle p_a, p_b \rangle$, $\langle p_c, p_d \rangle$, $\langle e_1, e_2 \rangle$, $\langle e_3, e_4 \rangle$. The predicates p_i represent the state of the pre-image in the transactions. The predicates e_j represent the state of the post-image in the transactions. If for every transaction wherein $\langle p_a, p_b \rangle$ occurs either conjunction $\langle e_1, e_2 \rangle$ or $\langle e_3, e_4 \rangle$ also occur, then the the combinations $\langle p_a, p_b, e_1, e_2 \rangle$ and $\langle p_a, p_b, e_3, e_4 \rangle$ represent a set of consistent state changes. Moreover all transactions originally covered by the separate consistent conjunctions can also be consistently covered by these combinations. In other words $(p_a \wedge p_b) \rightarrow (e_1 \wedge e_2)$ and $(p_a \wedge p_b) \rightarrow (e_3 \wedge e_4)$ are operators in the solution. The same is true for $\langle p_c, p_d \rangle$, $\langle e_1, e_2 \rangle$ and $\langle e_3, e_4 \rangle$ which result in the operators $(p_c \wedge p_d) \rightarrow (e_1 \wedge e_2)$ and $(p_c \wedge p_d) \rightarrow (e_3 \wedge e_4)$. It is important to point out that as soon as we generate the first pair of rules, the second pair must also follow in order to ensure full coverage. If this were not the case we would end up with a more general conjunction (say $e_1 \wedge e_2$)[7] that would subsume a more specific rule $(p_a \wedge p_b) \rightarrow (e_1 \wedge e_2)$.

[7] In the general case the consistent conjunctions that are combined may be rules themselves so we could say *more general rule* instead of *more general conjunction*.

Two issues arise here. The first is that the combining of consistent conjunctions to form overlapping operators is not limited to combining two conjunctions. This process may be repeated several times by continually extending the conjunctions. The second is that assuming we have both overlapping and non-overlapping operators we need to know which of these conjuncts can be combined to form consistent overlapping conjunctions. In both cases we require additional criteria that will allow us to stop combining and/or limiting the selection of consistent conjunctions that are combined. In order to prevent over-fitting the first criteria is to select only those consistent conjunctions for combination that do not have the structure of an operator. These conjunctions however, can be combined with any other of the consistent conjunctions. The second criteria is that a combination must result in at least two new conjunctions. The third criterion is that the all of the combinations must have the same support as conjunction that is common to all of the overlap conjunctions. In the example above $S(\langle p_a, p_b\rangle) = S(\langle p_a, p_b, e_1, e_2\rangle) \cup S(\langle p_a, p_b, e_3, e_4\rangle)$.

Note that the discussion above shows us that even though we may originally have overlapping operators that generate the transactions, we will not identify these if all the consistent conjunctions that comprise these overlapping operators have the structure of an operator. For example if we identify the consistent conjunctions $\langle p_a, e_1\rangle$, $\langle p_c, e_3\rangle$, $\langle p_b, e_2\rangle$ and $\langle p_d, e_4\rangle$[8] from the overlapping operators $(p_a \wedge p_c) \rightarrow (e_1 \wedge e_3)$ and $(p_a \wedge p_d) \rightarrow (e_1 \wedge e_4)$ and $(p_b \wedge p_c) \rightarrow (e_2 \wedge e_3)$ and $(p_b \wedge p_d) \rightarrow (e_2 \wedge e_4)$ then we will not be able to determine these operators. Of course it now becomes debatable whether or not those overlap operators are not redundant and therefore possibly incorrect.

Definition 3.2.21 (overlap component) An overlap component is a consistent conjunction that does not have the structure of a (qualified) operator and is combined with one or more consistent conjunctions to form an overlapping operator. More formally let $\mathbf{C}$ be the minimal cover set then $\mathbf{o}$ is the overlap component such that $(\mathbf{o} = \mathbf{c_i}) \ \wedge \ \mathbf{c_i} \in \mathbf{C} \wedge \text{not\_operator}(\mathbf{c_i})$

[8] $\langle p_a, e_1\rangle$ occurs with $\langle p_b, e_2\rangle$ and $\langle p_d, e_4\rangle$, and $\langle p_c, e_3\rangle$ occurs with $\langle p_b, e_2\rangle$ and $\langle p_d, e_4\rangle$.

Definition 3.2.22 (overlap conjunction) An overlap conjunction is a non-consistent conjunction that has the structure of a (qualified) operator consisting of at least one overlap component and forms an overlapping operator. More formally let $\mathbf{C}$ be the minimal cover set, $\mathbf{o_k} = \langle c_1, c_2, \cdots, c_n \rangle$ be a overlap conjunction and $\mathbf{O}$ be the set of all overlap conjunctions, then $\forall_{k,i}\ \mathbf{o_k} = \{\mathbf{c_i} \mid (\mathbf{c_i}, \mathbf{c_j} \in \mathbf{C}) \wedge (\mathbf{c_i} \neq \mathbf{c_j}) \wedge (S(\langle \mathbf{c_j}, \mathbf{c_k} \rangle) \neq \emptyset)\} \wedge (|\mathbf{o_k}| > 1) \ \wedge \ (\cup_{c_i \in \mathbf{O}} S(c_i) = \cup_k S(o_k))$

Definition 3.2.23 (candidate) A candidate conjunction is a member of the set of candidate conjunctions. A candidates conjunction is either a consistent, composite, combination or overlap conjunction.

Definition 3.2.24 (seed) A seed of a conjunction is the initial literal that is selected and extended with co-occurring literals to form a maximal.

With the above set of basic definitions we are now in a position to describe the proposed algorithm.

3.3 The Learning Algorithm

Given a set of transactions, it is possible for us to correctly identify a qualified operator. More concretely, we may determine such an operator in the limit, that is, when the number of transactions increases to infinity[9].

To do this we need "only" to determine the maximum set of linked literals that always co-occur i.e.: find a state transition that is consistently the same. For example if the qualified literals $\langle p_1, p_2, \ldots, p_j, e_1, e_2, \ldots, e_k \rangle$, where $p_{i \in [1 \ldots j]}$ are pre-image literals and $e_{l \in [1 \ldots k]}$ are qualified post-image literals, that co-occur, then this consistent conjunction represents a qualified operator. If there exists at least one transaction wherein these literals do not co-occur then they cannot represent a single state transition. For example assume we have N transactions wherein $N-1$ transactions have the qualified conjunction $\langle p_1, p_2, p_3, e_1, e_2, e_3 \rangle$ but there exists a

[9] Assuming we have L literals and that all transactions have anywhere between 1 to C simultaneously executing actions consisting of D literals, then we need only consider all possible $f(L, C, D)$ combinations of these concurrently executing actions.

single transaction that has the qualified conjunction $\langle p_1, p_3, e_1 \rangle$. In this case it is obvious that at least one state transition can always be described by the operator $(p_1 \wedge p_3) \rightarrow e_1$.

We assume that one or more actions may be executed concurrently. This means that a transaction may contain a state transition produced by one or more actions. It is therefore necessary to identify a set of consistent conjunctions that take into account all of literals within the transactions. We must determine the minimum set of conjunctions that cover all literals in the transactions. Specifically we must only introduce a consistent conjunction if it covers at least one literal not covered by any other consistent conjunction. This also implies that no two consistent conjunctions in the final set of covering conjunctions subsume one another.

A consistent conjunction may cover one or more literals also covered by another consistent conjunction. In the previous example not all of the literals $\langle p_1, p_2, p_3, e_1, e_2, e_3 \rangle$ are covered by the single operator $(p_1 \wedge p_3) \rightarrow e_1$. In order to cover the complete set of transactions we may find that an additional two operators are required: $(p_1 \wedge p_3) \rightarrow e_2$ and $p_2 \rightarrow e_3$. We can see that a non-deterministic operator exists: both $(p_1 \wedge p_3) \rightarrow e_1$ and $(p_1 \wedge p_3) \rightarrow e_2$ have the same pre-conditions but different effects. They can effectively be combined into a single operator $(p_1 \wedge p_3) \rightarrow ((e_1 \wedge \overline{e_2}) \vee (\overline{e_1} \wedge e_2))$.

It is important to point out the following issue regarding truly non deterministic actions. Assume we have the following three possible non-deterministic transitions: $\langle p_1, p_2, e_1 \rangle$, $\langle p_1, p_2, e_2 \rangle$ and $\langle p_1, p_2, e_1, e_2 \rangle$. The transition may be identified as consistent conjunctions which result in the following operator definitions: $(p_1 \wedge p_2) \rightarrow e_1$, $(p_1 \wedge p_2) \rightarrow e_2$ and $(p_1 \wedge p_2) \rightarrow (e_1 \wedge e_2)$. Because we must ensure that no subsumption occurs, only the first two (consistent conjunctions) operators are retained. In other words we assume that two actions take place in order for the effect $\langle e_1, e_2 \rangle$ to be observed simultaneously. Unlike the previous example we must combine these operators without using mutually exclusion to obtain $(p_1 \wedge p_2) \rightarrow (e_1 \vee e_2)$ in order to account for the simultaneous occurrence of $\langle e_1, e_2 \rangle$. If such a combination of operator definitions were to be undertaken in a post-processing phase then it would

be necessary to check if disjunction or mutual exclusion should be used[10].

Under the conditions described above, it is still possible not to identify one or more operators. If two or more different actions co-occur in all of the set of transitions T, then only one consistent conjunction and therefore one operator, may be identified. However the probability that different actions co-occur decreases as the number of transactions increases. In the limit, we will eventually encounter a transaction where those two actions do not co-occur. In fact the probability that two or more actions erroneously co-occur is very small. This probability (decreases) also depends on: the number of operators in the domain (increases), the number of actions that may be concurrently executed (decreases), whether or not a constant or varying number of actions execute simultaneously (varying) and the distribution with which the various operators (uniform).

3.3.1 Minimal Consistent Subset Cover Problem

In order to determine the complete set of operators, we need to search for the minimum set of consistent conjunctions that cover all the literals in all the transactions. As with many Machine Learning algorithms we must solve an instance of the Set Cover (SC) problem, more specifically the Minimal Consistent Subset Cover (MCSC) problem ([GEC+07]). This is a covering and not a partitioning problem, because the operators (and therefore the consistent conjunctions), may have (cover) common literals. Note that this is a MCSC problem (as opposed to a SC problem) because we do not have any prior knowledge of the conjunctions.

The MCSC problem is NP-hard ([GEC+07]) and we must therefore resort to the use of heuristics (page 15 in [Vaz01]) to make it tractable. We use a greedy heuristic that selects an uncovered literal l_i from a transaction $T(j)$. We then extend this seed literal to form a maximal conjunction $\mathbf{p}$. Once the consistent conjunction $\mathbf{p}$ has been identified then all occurrences of this conjunction in the set of transactions T are marked as covered. The process is repeated until no literal remains uncovered. This resulting set of maximal conjunctions constitute the set of candidate conjunctions.

[10]Whenever combining operators with the same antecedents if there exists at least one transaction wherein both of the effects occur simultaneously, then a disjunction must be used.

A final step removes all conjunctions that do not uniquely contribute to covering at least one literal not covered by any other maximal. The remaining conjunctions are then a set of consistent conjunctions that can be directly interpreted as operators.

The following sections describe the details of this algorithm. We describe the proposed algorithm in three steps. In this subsection we present the first version of the algorithm that learns non-overlapping operators in a noiseless setting Then in sub-section 3.3.3 we extend the algorithm to account for the learning of overlap operators. Finally in subsection 3.3.4 we further extend the algorithm in subsection 3.3.3 to learn in a noisy setting.

Algorithm 1: learnOperators(T, α, β, γ)

Input: T set of transactions
Input: α omission error extend frequency
Input: β commission error cut-off frequency
Input: γ omission error cut-off cover
Output: $\mathcal{C}$ set of maximal clauses (operator definitions)

```
1 T ← sort_literals_per_transaction(T)
2 T ← sort_transactions_by_length(T)
3 U ← identify_unique_literals(T)
4 S ← count_support_of_unique_literals(U, T)
5 M ← get_maximal_clauses(S, T, α, β)
6 G ← purge_subsuming_clauses(M, α)
7 C ← purge_non_unique_cover_clauses(G, γ)
8 C ← is_valid_simplified_operator(C)
9 return C
```

Algorithm 1 shows the main steps used in generating the set of consistent conjunctions. It takes as input a set of transactions T that make up the data set and the noise handling parameters α, β and γ. It outputs the set of operator definitions $\mathcal{C}$. This algorithm is structured into three parts: pre-processing (lines 1 to 4), learning operators (line 5) and post-processing (lines 6 to 8).

In line 1 the literals in each transaction are ordered lexicographically. Next (in line 2) we sort the transactions by increasing length so that transactions consisting of fewer literals appear first. This ordering will later allow us to efficiently select a seed from a transaction that has a minimum number of literals thereby reducing the time used in testing co-occurrence. We then identify the set of non-variant literals (line 3) and determine their support and support values (line 4). For each non-

variant literal we also record the locations where they occur. The location consists of a pair (idx, pos) where idx is the index of the transaction $T(idx)$ and pos is the index of the literal within the transaction. Note that, because actions of the same operator may execute concurrently and because such literals may also be common among actions of different operators, a non-variant literal may appear several times within the same transaction.

In line 5 we start by generating the set of candidates using the greedy covering algorithm. Each candidate conjunction is a unique (non-variant) consistent conjunction. For each of these conjunctions all locations that are covered by this variant are recorded. For example the consistent conjunctions $\langle p(X,Y), q(X,Z)\rangle$ and $\langle p(X,Y), q(Z,X)\rangle$ are non-variants. The coverage of $\langle p(X,Y), q(X,Y)\rangle$ is recorded as $\{idx(1) : [\langle 1,2\rangle, \langle 1,3\rangle], idx(2) : [\langle 3,4\rangle]\}$ meaning that variants of this conjunction occur in two transactions ($T(1)$ and $T(2)$) and that this variant occurs twice in the first transaction ($\langle 1,2\rangle$ and $\langle 1,3\rangle$ occur in transaction $T(1) = \langle p(X,Y), q(X,Z), q(X,W)\rangle$). Note that the literal $p(X,Y)$ is shared by the two occurrences of the variant $\langle p(X,Y), q(X,Z)\rangle$ in transaction $T(1)$. It can also just as easily be shared by another non-variant.

Next (line 6) all clauses that subsume any other clause are removed from the candidate solution. In line 7 all of those candidates that do not cover a literal that is not covered by any of the other maximal are removed. To test unique coverage we start off by generating for each candidate the set of locations (idx, pos) that it covers. We call this the cover set. Note that one n-tuple location exists for each literal in every occurrence of the candidate variant. For each candidate we then check if its cover set is a subset of the union of all other candidates. If it is then it does not contribute to covering a unique literal and can therefore be safely removed. Finally (line 8) all those conjunctions that do not represent an operator (rule) are discarded.

Algorithm 2 does the actual greedy search for candidates described at the start of this section. Let `T` be the set of transactions, `S` the non-variant set and `M` the output set of candidates. In line 1 the literals that have yet to be covered are recorded as the set of locations `U`, which is initialized with all of the literals in T.

Algorithm 2: get_maximal_clauses(S,T,α,β) Greedy algorithm.

Input: T set of transactions
Input: S non-variant set of literals
Input: α omission error extend frequency
Input: β commission error cut-off frequency
Output: M Set of maximal clauses

```
   // Order by (support, transaction length)
 1 U ← coverage_of_non_variant_set(T, S)
 2 M ← ∅
 3 while U ≠ ∅ do
 4     idx, pos, C ← select_one_uncovered(U)
 5     W ← rest_of_literals(T,idx,pos)
 6     max ← extend_co_occurring_literals(T,C,idx,pos,W,α,β)
 7     if S( max ) > β  then
 8         M ← M ∪ { max }
 9     endif
10     U ← U \ cover(max)
11 endw
12 return M
```

The n-tuples in the cover set `U` are ordered first by increasing support values and then by increasing transaction length. We pick the seed literals with lower support values first because then we need only test their co-occurrence against all literals of the non-variant set whose support values are at least as large as its own. This also reduces the number of transactions that have to be scanned when testing for co-occurrence because lower support values mean occurrences in fewer transactions. Assuming no noise in the data (specifically errors of omission) we need only test co-occurrence against literals that occur in the same transaction as the seed (or for that matter any other transaction where the seed's variant occurs). If we select the transaction with the least number of literals then we obviously also reduce the number of co-occurrence tests.

In order to generate a candidate we first select an uncovered seed ($\mathcal{C}$) that has the location (`idx, pos`) (line 4). Next we obtain the set `W` of all the literals in the transaction `idx` excluding the seed (line 5). The seed $\mathcal{C}$ is then extended with as many of the co-occurring literals in `W` as possible (line 6). The resulting maximal `max` is recorded as a candidate (line 8) and all of the literals covered by this maximal are deleted from the set `U` (line 10). This process (lines 4 to 10) is repeated until all

of the literals have been covered (line 3).

This algorithm is guaranteed to terminate if and only if we ensure that the maximal that is generated at each iteration covers the seed's original location. To see why the algorithm may not terminate consider what happens when the seed is extended. We may find several non-variant conjunctions that co-occur with the seed but only one of these will cover the seed's original location. For example assume the seed $p(X, Y)$ located at $(idx = 1, pos = \langle 1 \rangle)$ is extended with a literal $q(X, Y)$. We may find that the conjunctions $\mathbf{c_1} = \langle p(X, Y), q(X, Z) \rangle$ and $\mathbf{c_2} = \langle p(X, Y), q(Z, X) \rangle$ exist at locations $(idx = 1, pos = \langle 1, 3 \rangle)$ and $(idx = 1, pos = \langle 2, 3 \rangle)$, respectively. In order to find the maximal we must keep on extending one of these conjunctions. Assume we deterministically select the conjunction that has the maximum support value. This may be the $\mathbf{c_2}$ conjunction that does not cover the original seed's location. The cover of the set of `U` is then updated in line 10, which means that the location $(idx = 1, pos = \langle 1 \rangle)$ remains in `U`. The next iteration will select the same seed, which will generate the same set of extended conjunctions and hence result in the repeated generation of the same maximal. The process will then repeat itself without terminating.

Algorithm 3 represents the core of the ML algorithm. Let `T` be the set of transactions, $\mathcal{C}$ the maximally extended conjunction (initially this is set to the seed), the pair (`idx`, `pos`) the original seed's location and `W` the set of possibly co-occurring literals in transaction idx. This algorithm outputs a maximally extended conjunction $\mathcal{C}$.

In line 1 we initialize the flag `did_grow` to false. This flag is set to true if the current conjunction $\mathcal{C}$ was successfully extended with at least one co-occurring literal from the set `W`. The initial set of literals `again` is made empty. It holds any of the literals from `W` that are temporarily kept for later retesting of co-occurrence.

The algorithm starts off by selecting and removing a literal `lit` from the set `W` (line 4). Currently the initial set of literals in `W` are ordered such that they are linked.[11] The function `remove_one()` simply picks the first one that is available.

[11] See Definition 3.2.15 and the subsequent description of linked literals.

Algorithm 3: extend_co_occurring_literals(T,$\mathcal{C}$,idx,pos,W,α, β)

Input: T set of transactions
Input: $\mathcal{C}$ clause to be extend
Input: idx transaction index of selected uncovered clause $\mathcal{C}$
Input: pos position within transaction idx of selected uncovered clause $\mathcal{C}$
Input: W possibly co-occurring literals in transaction idx
Input: α omission error extend frequency
Input: β commission error cut-off frequency
Output: $\mathcal{C}$ Clause of co-occurring literals

```
1  did_grow ← false
2  again ← ∅
3  while W ≠ ∅ do
4      W, lit ← remove_one(W)
5      db ← extend_if_co_occur(C, lit, α, β)
6      linked, unlinked ← split_list(db)
7      linked ← filter_covers_original(pos, idx, linked)
8      unlinked ← filter_covers_original(pos, idx, unlinked)
9      linked ← commission_error_filter(linked, β)
10     unlinked ← commission_error_filter(unlinked, β)
11     linked ← omission_error_filter(linked, C, α)
12     unlinked ← omission_error_filter(unlinked, C, α)
13     switch linked, unlinked do
14         case {...}, ∅
15             C ← search_best(linked)
16             did_grow ← did_grow ∨ true
17         endsw
18         case ∅, {...}
19             again ← again ∪ { lit }
20         endsw
21         case ∅, ∅
               // Do nothing.  Try another.
22         endsw
23         case {...}, {...}
24             C ← search_best(linked)
25             did_grow ← did_grow ∨ true
26         endsw
27         if (W = ∅) ∧ (again ≠ ∅) ∧ did_grow  then
28             W ← again
29             did_grow ← false
30             again ← ∅
31         endif
32     endsw
33 endw
34 return C
```

The ordering is done only once at the start so it does not guarantee that all literals selected from `W` will be linked to the seed or among themselves. Alternate strategies for ordering and selecting the literals in W can also be used.

The next step (line 5) is to check for the co-occurrence of the conjunction $\mathcal{C} = \mathcal{C} \cup \{\texttt{lit}\}$. The result is a set `db` of non-variant conjunctions if they do co-occur otherwise an empty set is returned. The function `extend_if_co_occur()` checks for and groups all variants by matching the co-occurring conjunctions. It also records all of the locations of each of these variants. Some of the conjunctions in `db` however may be unlinked. As an example consider the co-occurrence check of $\langle p(X)\rangle \cup \langle q(Y)\rangle$. We may find two conjunctions $\mathbf{c_1} = \langle p(X), q(X)\rangle$ and $\mathbf{c_2} = \langle p(X), q(Y)\rangle$. In line 6 these conjunctions are split into two sets: those conjunctions that are linked ($\mathbf{c_1}$) and those that are unlinked ($\mathbf{c_2}$).

As was previously explained, in order to guarantee the convergence of the greedy cover algorithm, we must ensure that the original seed's location is still covered. In lines 7 and 8 we remove all those conjunctions whose locations do not include the original seed location.

According to the resulting sets of linked and unlinked conjunctions we have 4 cases to deal with. In the first case (line 14) no unlinked conjunctions were found. In other words we have successfully extended $\mathcal{C}$ with the literal `lit` to form linked conjunctions only. Of these we select the best one (line 15) and indicate, via the `did_grow` flag, that an extension was successful (line 16). The current criteria for selecting a conjunction is to pick the one with the highest support value.

Case two (line 18) deals with the case when only unlinked co-occurring conjunctions were found. Here we consider that the extension using the literal `lit` failed. However we record it for possible retesting later (line 19). This is required because a literal that is currently unlinked my later become linked once the conjunction $\mathcal{C}$ has been extended with another literal from `W`. To demonstrate this, consider the previous example's unlinked conjunction $\mathbf{c'_2} = \langle p(X), q(Y)\rangle$. We cannot discard $q(Y)$ because once $p(X)$ has been extended with the literal $r(X, Y)$ to form $\mathbf{c_2} = \langle p(X), r(X, Y)\rangle$ then we may later successfully extend it to form $\mathbf{c_3} = \langle p(X), r(X, Y), q(Y)\rangle$.

The third case (line 21) occurs when neither linked nor unlinked literals were found. Here we assume that there are no co-occurrences and the literal `lit` is discarded. It is safe to discard such a literal due to the anti-monotonic property of the support. The fourth and last case (line 23) is a combination of the first two cases. In other words we found both linked and unlinked co-occurring conjunctions. We proceed just as we did in the case one and discard the literal from further testing because it has been successfully used.

The steps above are repeated (while loop in line 3) until the conjunction cannot be extended any further, in other words $\mathcal{C}$ is a maximal. Note that the process is restarted whenever the current pool of possibly co-occurring literals `W` is depleted but the conjunction has been successfully extended and previously unlinked co-occurring literals (in `again`) still exist (line 27).

3.3.2 Learning Non-overlapping Operators

As with any other heuristic, the solution is not guaranteed to be optimal. In this section we explain how and why the algorithm may incorrectly produce some composite and (non-maximal) combination conjunctions. This provides us with the insight necessary to explain and possibly improve on the current results. We do this by considering the conditions under which the strategies employed by the heuristics fail and therefore induce error.

Let's assume we have two operators whose definitions are $\mathbf{op_1} = \langle p(X,Y), q(X,Z)\rangle$ and $\mathbf{op_2} = \langle p(Y,X), q(Z,X)\rangle$. Instances of these operators occur in a subset of the transactions set T such that: $\mathbf{op_1}$ and $\mathbf{op_2}$ co-occur in U transactions, $\mathbf{op_1}$ occurs in V transactions where $\mathbf{op_2}$ does not occur and $\mathbf{op_2}$ occurs in W transactions where $\mathbf{op_1}$ does not occur. Lets also assume that $|U| = |V|$ and $|U| = |W|$ and that all transactions V and W are longer (have more literals) than transactions U. The former conditions ensure that any of the literals $p(X,Y)$ or $q(X,Y)$ may be selected in Algorithm 2 as seeds because the ordering by support value is effectively random.[12] The latter condition ensures that a seed will be picked from one of the U trans-

[12] All literals have the support value of $|U \cup V \cup W|$ so any ordering is possible.

actions. Let's assume that a transaction $T(i)$ in U, has the the following literals $\langle p(X,Y), q(X,Z), p(U,W), q(Z,W)\rangle$, and that its first literal is selected as the seed.

Under these conditions Algorithm 3 can start with the seed $\mathcal{C}$ set to $p(X,Y)$ and `W` containing three literals: $p(X,Y)$, $q(X,Y)$ and $q(X,Y)$. The initial tests of co-occurrence use the support of the literals in `W` and determine that $p(X,Y)$ and $q(X,Y)$ possibly co-occur ($S(q(X,Y)) \subseteq S(p(X,Y))$[13]). A first attempt at extending the seed with $p(X,Y)$ verifies that the resulting conjunction is unlinked (the conjunction $\langle p(X,Y), p(U,W)\rangle$ in $T(i)$ is unlinked). This literal is kept in `again` for later retesting. Next the seed is extended to form the consistent conjunction $\langle p(X,Y), q(X,Z)\rangle$. However it does not stop here.

The set `W` still contains the literal $q(X,Y)$ for testing. At this point the $\mathcal{C}$ conjunction has the support $S(\langle p(X,Y), q(X,Z)\rangle) = U \cup V$, the literal has support $S(\langle q(X,Y)\rangle) = U \cup V \cup W$ and hence they possibly co-occur ($U \cup V \subseteq U \cup V \cup W$). The process continues reaching the point where $\mathcal{C}$ is the conjunction $\langle p(X,Y), q(X,Z), q(Z,W)\rangle$ and the set `again` contains the literal $p(X,Y)$. Once more the process does not terminate because the last attempt to extend the conjunction succeeds and the set `again` is not empty. The literal in `again` is used to extend the conjunction $\mathcal{C}$ resulting is the composite $\langle p(X,Y), q(X,Z), p(U,W), q(Z,W)\rangle$.

The example above shows that the algorithm may generate composites. Note that selecting the seed may use more sophisticated criteria such as picking the next seed from a transaction which has the least number of co-occurring literals. However, as we will see later, such criteria will not allow us to deal with the issue of errors of omission in the data set.

For the next example let's assume we have two operators whose definitions are $\mathbf{op_1} = \langle p(X,Y), q(X,Z), r(X,Z)\rangle$ and $\mathbf{op_2} = \langle p(Y,X), q(Z,X), s(W,Z)\rangle$. Instances of these operators occur in a subset of the transactions set T such that: $\mathbf{op_1}$ and $\mathbf{op_2}$ co-occur in U transactions, $\mathbf{op_1}$ occurs in V transactions where $\mathbf{op_2}$ does not occur and $\mathbf{op_2}$ occurs in W transactions where $\mathbf{op_1}$ does not occur. Let's also assume that $|U| = |V|$ and $|U| = |W|$ and that all transactions in V and W are longer (have

[13] In this case both are equal to $U \cup V \cup W$

more literals) than transactions in U. The former conditions ensure that any of the literals $r(X,Y)$ or $s(X,Y)$ may be selected in Algorithm 2 as a seed because they have the least support.[14] The latter condition ensures that a seed will be picked from one of the U transactions. Let's assume that a transaction $T(i)$ in U, which has the the following literals $\langle p(X,Y), q(X,Z), r(X,Z), p(U,W), q(Z,W), s(W,Z)\rangle$, has the literal $r(X,Z)$ selected as the seed.

Let's assume that the conditions above, Algorithm 3 starts with the seed $\mathcal{C}$ set to $r(X,Z)$ and $\mathtt{W}$ contains four literals: $p(X,Y)$, $q(X,Y)$, $q(X,Y)$ and $s(X,Y)$. When the algorithm first attempts to extend $\mathcal{C}$ with the literal $s(X,Y)$ the co-occurrence test fails and this literal is therefore discarded.[15] The algorithm then proceeds in the same manner as the previous example generating the non-maximal combination $\langle p(X,Y), q(X,Z), r(X,Z), p(U,W), q(Z,W)\rangle$. It is non maximal because the literal $s(X,Y)$ was discarded. Note however that if the literals were tested in a different order (testing $s(X,Y)$ last) then the final conjunction would be maximal (composite).

The example above shows that the initial co-occurrence testing of the literal may inadvertently remove a literal from a conjunction making it non-maximal. This is because every time we extend a conjunction its support may diminish (anti-monotonic property) so that literals that previously did not co-occur may later do so. Note that there exists an ordering of $\mathtt{W}$ for testing co-occurrence that will ensure that all relevant literals will be added (the literals with least support value should be tested last, in the previous example $s(X,Y)$ would be the last literal). Determining and using such an ordering however is not advantageous because it not only increases processing time but also generates composites and combinations that do not contribute to the correct solution.

When the errors above are induced, it does not mean that the correct solution cannot be found. After generating the composites and combinations, not all of the literals will be covered. In both of the above cases literals in the transactions U and V will not be covered. These literals will be selected later as seeds thereby

[14]The literal $r(X,Y)$ has support $U \cup V$ and literal $s(X,Y)$ has support $U \cup W$.

[15]Literals $r(X,Y)$ and $s(X,Y)$ do not co-occur because $(U \cup V) \nsubseteq (U \cup W)$ and $(U \cup W) \nsubseteq (U \cup V)$

generating the consistent conjunctions. After coverage of all literals have been found the composites and combinations can then be removed because they do not provide unique coverage of at least one literal.

As concluding remarks we make the following observations. First we can see that the pruning of the search space is a result of checking for the co-occurrence of a conjunction and a literal. This pruning may inadvertently remove literals generating non-maximal conjunctions. Second we repeatedly check for linkage between a conjunction and a literal. There may exist an order of literals in `W`, which could, at worst, result in an exponential number of comparisons. Finally even if *intermediate* erroneous (non-consistent conjunctions) are generated this does not prohibit the identification of the correct solution.

3.3.3 Learning Overlapping Operators

The algorithm described in the previous section is able to determine all of the consistent conjunctions without error.[16] However it is still unable to detect a specific type of operator, referred to as overlapping operators (see Definition 3.2.20). These operators' literals do not always co-occur but they are made up of a combination of two or more consistent conjunctions. The problem with identifying such operators is twofold: the need to combine consistent conjunctions without over-fitting[17] and being able to combine the consistent conjunctions repeatedly in order to correctly determine overlap operators consisting of several maximals. This section describes an extended version of the algorithm that does exactly this.

Algorithm 4 takes as input the set of transactions `T` and the set of consistent conjunctions $\mathcal{C}$ and produces a new set of candidates $\mathcal{C}$ so that the latter set of conjunction may possibly contain the definitions of one or more overlap operators. First we determine which of the candidates are overlap components (see Definition 3.2.21) that can be combined to form overlap operators (line 2). We then generate a set[18] of

[16] Without error here meaning that no additional candidates besides the consistent conjunctions are retained.

[17] Over-fitting here refers to the combination of operators that should not be combined.

[18] Implemented as a Trie data structure.

Algorithm 4: extend_to_overlap_actions(T,$\mathcal{C}$,β)

Input: T set of transactions
Input: $\mathcal{C}$ set of candidates
Input: β commission error cut-off frequency
Output: $\mathcal{C}$ set of possibly extended candidates

```
1  while true do
2      to_extend,extended_with ← valid_simplified_operators(C)
3      db ← non_variants_trie(C)
4      if to_extend = ∅ then
5          return C
6      endif
7      extents ← extend_pairs(T, to_extend, extended_with, β)
8      extents ← purge_super_non_unique_cover(extents)
9      if extents = ∅ then
10         return C
11     else
12         all ← C ∪ extents
13         C ← purge_super_non_unique_cover(all)
           // New set of conjunctions unchanged
14         if C ⊆ db then
15             return C
16         endif
17     endif
18 endw
```

unique variants `db` (line 3), which will be used to terminate the process of combining components. If we find that no overlap components exist (line 4) then no (further) combining of maximals is performed and the current set of candidates $\mathcal{C}$ is returned (line 5). Otherwise the overlap components are combined with any maximal (line 7) to produce the list `extents` of newly extended conjunctions, some of which may possibly represent overlap operators.

The `extents` list is then purged of any conjunctions that do not provide unique coverage (line 8). However, unlike the test of unique coverage used in the previous algorithms, care must be taken here. Note that according to the definition of overlap operators, these will never provide unique coverage because such coverage is already provided by the set of its combined subsuming conjunctions. In order to solve this, we purge all shorter conjunctions that do not cover at least one unique literal as compared all of the conjunctions that are at least as long as itself. To do this the function `purge_super_non_unique_cover()` simply orders the `extents` in increasing order of their length and then checks each conjunction `extents(i)` against all other `extents(j)` for all $j > i$[19].

If no `extents` are generated (line 9) then the current set of candidates $\mathcal{C}$ is returned (line 10). If, on the other hand, extensions were generated, then we must check if a new set or the same set of candidates were generated. To do this we first collect all of the prior and current conjunctions (line 12), remove any unneeded conjunctions (line 13) and then check if the set of all conjunctions is the same as the set of conjunctions we had last generated (or started off with). If this is so (line 14) then we have reached a fixed-point and can return the current set of conjunctions $\mathcal{C}$ (line 15). If not, then we can attempt to extend the current set of candidates $\mathcal{C}$ (while loop that starts at line 1) until we cannot extend them further or keep generating the same set of candidates.

The function `extend_pairs()` is described in Algorithm 5. It takes as input the set of candidates that have been split into overlap components `Extend` and non-overlap conjunctions `To_extended_with`. The algorithm attempts to extend (line

[19] A total of $n(n-1)/2$ checks are performed.

Algorithm 5: extend_pairs(Extend,To_extend_with,β)

Input: Extend set of non-action candidates
Input: To_extend_with set action candidates
Input: β commission error cut-off frequency
Output: Acc set of possibly overlapping candidates

```
1  Acc ← ∅
2  foreach e ∈ Extend do
3      Extended_with' ← ( Extend \ {e} ) ∪ Extended_with
4      Extents ← combine_pairs(e, Extended_with', β)
5      Extents ← greedy_set_cover(e, Extents, α)
6      if |Extents | > 2 then
7          Acc ←Acc ∪ Extents
8      endif
9  endfch
10 return Acc
```

4) each overlap component e in `Extend` (line 2) with the set `Extended_with'` that consists of every other overlap component and the non-overlap conjunctions (line 3). From the resulting set of combined conjunctions `Extents`, only the minimum subset of those conjunctions that once combined have the same support as the original conjunction e, are selected. Once again a greedy heuristic is used to select these conjunctions (line 5). If at least two such extended conjunctions exist (line 6) then they are recorded in the accumulator `Acc` (line 7). Note that the requirement that at least two new extended conjunctions have the same original support, ensures that we do not inadvertently combine non-overlap conjunctions, which would otherwise result in over-fitting.

Algorithm 6 takes as input a single overlap component `To_extend` and a set of candidates `Extend_with` and returns a set of conjunctions, some of which may represent possible overlap operators. Each conjunction e in `Extend_with` (line 2) is used to extend the conjunction `To_extend` (line 3). Note that the function `extend_if_possibly_co_occur()` unlike the function `extend_if_co_occur()` in Algorithm 3, returns the set of locations `db` of any combination of the conjunctions if their exists at least one transaction in which they simultaneously occur (co-occurrence is not a requirement). The set `db` is then split into the sets of conjunctions that are `linked` and `unlinked` (line 4). If any `linked` conjunctions exist (line 7), then they are recorded in the accumulator `Acc` (line 8), which

Algorithm 6: combine_pairs(to_extend,Extended_with,β)

Input: To_extend overlap component
Input: Extend_with set of overlap components and non-overlap candidates
Input: β commission error cut-off frequency
Output: Acc set of possibly overlapping candidates

```
1  Acc ← ∅
2  foreach e ∈ Extend_with do
3      db ← extend_if_possibly_co_occur(To_extend, e, β)
4      linked, unlinked ← split_list(db)
5      linked ← commission_error_filter(linked, β)
6      unlinked ← commission_error_filter(unlinked, β)
7      if linked ≠ ∅ then
8          Acc ←Acc ∪ linked
9      endif
10 endfch
11 return Acc
```

is then returned as the result (line 11). Note that unlike Algorithm 3 no verification is made to ensure that the original locations (`idx`, `pos`) of `to_extend` are are still covered. The equivalent check is performed by the the `greedy_set_cover()` function (Algorithm 5, line 5).

Algorithm 7 once again uses a greedy heuristic to identify the minimum set of overlapping candidates `Extents` that have the same cover as the original extended conjunction `To_extend`. It starts off by first ordering the `Extents` according to a given criteria (line 2) and calculating the cover `Must_cover` of the conjunction `To_extend` (line 3). The function $S(\mathbf{c})$ represents the support set of the conjunction $\mathbf{c}$. It then selects one of the extents e at a time (line 5) and removes its coverage `Covered` (line 7) from the required covering `Must_cover` (line 8). The selected conjunction e is recorded in the accumulator `Acc` to be retuned later (line 9). The process is repeated until either all of the required cover has been provided by the set of conjunctions in `Acc` or we run out of candidates (line 4). If the required coverage can be guaranteed (line 11) then the accumulator is returned (line 12) otherwise an empty set is returned (line 14).

Several orderings (line 2) were tested. The one that provided the best results places those conjunctions that provide the correct operator definition first, it places second those with the largest cover value and last the conjunctions with least number

Algorithm 7: greedy_set_cover(To_extend,Extents)

Input: To_Extend overlapping component that was extended
Input: Extents set of possibly overlapping candidates
Output: Acc minimum cover set of possibly overlapping candidates

```
1  Acc ← ∅
2  Extents ← sort_greedy_heuristic (Extents)
3  Must_cover ← {location_i | location_i ∈ S( To_Extend )}
4  while ( Must_cover ≠ ∅ ) ∧ ( Extents ≠ ∅ ) do
5      e ← next (Extents)
6      Extents ← Extents \ e
7      Covered ← {location_i | location_i ∈ S(e)}
8      Must_cover ← Must_cover \ S(e)
9      Acc ← Acc ∪ {e}
10 endw
11 if ( Must_cover = ∅ ) then
12     return Acc
13 else
14     return ∅
15 endif
```

of literals. No tests that ordered by support value were assessed because the greedy algorithm was only tested with coverage and not support.

3.3.4 Dealing with Noise

It is desirable that the ML algorithm be robust enough to deal with noise in the data set. Noisy data may be present in the set of transactions as a result of two types of errors: errors of commission and errors of omission. Errors of commission represent the existence of literals in the transactions that should not be present. Errors of omission represent literals that are missing from the transactions. A combination of errors of omission and omission may also be present, in other words a correct literal may be substituted by an incorrect one. A literal is incorrect if at least one of its arguments is incorrect. In this section we will show how the algorithm must be adapted in order to deal with errors of omission and commission.

It is important to note that the errors of commission do not necessarily represent errors *per se* in the data. Transactions will naturally hold many literals (facts) that are irrelevant to the actions that are executed. For example in a mobile robot domain, the positions of the walls and other fixed objects may never change (non

fluents that are not part of any operator's effects) nor provide pre-conditions for the execution of some actions (non fluents that are not part of any operator's post-conditions). So these literals should not be part of any of the operators' definitions. We will nevertheless treat this case as a common error of commission.[20]

Errors of Commission

Let's assume that we have at least one transaction consisting of the literals $\langle c_1, c_2, p_1, p_2, e_1, e_2 \rangle$ where the (possibly qualified) literals c_i represent commission errors and p_j and e_k an operator's pre and post-condition respectively. The ML algorithm should therefore only identify the operator $(p_1 \wedge p_2) \rightarrow (e_1 \wedge e_2)$. However, as it stands, the algorithm above will generate the following two consistent conjunctions: $\langle c_1, c_2 \rangle$ and $\langle p_1, p_2, e_1, e_2 \rangle$. To explain why, consider that if $(p_1 \wedge p_2) \rightarrow (e_1 \wedge e_2)$ is the correct operator then there must exist at least one transaction $T(i)$ such that $\langle c_1, c_2 \rangle$ and $\langle p_1, p_2, e_1, e_2 \rangle$ do not co-occur. Hence both these conjunctions will be identified and retained as candidates. Because both candidates are required to cover all of the transaction literals, neither is removed during the unique cover check. This means that all commission error maximals that exist are kept in the final solution. Moreover, assuming a constant error rate and uniform distribution of these errors, as the number of transactions increases so does the number of such maximals.

Two strategies were used to avoid such errors. First we observe that in the case of commission errors due to the occurrence of possibly frequent non-fluents, we need only to remove all those conjunctions that cannot be interpreted as an operator. We therefore include a last call in Algorithm 1 (line 8) to a new function `are_valid_simplified_operator()` which checks that each candidate has the following conditions, otherwise it is are remove:

- At least one pre-condition must exist (at least one non qualified literal must exist),
- At least one post-condition must exist (at least one qualified literal must exist),

[20] Non-fluents will never be qualified with the `add/1` and `del/1` predicates. However we treat all commission error literals, qualified or not, in the same way.

- All delete post-condition (literals qualified with the `del/1` predicates) must exist in the pre-conditions,
- All add post-condition (literals qualified with the `add/1` predicates) must exist in the post-conditions,
- No post-conditions clobber each other (the same function symbol does not exist in any literals qualified with both the `add/1` and `del/1` predicates).

Second, we observe that in general the support values of the commission error conjunctions are low. In such cases we can use *a priori* information to establish a cut-off frequency, which we will refer to as β. Any maximal whose support value is not greater than β is discarded. Note that the value of β should be a function of the number of transactions because as the number of transactions increases so does the chance that two or more literals will co-occur. Experiments show however that even though the number of commission error conjunctions increases linearly their support value remain low. So the use a of a constant β is quite effective. It has also been observed that the algorithm remains relatively stable with respect to changes in the β values.

In order to incorporate the β parameter into the ML algorithm changes are required in the `get_maximal_clauses()` (Algorithm 2, line 7), `extend_co_occurring_literals()` (Algorithm 3) and `combine_pairs()` (Algorithm 6) functions. In the `get_maximal_clauses()` we simply check that all maximals must have a support value larger than that of β. Note that even if the maximal is not recorded, its cover is still removed to ensure convergence. In the case of the `extend_co_occurring_literals()` the `extend_if_co_occur()` function (line 5 in Algorithm 3) an additional parameter β is added. This function performs an initial test to check if $S(\mathcal{C}) \subseteq S(\texttt{lit})$ or $S(\texttt{lit}) \subseteq S(\mathcal{C})$. If neither is true then the conjunctions do not co-occur and are therefore not combined. If they do, then the upper bound[21] $S_0 = S(\mathcal{C}) \cap S(\texttt{lit})$ is calculated. If the $|S_0| \leq \beta$ then we assume the possibly combined conjunction is noise and an empty set is returned. Oth-

[21] This is an upper bound because the conjunctions may be unlinked or produce sets of non-variants.

erwise all locations of all possible combinations of these conjunctions are determined and returned grouped by non-variants. Note that once we do extend the conjunction we may find two or more non-variants, each with a different support value less than the upper bound $|S_0|$. For example consider the conjunctions $\mathcal{C} = \langle p(X,Y)\rangle$ and $\texttt{lit} = q(X,Y)$. Their combination may result in the non-variant extensions $\mathbf{c_1} = \langle p(X,Y), q(X,Y)\rangle$ and $\mathbf{c_2} = \langle p(X,Y), q(Y,X)\rangle$, $\mathbf{c_3} = \langle p(X,Y), q(Z,W)\rangle$[22] such that $S_0 = S(\mathbf{c_1}) \cup S(\mathbf{c_2}) \cup S(\mathbf{c_3})$ and hence $S_0 \supseteq S(\mathbf{c_1}) \cup S(\mathbf{c_2})$. This means that once we do determine the extensions and their correct support values we must repeat the check for commission errors (lines 9 and 10 in Algorithm 3).

The changes in the `combine_pairs()` (Algorithm 6) function are similar. The function `extend_if_possibly_co_occur()` (line 3) also has an additional parameter β which is used to remove all extensions whose support is below this value. However unlike the previous version of `extend_if_co_occur()` this function only checks that an $S_0 = S(\mathcal{C}) \cap S(\texttt{lit})$ is not empty because the generation of overlap operators does not require (full) co-occurrence. Once the extensions have been determined it also performs a final pruning of commission errors (lines 5 and 6).

Errors of Omission

Dealing with errors of omission is significantly more challenging. In order to understand why assume we have an operator $(p_1 \wedge p_2) \rightarrow (e_1 \wedge e_2)$. In other words we expect to find the consistent conjunction $\mathbf{c_1} = \langle p_1, p_2, e_1, e_2\rangle$. Because of errors of omission and given enough transactions we may nevertheless find all subsets of $\mathbf{c_1}$. For example if for a given transaction the literal p_1 is missing then we will find the consistent conjunction $\langle p_2, e_1, e_2\rangle$. Note that any number of literals may be missing from an action thereby exacerbating the problem. In the worst case scenario we will identify each unique literal of $\mathbf{c_1}$ as a consistent conjunction.[23]

The ML algorithm therefore exhibits a bias towards shorter consistent conjunctions because only a single transaction with an error is required to prematurely

[22] Unlinked extension.

[23] This is possible even if only one literal per action is missing. For this to occur in the case of an operator consisting of n literals, we need only have n transactions, each of which having one different literal missing.

terminate the extension of a conjunction. However because errors are fairly uncommon we have observed that the relative difference in support between a conjunction $\mathbf{p}$ and any of its error induced subsuming conjunctions $\mathbf{p}'$ can be used to force the extension of the subsuming conjunction. This may be formally expressed in the following way. Let $\mathbf{p}$ be the maximal we expect to be consistent, $\mathbf{p}'$ be the error induced consistent conjunction such that $\mathbf{p}' \prec \mathbf{p}$ and $\mathbf{e} = \langle e_1, \ldots, e_k \rangle$[24] represent the missing literals ($\mathbf{p} = \mathbf{p}' \cup \mathbf{e}$). We can now calculate the relative support for $\mathbf{p}$, $\mathbf{p}'$ and $\mathbf{e}$ as follows: $S_0 = S(\mathbf{p})$, $S_1 = S(\mathbf{p}') \setminus S(\mathbf{p})$ and $S_2 = S(\mathbf{e}) \setminus S(\mathbf{p})$. If $|S_1| \ll |S_0|$ or $|S_2| \ll |S_0|$ then we may assume that $\mathbf{p}'$ is error induced and must therefore be extended further with $\mathbf{e}$. Once again we introduce a parameter α that quantifies the relative difference of support that will force further extending of a conjunction. Using this parameter we say that if $S_1/S_0 < \alpha$ or if $S_2/S_0 < \alpha$ then the consistent conjunction $\mathbf{p}'$ is extended further.

In order to deal with omission errors, the first change requires that we force the extension according to the conditions above. The function `extend_if_co_occur()` (Algorithm 3, line 5) performs an initial check on the upper bounds of the support. Assuming $\mathcal{C}$ is the conjunction to be extended with the literal `lit`, it determines the supports: $S_0 = S(\mathcal{C}) \cap S(\texttt{lit})$, $S_1 = S(\mathcal{C}) \setminus S_0$ and $S_2 = S(\texttt{lit}) \setminus S_0$. If $|S_0| > 0$ and either $|S_1| > 0$ or $|S_2| > 0$, but not both,[25] then we assume that co-occurrence may be possible and all locations of all combinations of $\texttt{extended} = \langle \mathcal{C} \cup \{\texttt{lit}\} \rangle$ are identified. If on the other hand, $|S_0| > 0$ and both $|S_1| > 0$ and $|S_2| > 0$ then we calculate the ratio $r = \min(|S_1|, |S_2|)/(|S_0| + \min(|S_1|, |S_2|))$. In this case if $r \leq \alpha$ then we assume that omission errors have been detected and the locations of all possible co-occurring combinations of `extended` are identified. For all other cases an empty set is returned.

Once the correct support values of the co-occurring combinations have been determined, we must repeat the check above (Algorithm 3, lines 11 and 12). Here we assume that we only have the initial conjunction $\mathcal{C}$ and the set of `linked` and `unlinked` conjunctions consisting of the extensions $\texttt{extended}(i)$. The sup-

[24] Possibly a literal.

[25] Both may be empty.

ports $S_0(i) = S(\texttt{extended}(i))$ and $S_1 = S(\mathcal{C})$ are evaluated and the ratio $r(i) = |S_1|/(|S_0(i)| + |S_1|)$ is calculated. For every $\texttt{extended}(i) \in \texttt{linked} \cup \texttt{unlinked}$, if $r(i) \leq \alpha$ then we assume that omission errors have been detected and the extension is recorded otherwise it is discarded.

The changes above can successfully generate non-consistent maximals that represent the most probable definitions of an operator. However it has several unintended repercussions, most notably in regards to the selection of the literals that possibly co-occur with the seed and the resulting convergence of the greedy covering heuristic. To see why, recall that in the function `get_maximals_clauses()` (Algorithm 2) an uncovered seed is selected and only extensions that cover its original location are kept. This means that the greedy cover algorithm will keep selecting uncovered literals until, sooner or later, none exist. This is possible if we assume that the seed's transaction contains all of the literals that constitute the consistent conjunctions. In fact this is why we selected the shortest transactions first. With errors of omission this is simply not true, one or more of the required literals may be missing.

In order to solve this problem we select the transaction that contains a seed literal according to an alternate criteria: select the longest transaction. In this way we increase the chances of finding all of the maximal's literals. Alternative strategies included selecting transactions with the greatest number of co-occurring literals or greatest number of unique[26] co-occurring literals. However the quality of the solutions in these cases was generally worse. It is important to note that even if we do select the longest transaction, we must still select a seed from that transaction (function `rest_of_literals()` in Algorithm 2, line 5 must now return a new seed (`idx`, `pos`)). Such a seed may inadvertently be linked to a maximal that is missing one or more literals yielding the wrong solution.

Assuming we do select a transaction and respective seed that lead to the correct (possibly consistent) maximal, then only their locations are marked as visited (Algorithm 2, line 10). This means that all literals of the subsuming error induced conjunctions will not be covered. In the next rounds, the greedy algorithm will pick

[26] We do not count variants of the same literal.

the same uncovered seed, which results in the selection of the longest transaction and seed that was previously chosen, thereby generating the same maximal. The process is then repeated indefinitely and therefore fails to terminate. In order to solve this issue, for any given seed, we only select the longest transaction if it contains at least one variant of the seed not yet covered. In this way, when we do pick the same literal again, we will not repeatedly select the same (longest) transaction. However this means we will not only generate the correct (possibly consistent) maximal but also all error induced consistent conjunctions. This significantly increases the complexity of the algorithm. One may consider avoiding repeatedly selecting the initial seed literal (or extending conjunction $\mathcal{C}$) by marking it as visited as soon as we detect a possible error of omission. This would avoid the needless generation of all the error induced conjunctions, however it produces worse results because it means that we may unintentionally remove shared literals that would otherwise lead to the identification of other correct (possibly consistent) maximals.

We can now see that when errors of omission exist, the set of generated candidates will also include a number of error induced subsuming conjunctions. These incorrect conjunctions are eliminated in the `learnOperators()` function (Algorithm 1, line 6) prior to the removal of non unique covering conjunctions. It is important to point out that the removal of the error induced subsuming conjunctions must be done prior to the filtering of non-unique covering conjunctions. To understand why consider a maximal $\mathbf{p}$ and all its error induced conjunctions $\mathbf{p_i}$. We know that in these conditions $\forall i : \mathbf{p} \prec \mathbf{p_i}$. Because of the anti-monotonic property this implies that $\forall i : S(\mathbf{p}) \subseteq S(\mathbf{p_i})$. In the worst case scenario we may have the case that $S(\mathbf{p}) \subseteq \bigcup_i S(\mathbf{p_i})$. In other words $\mathbf{p}$ will be removed because it does not cover any unique literal not covered by any other conjunction.

The removal of the error induced conjunctions simply repeats the tests that were already performed during conjunction extension (Algorithm 3, lines 11 and 12) as follows. All candidates are ordered in increasing length of the conjunctions. Each conjunction $\mathbf{p_i}$ is then compared to all of its longer conjunctions $\mathbf{p_j}$.[27] If the

[27] Total of n(n-1)/2 comparisons.

conjunction subsumes some other (longer) conjunction ($\mathbf{p_i} \prec \mathbf{p_j}$) then the supports $S_0 = S(\mathbf{p_i})$ and $S_1 = S(\mathbf{p_j})$ are evaluated and the ratio $r = |S_1|/(|S_0| + |S_1|)$ is calculated. If $r \leq \alpha$ then we assume that omission errors have been detected and the conjunction $\mathbf{p_i}$ is discarded. In any other case the conjunction $\mathbf{p_i}$ is kept.

Similar changes must also be made to the functions that deal with the overlap operators. Accordingly `extend_pairs()` has an α parameter that is only used by the `greedy_set_cover()` function (Algorithm 5, line 5). The changes to the `greedy_set_cover()` function are limited to the `if` statement in line 11 of Algorithm 7. Instead of verifying that all of the support of the `To_extend` conjunction is reached, we assume that due to errors part of this support need not be guaranteed. We therefore stipulate a maximum support value `can_miss` that can be ignored such that `can_miss` $= \alpha|S(\texttt{Must\_cover})|$. The `if` statement's condition is simply changed from $S(\texttt{Must\_cover}) = \emptyset$ to $|S(\texttt{Must\_cover})| \leq$ `can_miss` thus allowing for some of the support to be missing.

Unlike `extend_if_co_occur()` the function `extend_if_possibly_co_occur()` in line 3 of Algorithm 6 does not require any changes. Recall that in this case we select only combinations of conjunctions that need only occur together at least once. So no relative measure of co-occurrence error is available to be used with the α parameter.

The removal of the error induced conjunctions in the function `learnOperators()` (Algorithm 1, line 6) means that the literals of these conjunctions need not be covered by the solution set of consistent and overlap conjunctions. As with the `greedy_set_cover()` function above this means that both `purge_non_unique_cover_clauses()` (in Algorithm 1, line 7) and `purge_super_non_unique_cover()` (in Algorithm 4, lines 8 and 13) must also be changed to allow for some of the literals not to be covered. A new parameter γ was therefore introduced that allows us to indicate the maximum number of literals that need not be covered. It is important to point out that the same parameter α was not used because it represents a fraction of the support value that does not need to be covered, whereas the γ values is relative to the cover of the literals. The value of this parameter depends on the length of the operators. If the average length of the operators is higher,

then the number literals that need not be covered will also increase irrespective of the support value. Experimentation show that in most cases a constant value of approximately $\gamma = \alpha/10$ works well.

In both functions above we define a value `miss_no_more`(i) such that it is equal to $\gamma|\bigcup_{i \neq j} C(\mathbf{p_j})|$ for all $\mathbf{p_j}$ candidates excluding the candidate $\mathbf{p_i}$ whose unique coverage is being tested. The value `miss_no_more`(i) is calculated for all $\mathbf{p_i}$ in the set of candidates. If any of the $\mathbf{p_i}$ conjunctions covers less that `miss_no_more`(i) literals, then we assume that it does not have a unique cover. Such a conjunction is discarded from the final solution. Recall that essentially the only difference between these two functions is the order in which the literals $\mathbf{p_i}$ are tested.

A final note on the parameters α and γ. Like β these values should not be constant but should increase with the increase in the number of transactions. However, experimentation has shown that the use of constant values works quite well.

3.4 Concluding Remarks

The algorithm described in this chapter addresses the solution of an NP-complete problem. Moreover we have identified three situations: learning standard operators in a noise free settings; learning overlap operators in a noise free setting; and learning in a noisy setting. As is usual in these cases, because efficiency is at stake, we have adopted an heuristic approach. A worst case exponential cost still occurs in both: the search for standard operators (see Algorithm 3); and overlap operators (see Algorithm 4). In the first case we have seen that the check for linkage between literals may result in an exponential cost in the search. In addition to this it is important to note that in both these phases of the algorithm we must identify all non-variants of an extended clause and their locations. In order to check if two clauses are or are not variants we must match these clauses, which is also NP-complete.

We have also seen that in order to guarantee convergence in the presence of noisy data, all clauses, including those candidates that will later be removed from the solution, must be generated and evaluated. This means that a greater num-

ber of clauses are generated thereby increasing the the computational cost of the algorithm. We have also seen that dealing with errors of omission is particularly difficult because missing literals will generate multiple maximal clauses. These clauses must be tentatively extended further in order to check for small changes in relative support values. Such checks also increase the cost of the search.

Chapter 4

Experimental Evaluation

4.1 Introduction

In this chapter we present the design, execution an analysis of the experiments that empirically demonstrate the effectiveness of the algorithms. This description includes details on the planning domains used and parametrization selected in the generation of the data sets for each domain. The analysis of the results provides details on the error rates used to measure the learning effectiveness of ML algorithm and reports on the error values obtained from the execution of experiments. A general review of all the experiments are provided, but only two example graphs are shown. The full set of graphs is available in Appendix B. We end this chapter with a discussion of the results obtained.

4.2 Domains

In order to evaluate the effectiveness of the ML algorithm, a set of experiments were performed using 4 different domains. The first 3, planning domains, were obtained from the Sixth International Planning Competition (IPC).[1] The planning domains

[1]The 2008 IPC can be found at `http://www.icaps-conference.org/index.php/Main/Competitions`. We used the competition and example domains from the deterministic planning track found in `http://ipc.informatik.uni-freiburg.de/Domains?action=AttachFile&do=view&target=ipc2008-no-cybersec.tar.bz2` and `http://ipc.informatik.uni-freiburg.de/HomePage?action=AttachFile&do=`

used are the Mars Rover Domain ([ECA+03]), the Elevator domain ([KS00]), and the Woodwork[2] domain. An additional workflow specific domain was also included, which describes a greater part of a bridge inspection process. We refer to this domain as the Bridge Inspection domain. The description of this process is found in Appendix A.1.4, which was copied verbatim from the Office Of Maintenance of the Florida Department of Transportation. [3] All original STRIPS operators are described using an equivalent material implication. The antecedents describe the operator's pre-conditions and the consequent the operator's post-conditions. In the case of STRIPS operators whose pre and post-conditions are not conjuncts, a set of two or more material implications, which are equivalent, were generated. The full set of definitions of the operators of all the domains that were used in generating the data sets is found in Appendix A.1.

The Mars Rover domain describes the actions performed by the Mars Rover as it roams around Mars' surface sampling soil and rocks and photographing the terrain in order to identify possible regions of geological interest. It includes operators to navigate the rover, take samples, take photographs and communicate this data back to the lander. No changes were made to the original IPC definitions. In all it consists of 9 operators, made up of a total of 63 literals (38 literals in the non-variant set). Each operator has an average of 7 literals. This domain is characterized by a large difference in the number of literals in each operator (one operator has the maximum of 9 literals, another operator with the minimum of 2 literals) and the reduced number of literals in the operator's post-conditions (average number of add post-conditions is 1.2, average number of delete post-conditions is 0.7, maximum number of post-conditions is 4 literals). Even though this domain seems simple it has two sets of operators that are very similar and differ only in two literals (sampling operators are similar save for the types of samples, the communication operators are similar save for the type of data sent; both sampling and communications operators

---

`view&target=example-domains.zip` respectively. These domains were changed slightly. These changes are also described.

[2] `http://ipc.informatik.uni-freiburg.de`

[3] `http://www.dot.state.fl.us/statemaintenanceoffice/CBR/Bridge\%20Inspection\%20Process2.pdf`

also share literals). This allows us to test the algorithm's effectiveness in determining all relevant literals even if shared amongst different operators.

The Elevator domain describes the operators that move a lift up or down between two given floor counts and allow people to board and/or leave the lift. No changes were made to the original IPC definitions. This seemingly simpler domain has a mere 6 operators consisting of a total of 45 literals (20 literals in the non-variant set); however all operators have either 7, 8 or 9 literals, 4 of which are post conditions (on average each operator is therefore more complex). In addition to this, the number of shared literals is higher. For example, all 4 operators that describe the lift moving up/down fast/slowly differ in just two literals. In the case of the board and leave operators, these just differ in 4 literals, one of which simply has a different ordering of the arguments. The greater number of shared literals is used to test the ability for the algorithm to identify very similar operators (which includes non-deterministic operators).

The Woodwork domain describes the operators used for cutting wood parts and in preparing and finishing the wood parts so that they can later be combined into finished goods. The wood parts may be cut or sawed into three standard sizes (small, medium or large). Any wood part that is cut with a saw must be loaded to and unloaded from an automated saw machine. The wooden parts may be prepared by either grinding or planing. Finally wood parts may be given a varnished (immersion or spray) or glazed finish. This domain represents a significant increase in complexity compared to the previous ones. It has 13 operators consisting of a total of 164 (2.6 times more than the Mars Rover domain) literals (57 literals; 1.7 times more than the Mars Rover domain). This domain's operators are also more complex in that each consists of an average number of 12.6 literals (versus the 7.5 of the Elevator domain; 1.68 times greater).

This original domain uses constants, such as `small` and `large` in order to differentiate between the various similar operators. The ML algorithm does not consider the use of constants (all transactions are variabalized prior to processing). In order to deal with this, the literals that use such constants are either renamed or the the constant themselves are transformed into function symbols. For example

Listing 4.1: Bridge Inspection Domain: overlapping actions.

```
: check_girder_1(B:bridge, beam(S:support_structure))
 made_of_wood(B, S) *
 checked_for_woodrot(B, S) *
 checked_for_crushing(B, S) *
 checked_for_splitting(B, S) *
 checked_for_cracking(B, S)
 =>
 checked_girder(B, beam(S)).

: check_girder_2(B:bridge, beam(S:support_structure))
 made_of_concrete(B, S) *
 checked_for_cracking(B, S) *
 checked_for_spalling(B, S) *
 sounded_for_hollow_areas(B, S)
 =>
 checked_girder(B, beam(S)).

: check_pier_caps_1(B:bridge, pier_caps(S:support_structure))
 made_of_wood(B, S) *
 checked_for_woodrot(B, S) *
 checked_for_crushing(B, S) *
 checked_for_splitting(B, S) *
 checked_for_cracking(B, S)
 =>
 checked_pier_caps(B, pier_caps(S)).

: check_pier_caps_2(B:bridge, pier_caps(S:support_structure))
 made_of_concrete(B, S) *
 checked_for_cracking(B, S) *
 checked_for_spalling(B, S) *
 sounded_for_hollow_areas(B, S)
 =>
 checked_pier_caps(B, pier_caps(S)).
```

the literal `goalsize(P, small)` is changed to `goalsize_small(P)` and the literal `increase(Total_cost :cost, 30)` is changed to `increase(Total_cost :cost, immersion_varnish_cost(X :part))`. No additional changes were made to the original domain.

The Bridge Inspection domain describes the operators used in a bridge inspection process. It is made up of 80 operators consisting of a total of 349 (2.1 times more than the Woodwork domain) literals (164 literals in the non-variant set; 2.9 times more than the Woodwork domain). These operators are, however, less complex in that they are made up of fewer number of literals (average of 4.4 versus the 12.6 of the woodwork domains and the 7.5 of the Elevator domain). The operators are divided into four main groups: inspecting the riding surface, inspecting the superstructure, inspecting the sub-structure and generating the final recommendations. Each of these main groups contains a set of fairly short non-deterministic operators. In addition to this, many of these non-deterministic operators are required to

Listing 4.2: Bridge Inspection Domain: overlap components.

```
Component(A):
0:checked_for_splitting(_0:bridge, _1:support_structure)
1:made_of_wood(_0:bridge, _1:support_structure)
2:checked_for_crushing(_0:bridge, _1:support_structure)
3:checked_for_woodrot(_0:bridge, _1:support_structure)
4:checked_for_cracking(_0:bridge, _1:support_structure)

Component(B):
0:sounded_for_hollow_areas(_0:bridge, _1:support_structure)
1:made_of_concrete(_0:bridge, _1:support_structure)
2:checked_for_spalling(_0:bridge, _1:support_structure)
3:checked_for_cracking(_0:bridge, _1:support_structure)

Component(C):
0:add(checked_girder(_0:bridge, beam(_1:support_structure)))
1:checked_for_cracking(_0:bridge, _1:support_structure)

Component(D):
0:add(checked_pier_caps(_0:bridge, pier_caps(_1:support_structure))),
1:checked_for_cracking(_0:bridge, _1:support_structure),
```

satisfy the pre-conditions of the operators across the four main groups. For example the pre-condition of checking for cracking is common to operators for checking the riding surface, girders, pier caps, columns and piles. These conditions are also used irrespective of whether or not the material of the element is made of wood or concrete (see Listing 4.1 for some examples). In addition to the above, this domain also contains fully overlapping operators (see Definition 3.2.20 and Section 3.3.3 for details). In other words two or more actions exist wherein both the same pre and post conditions occur. In this case the common pre and post conditions do not always co-occur but may still be combined to form correct operator definitions. To make things clear the example in Listing 4.1 shows the definition of two operators[4] `check_girder(B :bridge, beam(S :support_structure))` and `check_pier_caps(B :bridge, pier_caps(S :support_structure))`. Each of the four operator definitions is made up of a combination of two of the four overlap components shown in Listing 4.2. It is important to note that a correct operator definition may consist of two or more overlap components and that a domain (such as the Bridge Inspection) may contain both non-overlap and overlap operators (which do not necessarily have common overlap components).

[4]The naming convention for the operators uses the same base name operator but adds a post-fix containing a unique numerical identifier. Unique post-fix identifiers are used to group non-deterministic or otherwise similar actions. For example the same check actions that is done for different materials.

Appendix A.2 provides a table that shows some basic statistics regarding the domains described in this section.

4.3 Experimental Parameters

The aim of the experiments is to determine if we can identify all of the domain operators given a data set consisting of a fixed number of transactions. More specifically we wish to know if all operators can be automatically acquired in the limit (in respect to number of transactions) and if so under what conditions. The first parameter is therefore the number of transactions. The ML algorithm was applied to a number of data sets each consisting of an increasing number of transactions. We start with a data set of 10 transactions and increase each new data set with an additional 10 transactions. The maximum number of transactions used in an experiment is the number of transactions at which all operators are successfully acquired.[5] Experiments may have data sets with a maximum number of transactions that vary anything between 170 and 1500.

For a given set of parameter values the experiments were repeated a fixed number of times. Each time a new set of transactions is randomly generated. Due to the lengthy processing time required by the ML algorithm, only 5 such random repetitions were executed. The results are reported according to the average of these 5 runs.

We are interested in investigating whether and how the number of actions and their distribution influence the efficacy of the learning algorithm. For that purpose two additional experimental parameters were used:[6] the number of actions concurrently executed per transaction and the distribution of the executing actions. The following combinations of these parameters have been used: a uniform distribution with a constant number of actions per transaction (uniform n, we have set $n = 4$),

[5] We usually extend this value to ensure that the results are stable. In other words the error remains at its minimum.

[6] Initial experiments used an additional parameter to alter the order in which the literals in the transactions are analysed. The Fisher-Yates shuffling algorithm was used to this end. No changes were detected in the results and this test was therefore dropped.

a uniform distribution with a varying number of actions per transaction (uniform n to m, we have set $n = 1$ and $m = 5$), a non-uniform distribution with a constant number of actions per transaction (non-uniform n, we have set $n = 4$) and a non-uniform distribution with a varying number of actions per transaction (non-uniform n to m, we have set $n = 1$ and $m = 5$).

Many non-uniform distributions are possible, however, according to research in the workflow area,[7] process descriptions usually exhibit an $80\% - 20\%$ distribution in the use of the process modelling constructs. We have therefore also used the same distribution for our experiments. More specifically in each domain, a set of 20% of the operators were selected for the generation of 80% of the actions in each transaction. The rest of the operators were assigned to actions a mere 20% of the time. We have nevertheless retained a uniform distribution of actions within each of these two groups. For example in the Mars Rover domain, of the 9 operators, 2 where assigned a higher probability of occurrence (both `navigate(X:rover, Y:waypoint, Z:waypoint)` and `calibrate(R:rover, I:camera, T:objective, W:waypoint)` were given a probability of 0.4 of occurring. Each of the remaining operators are given a probability of 0.029 of occurring.).

Note that we are still left with the choice of selecting the most appropriate operator definitions. We have not used any specific criteria, such as operator length or number of arguments, but have chosen these intuitively. In the case of the Mars Rover domain for example it seems fairly obvious that the rover will invariably navigate from point to point more often than it will either take samples or send sample analysis results to the lander.

Actions that execute simultaneously may naturally share common state relations. Many of these relations represent non-fluents (in other words relations in the world state that are not changed by the operators). For example two robots in the same space may detect the same obstacles, whether they are fixed (such as

[7] `http://www.bpm-research.com/2008/03/03/how-much-bpmn-do-you-need/`. See also `http://www.razorleaf.com/2009/12/business-process-change-3/` and `http://en.wikipedia.org/wiki/80/20_rule`

walls) or not (such as people). We therefore use two parameters that allow us to identify which literals may be shared amongst actions and the maximum number of instances of such literals that can be shared in a single transaction. For example in the case of the Mars Rover domain the intersect list contains the following literals: `visible(Y:waypoint, Z:waypoint)`, `at(R:rover, Z:waypoint)`, `at_lander(L:lander, Y:waypoint)`.[8] In this case the maximum intersects per transaction was set to 4. Each domain has its own set of intersect literals but all domains have a maximum number of literal intersects set to 4.

In real world problems we expect each transaction to contain state data that may not be pertinent to the definition of one or more of the operators. We are therefore interested in knowing if the algorithm can avoid including such extraneous, albeit related, data in operator definitions and hence prevent overfitting. For example in a robot domain the operator definition describing how to open a door need not include the relations regarding the lighting switch that's on one of the door's support walls. Note that in general to avoid such overfitting may not be possible because this can require additional information, inference or assumptions which may not be available. For example no information may be available that allows us to infer that the light switch is not relevant to opening a door.[9] We include this extraneous data as errors of commission in the transaction's pre-image only (See Section 3.3.4). We therefore include a parameter that indicates the rate of commission errors and if such commission errors are limited to the pre-image only.

As with any research in ML we are also interested in knowing if the algorithm is robust in the presence of noise. However we have several problems in determining what are the expected error types and rates. This may depend on the domain itself or even be unknown when considering human actions (which is our primary focus). Consider the following error types. We may have errors at the level of relations: some relations may be missing (errors of omission) or may represent extraneous

[8]Note that we have not restricted the use of these literals to non-fluents nor to the set of pre-conditions only.

[9]Without additional inference the only way we can learn that the light switch is irrelevant is if we have at least one example of opening a door wherein none of its supporting walls has a light switch.

relations (errors of commission). Even if the relations are correct they may also contain incorrect argument values (such as typographical errors in the terms or correct but slightly different values such as numeric values with limited precision).

The first question is: are these errors limited to the pre-image only? In the case of a robot for example, whose proximity sensor has temporarily failed, attempting to traverse a wall will (almost certainly) never succeed. So in such a case, no post-image state will be affected by such an action[10] and hence we need not consider post-image errors in this domain. In human centred processes this may not be the case. Take a (paperless) claim processing workflow for example. Assume that all of a case's data has been correctly pre-processed prior to verifying the policy terms of the claim. If an adjudicator reviewing the claim incorrectly forgets to check a legal requirement, then it is erroneously signed and routed for claim payment. No *Physics laws* apply here which will avoid any invalid post-conditions as they do in the case of the robot crashing into a wall. It is therefore, in general, necessary to consider incorrect post-conditions.

The second issue is what are the expected error rates we should consider. Once again this depends on the domain that is considered. For example robotic actuators and sensors have known predefined maximum error rates that can be used to generate the data set. In the case of human orientated processes this may be unknown. For example, in the paperless claims processing workflow, error rates associated with document scanning may be known, but pre-processing steps such as completing or correcting claims entries, determining the claims type and matching membership to the claims type may introduce an unknown amount of human errors (such as misspelling). The only way to deal with this is to repeat the various experiments with an ever increasing error rate and verify how robust the ML algorithm is.

The injection of errors significantly increases the number of candidates that are generated and therefore the processing time that is required to execute the ML algorithm. Because of this only a limited number of experiments with error were executed. For all of the combinations of parameters described above (fixed and

[10]We may however record a new (incorrect) position which results in a non-deterministic action.

varying number of concurrent actions, uniform and non-uniform distributions and number of transactions), the experiments were repeated in 3 different settings: 10% errors of commission only, 10% errors of omission only and finally a combination of 5% errors of commission and 5% errors of omission.

4.4 Data Set Generation

Each transaction contains a set of literals that represent the simultaneous execution of one or more actions. The challenge is to generate such transactions consistently for a given number of actions that are executed concurrently. The oracle's set of operator definitions are used to generate these transactions. Each such operator may be assigned a real value representing the probability of occurrence of an action of that operator. This allows us to set the distribution of the action executions. All operators that are not assigned such a value are given a default value that ensures the uniform distribution of the respective actions. In order to make this clear let us assume we have a set of 10 oracle operator definitions. Let's also assume that we assign a uniform probability of occurrence of 40% to each of the two actions. This means that the rest of the 8 actions can be assigned a value $(1.0 - 0.8)/8 = 0.025$ as their (uniform) probability of occurrence.

Once the operators for a given transaction have been selected they are instantiated. At this point, both the parameters that contain the list of intersects and the maximum value of intersects per transaction, are used to unify as many shared literals within the same transaction as possible (never surpassing the maximum). This allows us to generate transactions of non-separable actions, which would otherwise be a trivial problem to solve.[11] The final step assigns unique object identifiers to each of the transaction's variables thereby generating a set of ground literals that represents a single change in world state. A list of these ground literals constitutes the basic data set that is then used by the ML algorithm.

Unfortunately because we randomly generate intersecting literals, the resulting

[11]Separable actions means that we can identify each action within a transaction simply by collecting only those literals that are linked to each other.

transaction may not be consistent. We must therefore ensure that those intersecting literals we add, do not (due to the binding of variables) create inconsistent transactions. Every time a literal is selected as a possible intersect and is bound to another variant, two consistency checks are done. First we check that the resulting pre and post conditions of each action are compatible. In other words: all terms of the delete literals must exist as a pre-condition literal, no terms of the add literals must exist as a pre-condition literal and no literal must exist as a term in both an add and delete literal. The second test ensures that no two concurrent actions *clobber* each other. So all pairs of actions are compared to ensure that: the pre-condition of an action is not the negated literal of another action's pre-condition, a term of an add literal of an action is not the same as a delete literal of another action, a positive pre-condition of action is not a term of a del literal of another action, and a negative pre-condition of action is not a term of an add literal of another action.

Although the tests above are fairly complete they cannot guarantee domain specific semantic consistency. A domain's world state consistency does not only depend on the operator definitions themselves, but is also determined by the initial world state of a problem solving episode and possibly even by additional intrinsic background knowledge. To understand this statement let us consider the Mars Rover domain. If we stipulate that only one rover and one Mars lander exist at any one time and that the lander has only one communication channel, then it is not possible for that rover to send two sets of sample data simultaneously. Our transactions however may have the same rover taking various samples simultaneously and even sending data at the same time through the same channel. This should not affect the correctness of the results we obtain because we are only interested in acquiring the operator definitions.

In the experiments designed to assess the robustness of the algorithm we proceeded as follows. The ML algorithm receives a data set consisting of a list of ground transactions. Before it parses and processes this data, it injects a given amount of error into each of the transactions. It first adds errors of commission and then errors of omission as per the corresponding parameters. Note that this step is executed as a pre-processing step to the ML algorithm because the same base data set is pro-

vided to a number of processors, each executing the same ML algorithm, for parallel processing. The aim is to significantly increase experiment execution throughput in order to facilitate experiment analysis.[12]

4.5 Analysing the Experimental Results

In order to measure the algorithm's performance we compare the algorithm's output with the oracle's list of operator definitions. Each of the final conjunctions of qualified literals generated by the ML algorithm are directly compared to all of the oracle's operators, which are also represented as a conjunction of qualified literals. The best of each of these comparisons is then used to evaluate the solution. The average of all of the best comparisons are presented in the results.

Three types of error measures were defined: mean number of incorrect operator definitions (ϵ_1), mean number of incorrect literals in operator definitions (ϵ_2) and the mean number of oracle operators not correctly identified (ϵ_3). The standard deviation of each of these errors is also determined. We compare the results for each of the parameters used in the experiments. We also compare results between the various domains in order to detect common trends in performance.

The evaluation starts off by first matching each of the solution's conjunctions with all of the oracle's operator definitions in order to determine which one is the closest match[13]. At this point the number of matched operator and the number of matched operators literals are also counted. We have the following values at our disposal after matching and counting has been performed:

- **nu_ops**: the total number of oracle operators;
- **tot_candidates**: the total number of candidate clauses generated by the ML algorithm;

[12]Experiments suggest that for some domains (such as the elevator domain) could have their throughput significantly increased if the ground transactions were also distributed amongst processes because much time is used checking and ensuring transaction consistency.

[13]Associative commutative idempotent (ACI) pattern matching is known to be NP-complete ([KN87]). An algorithm that uses Constraints Solving Programming (CSP), which is based on a more general subsumtpion algorithm ([SM10]), is used to find a match. An optimal solution is not guaranteed, but experiments show that the matching are of good quality.

- **tot_num**: the total number of literals that make up oracle operators;
- **matched_props**: the total number of candidate clauses' literals generated by the ML algorithm that match;
- **unmatched_props**: the total number of candidate clauses' literals generated by the ML algorithm that do not match.

The information above is then used to calculate the various errors described below.

$$error_{1a} = \frac{|tot\_candidates - nu\_ops|}{nu\_ops} \tag{4.1a}$$

$$matched\_candidates = tot\_candidates - nu\_unmatched \tag{4.1b}$$

$$error_{1b} = \frac{|matched\_candidates - nu\_ops|}{nu\_ops} \tag{4.1c}$$

$$error_1 = error_{1a} + error_{1b} \tag{4.1d}$$

The *mean number of incorrect operator definitions* (Equations 4.1a, 4.1b, 4.1c, 4.1d) evaluates how many candidates are incorrectly identified. A candidate match is successful only if all of its literals match those of an oracle's operator. The value of $error_1$ (Equation 4.1d) consists of two parts: $error_{1a}$, which measures the total number of incorrect operators (irrespective of a match or not) and $error_{1b}$, which measures the total number of incorrectly matched operators. We have opted to use these two parts in order to correctly identify errors pertaining both to the number of operators that are identified and the number of operators that are correctly identified (for example all of the oracle's operators may be correctly matched but an additional number of candidates may also have been generated that will result in a non zero error). A correct solution will yield a value of 0. In general this error will be the largest of all errors because many operators may be partially identified (some literals

may be missing or additional incorrect literals may be included).

$$matched\_total = matched\_props + unmatched\_props \tag{4.2a}$$

$$error_{2a} = \frac{|matched\_total - tot\_num|}{tot\_num} \tag{4.2b}$$

$$error_{2b} = \frac{|unmatched\_props|}{tot\_num} \tag{4.2c}$$

$$error_2 = error_{2a} + error_{2b} \tag{4.2d}$$

The *mean number of incorrect literals in operator definitions* (Equations 4.2a, 4.2b, 4.2c, 4.2d) evaluates how many candidates literals are incorrectly identified. A candidate match may be partially successful in that only part of its literals match those of an oracle's operator. The value of $error_2$ (Equation 4.2d) consists of two parts: $error_{2a}$, which measures the differences in total expected number of operator literals and $error_{2b}$, which measures the total number of incorrectly matched operator literals. As with the previous error measure, we have opted to use these two parts in order to correctly identify errors relative to the total number of literals and the number of matched literals (for example all operator literals may be matched but additional incorrect literals may also be generated, irrespective of whether or not the correct number of operators is found, will result in a non zero error). A correct solution will yield a value of 0. In general this error is lower than that of the mean number of incorrect operator definitions.

The mean number of incorrect literals in operator definitions uses the counters *matched_props* and *unmatched_props* whose calculation require some care. Four cases should be considered when accumulating these values. Case 1 occurs when all literals of the candidate operators match with those of an oracle's operator. Here obviously only the value of the matched literals is incremented by the number of matched literals.

Case 2 occurs when the set of candidate's literals is a subset of an oracle's operator literals. In this case we assume the candidate operator, as a whole, is incorrect (extraneous) and we therefore only increment the value of the unmatched literals by the number of literals in the candidate.

Case 3 occurs when the set of candidate's literals is a superset of an oracle's

operator literals. Here we consider that the candidate is partially correct (one or more extra literals) and therefore maximize the corresponding error by incrementing only the value of the unmatched literals by the number of unmatched literals in the candidate.

Case 4 occurs when the set of the candidate's literals is neither a super-set nor a subset of an oracle's literals set. In this case we increment both the number of matched and unmatched literals. Note that here we cannot decide whether the candidate as whole is incorrect (case 2) or if only part of the candidate is incorrect (case 4). We have therefore opted to count either the candidates or the oracle's unmatched literals, whichever minimizes the error[14].

The *mean number of oracle operators not correctly identified* indicates the total number of the oracle's operators that are not present in the list of candidate operators. As with the mean number of incorrect operator definitions the operator definition is only correct if all of its literals match exactly. A correct solution will yield a value of 0. Correct solutions will, in principal, allow successful automated planning. However the planning problem increases substantially with the addition of incorrect candidate operators. So these should also be reduced to a minimum.

4.6 Noiseless Data Experiments

We have carried out a series of experiments in order to determine if, for a given number of transactions, all of the oracle's operator definitions are determined without any error, assuming the data is noiseless. The ML algorithm that was used, save for the case of the Woodwork domain, was the algorithm that incorporates error handling. The parameters α, β and γ, which are related to error handling, were all set to 0. The smallest point (number of transactions) after which all three errors remain 0 is referred to as the *stable point* (T_s). It is important to note that we have extended several experiments beyond the point of stability to ensure that the errors

[14]We should either opt to minimize this error to form a lower bound or maximize it in order to consider a worst case scenario. If we minimize then case 3 should minimize the error accumulating the matched literals. If we maximize the error then case 4 should be changed to select the oracle's or candidates unmatched literal set, whichever is is largest.

remain 0. However it is not guaranteed that no error will be found at a later point.

In order to compare the results the following features were used: the number of oracle operators that must be determined (*ops.*), the maximum number of transactions used (T), the stable point (T_s average number of transactions), number of candidate conjunctions identified (*max.*), processing time per transaction (average $t/|T|$ in seconds), length of the transactions ($\Delta|T_i|$ is the average number of literals per transaction) and number of literals that are common to two or more actions ($\Delta|I_i|$ is the average number of literals per transaction). Both $\Delta|T_i|$ and $\Delta|I_i|$ are rough indicators of the size and complexity of the problem. Larger values of $\Delta|I_i|$ indicate a potentially larger number of linked literals per transaction. The values of *max.*, $t/|T|$, $\Delta|T_i|$, and $\Delta|I_i|$ are the mean values of the 5 repeated experiments for the maximum value of T where the solution is best.

Table 4.1 shows the results obtained. First and foremost they indicate that when the data set has no noise, then the operator definitions can be learned without any errors. Note that in the case of non-uniform distributions with varying number of concurrent actions, the results of the Woodwork domain is not conclusive (T_s marked with an asterisk). The Woodwork domain did not reach a stable point and we therefore require larger values of T to confirm stability. This experiment was not extended for lack of time.

The results also show that we do not require an exponential increase in the number of transaction per data set in order to obtain a correct solution. Consider the Woodwork domain, which exhibits the worst performance and the Bridge Inspection domain, which is the largest domain with 80 operators. Even though the Bridge Inspection domain has a little more than 6 times the number of operators than the Woodwork domain, the increase in the number of transactions at the stable point, is in the same order of magnitude.

We can observe that the non-uniform distribution of the actions causes the greatest increases in error. This means that the number of transactions of the stable point increases significantly (value of T_s in the last two rows of each domain). We can also see that this is true irrespective of whether or not the transactions have a constant or varying number of concurrently executing actions. This result is expected because

Parameters	**Elevator**						
	ops.	T	*max.*	$t/\|T\|$	$\Delta\|T_i\|$	$\Delta\|I_i\|$	T_s
uniform, n=4	6	500	6.4	0.078	23.62	6.42	40
uniform, n=1 to m=5	6	500	6.4	0.075	18.40	4.25	20
non-uniform, n=4	6	500	7.4	0.140	25.48	7.29	70
non-uniform, n=1 to m=5	6	500	9.6	0.225	20.07	4.74	60
Parameters	**Mars Rover**						
	ops.	T	*max.*	$t/\|T\|$	$\Delta\|T_i\|$	$\Delta\|I_i\|$	T_s
uniform, n=4	9	1000	11.6	0.053	23.03	5.02	50
uniform, n=1 to m=5	9	1000	17.4	0.053	17.87	3.23	60
non-uniform, n=4	9	1000	12	0.032	18.13	6.88	340
non-uniform, n=1 to m=5	9	1000	23	0.054	14.68	4.20	230
Parameters	**Woodwork**						
	ops.	T	*max.*	$t/\|T\|$	$\Delta\|T_i\|$	$\Delta\|I_i\|$	T_s
uniform, n=4	13	170	24.6	5.38	41.69	8.63	60
uniform, n=1 to m=5	13	170	13.0	1.00	32.51	5.72	90
non-uniform, n=4	13	700	46.6	56.27	46.55	11.93	700*
non-uniform, n=1 to m=5	13	700	13.0	7.70	35.67	8.47	590
Parameters	**Bridge Inspection**						
	ops.	T	*max.*	$t/\|T\|$	$\Delta\|T_i\|$	$\Delta\|I_i\|$	T_s
uniform, n=4	80	1000	75	0.034	15.74	1.71	220
uniform, n=1 to m=5	80	1000	75	0.026	12.03	1.12	290
non-uniform, n=4	80	1000	75	0.026	12.97	0.94	650
non-uniform, n=1 to m=5	80	1500	75	0.019	9.87	0.64	990

Table 4.1: Results of experiments with noiseless data. *Parameters* indicate the distribution and number of actions that are executed per transaction, *ops.* the number of operators in the domain, T the number of maximum transactions in the experiment, T_s the stable point at which all errors become 0, *max.* the number of maximals that were generated, $t/|T|$ average time of execution per transaction in seconds, $\Delta|T_i|$ the average number of literals per transaction and $\Delta|I_i|$ the average number literals that are shared amongst the various concurrent actions.

non-uniform distributions of actions increase the chances that conjunctions, which do not belong to the same actions, co-occur more often. Therefore a larger number of transactions are required in order for such conjunctions not to occur.

Contrary to intuition, experiments wherein a varying number of actions are executed concurrently per transaction, did not obtain lower error rates. The rational behind this is that transactions consisting of less literals will result in fewer co-occurring literals thereby increasing the chances that conjunctions of different actions do not co-occur. If we compare all 8 experiments with parameters $n = 1$ to $m = 5$ against the equivalent experiments with parameters $n = 4$, we see that half

of those cases show an increase in error (larger values of T_s and corresponding $max.$) while the rest show a decrease (smaller values of T_s and corresponding $max.$). If we limit our comparisons to those cases where only a non-uniform distribution was used, then we do observe the expected reduction in error in all domains except that of the Bridge Inspection domain. Note that this domain differs substantially from the other domains in that it has many more operators, 5 of those operators are overlap operators and, more significantly, the ratio between number of concurrent actions per transaction versus the number of operators is much lower. Experiments using the Bridge Inspection domain wherein the ratio just described is approximately the same for all domains was not done due to the very large processing times required by the algorithm.[15] Until we do this, no conclusion can be drawn.

The Woodwork domain is the most demanding among all domains, as processing time is concerned (larger values of $t/|T|$). We also see that relative to the other domains, save the Bridge Inspection domain whose ratio between the number of concurrent action and the number of oracle operators is substantially lower, the value of its stable points are also higher. The Woodwork domain represents a more complex problem because not only are its transaction length much larger ($\Delta|T_i|$) but the number of linked literals are also larger (roughly proportional to $\Delta|I_i|$). Note that the greatest increase in $t/|T|$ occurs when $n = 4$. We also see a corresponding increase in $max.$, which is expected. To understand why, recall that when we select a literal with which to extend the conjunction, we select the first co-occurring one. As was explained we may inadvertently select one that generates a conjunction that does not have unique coverage. In the case of $n = 4$, the probability of generating such a conjunction increases.[16] Because the coverage is not ensured by these non-consistent conjunction, the greedy cover algorithm repeats the search and therefore ends up generating a larger number of maximals.

The results show that the ratio between the oracle's operators versus the number

[15] We would need to generate transactions in which an average of about 37 actions are executed simultaneously.

[16] The greater the number of concurrently executing actions in regard to the total number of operators, the greater the chances that two or more actions co-occur. In these conditions it is easier to find non-consistent maximals

of concurrently executing actions may have a significant impact on the solution. We can observe that the Elevator, Woodwork and Mars Rover domains, which have approximately the same ratio, in general generate more candidates (especially in the case for $n = 4$) than those required. The Bridge Inspection domain however seems to always generate the correct number of consistent conjunctions.[17] This re-enforces the idea that with an increase in the concurrent execution of actions, the ML algorithm also has a higher probability of finding non-consistent maximals.

In order to test the robustness of the original (no noise handling) ML algorithm, initial tests (all domains for the single case of a uniform distribution and $n = 4$) were executed with data sets that contained 10% errors of commission.[18] The results where not acceptable and therefore discontinued. In all cases we were able to correctly identify the oracle's operator definitions, but the number of incorrect operators definitions, grows steadily with the increase in the number of transactions (no stable point exists). More specifically, each new combination of error induced conjunctions not previously encountered, is added as a new potential operator.

4.7 Noisy Data Experiments

In order to test the robustness of the algorithm all the experiments described above in Section 4.6 were repeated under three noise settings: 10% commission errors, 10% errors of omission, and a combination of 5% commission errors and 5% omission errors. A few attempts were made to experimentally determine a better set of values for the parameters α, β and γ although we usually used the values 0.1, 2 and 0.02 respectively. Results for the Woodwork domain are not presented because of the excessive processing time that was required to execute the experiments. As a result a total of 36 experiments[19] were selected for comparison.

[17] Note that this is not the same number as the oracle operators due to the existence of overlap actions.

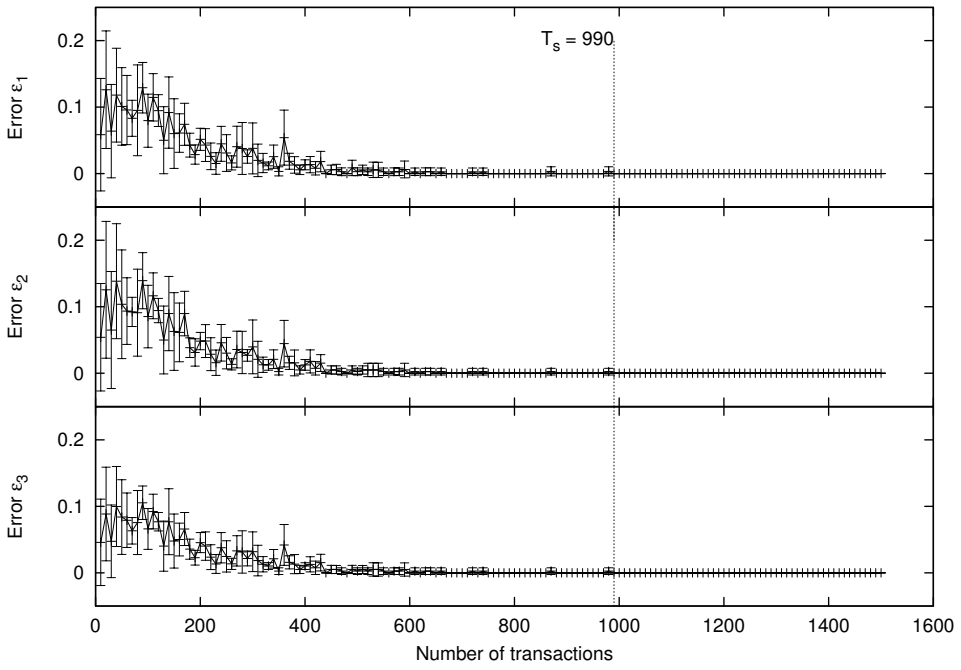
[18] Note that this noise does not include literals used by the oracle definitions themselves. This skews the results in our favour.

[19] 4 non-noise related parameters $\times$ 3 noise related settings $\times$ 3 domains

Elevator										
Parameters					Commission (10%)					
D	Conc.	α	β	γ	*ops.*	T	T_s	$\epsilon_{1(\%)}$	$\epsilon_{2(\%)}$	$\epsilon_{3(\%)}$
uniform	n=4	0	3	0	6	500	40	0.0	0.0	0.0
	n=1 to m=5	0	3	0	6	500	20	0.0	0.0	0.0
non-uniform	n=4	0	3	0	6	1000	70	0.04	0.03	0.0
	n=1 to m=5	0	3	0	6	1000	60	0.04	0.06	0.0
Parameters					Omission (10%)					
D	Conc.	α	β	γ	*ops.*	T	T_s	$\epsilon_{1(\%)}$	$\epsilon_{2(\%)}$	$\epsilon_{3(\%)}$
uniform	n=4	0.1	2	0.02	6	1000	40	1.92	3.99	0.0
	n=1 to m=5	0.1	2	0.02	6	1000	20	3.54	5.52	0.77
non-uniform	n=4	0.1	2	0.02	6	1000	70	1.88	1.61	0.82
	n=1 to m=5	0.1	2	0.02	6	1000	60	1.86	1.81	0.84
Parameters					Both (5%+5%)					
D	Conc.	α	β	γ	*ops.*	T	T_s	$\epsilon_{1(\%)}$	$\epsilon_{2(\%)}$	$\epsilon_{3(\%)}$
uniform	n=4	0.1	2	0.02	6	1000	40	0.41	0.77	0.03
	n=1 to m=5	0.1	2	0.02	6	1000	20	0.77	1.10	0.24
non-uniform	n=4	0.4	2	0.02	6	1000	70	0.57	0.76	0.28
	n=1 to m=5	0.1	2	0.02	6	1000	60	1.19	0.73	0.60

Table 4.2: Results of experiments with Elevator domain noisy data. D is the type of distribution, *Conc.* is the type of concurrency of actions used, α, β and γ are the ML algorithm's error handling parameters, *ops.* the number of operators in the domains, range T_s to T is the total number of transactions within which the error means are calculated and ϵ_1, ϵ_2 and ϵ_3 the measured errors.

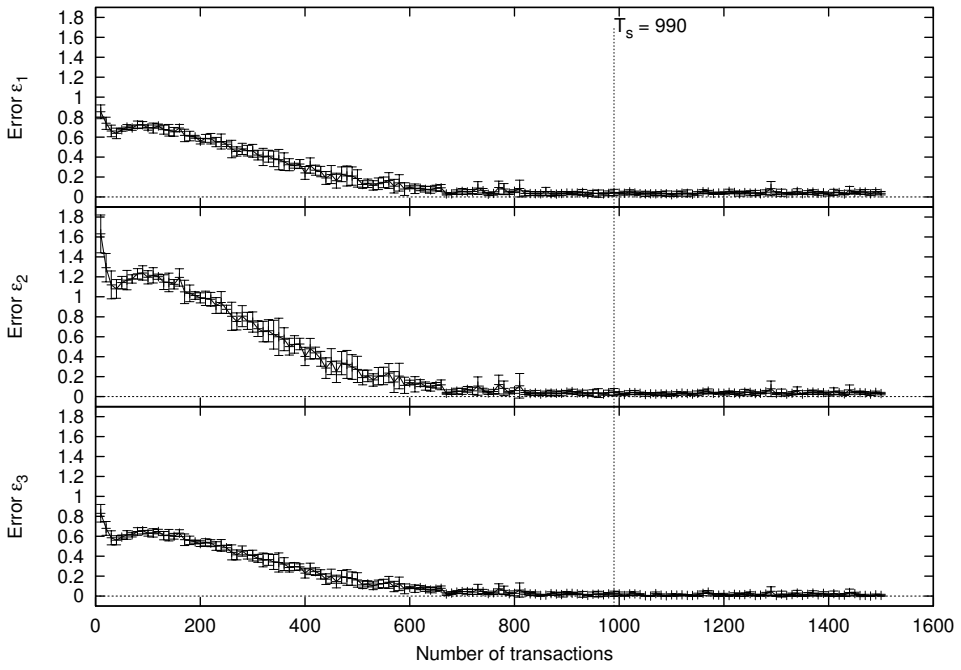
Figure 4.1: Bridge Inspection Noiseless Data Set: Non-uniform, n$^{\underline{o}}$ of operators n=1..5, $T_s = 990$.



Mars Rover										
Parameters					Commission (10%)					
D	Conc.	α	β	γ	*ops.*	T	T_s	$\epsilon_{1(\%)}$	$\epsilon_{2(\%)}$	$\epsilon_{3(\%)}$
uniform	n=4	0	2	0	9	500	50	0.07	0.13	0.0
	n=1 to m=5	0	2	0	9	500	60	0.0	0.0	0.0
non-uniform	n=4	0	2	0	9	1000	340	0.43	0.82	0.0
	n=1 to m=5	0	2	0	9	1000	230	0.11	0.23	0.0
Parameters					Omission (10%)					
D	Conc.	α	β	γ	*ops.*	T	T_s	$\epsilon_{1(\%)}$	$\epsilon_{2(\%)}$	$\epsilon_{3(\%)}$
uniform	n=4	0.1	2	0.02	9	1000	50	9.03	10.03	0.56
	n=1 to m=5	0.1	2	0.02	9	1000	60	9.78	9.75	0.09
non-uniform	n=4	0.1	2	0.02	9	1000	340	10.22	11.85	5.51
	n=1 to m=5	0.1	2	0.02	9	1000	230	2.22	1.29	1.23
Parameters					Both (5%+5%)					
D	Conc.	α	β	γ	*ops.*	T	T_s	$\epsilon_{1(\%)}$	$\epsilon_{2(\%)}$	$\epsilon_{3(\%)}$
uniform	n=4	0.1	2	0.02	9	1000	50	1.44	1.94	0.46
	n=1 to m=5	0.1	2	0.02	9	1000	60	0.75	0.85	0.21
non-uniform	n=4	0.1	2	0.02	9	1000	340	7.06	6.34	5.07
	n=1 to m=5	0.1	2	0.02	9	1000	230	1.34	0.79	0.74

Table 4.3: Results of experiments with Mars Rover domain noisy data. D is the type of distribution, *Conc.* is the type of concurrency of actions used, α, β and γ are the ML algorithm's error handling parameters, *ops.* the number of operators in the domains, range T_s to T is the total number of transactions within which the error means are calculated and ϵ_1, ϵ_2 and ϵ_3 the measured errors.

Figure 4.2: Bridge Inspection Noisy Data Set: 5% omission and 5% commission noise, Non-uniform, n^{o} of operators n=1..5, $T_s = 990$.



Bridge Inspection										
Parameters					Commission (10%)					
D	Conc.	α	β	γ	*ops.*	T	T_s	$\epsilon_{1(\%)}$	$\epsilon_{2(\%)}$	$\epsilon_{3(\%)}$
uniform	n=4	0	2	0	80	400	220	0.01	0.01	0.01
	n=1 to m=5	0	2	0	80	500	290	0.0	0.0	0.0
non-uniform	n=4	0	2	0	80	1000	650	0.28	0.27	0.26
	n=1 to m=5	0	2	0	80	1500	990	0.13	0.14	0.13
Parameters					Omission (10%)					
D	Conc.	α	β	γ	*ops.*	T	T_s	$\epsilon_{1(\%)}$	$\epsilon_{2(\%)}$	$\epsilon_{3(\%)}$
uniform	n=4	0.1	2	0.02	80	1000	220	4.28	4.55	2.23
	n=1 to m=5	0.1	2	0.02	80	1000	290	6.86	6.79	4.06
non-uniform	n=4	0.4	2	0.0	80	1000	650	7.08	7.23	4.24
	n=1 to m=5	0.4	2	0.0	80	1500	990	7.95	8.08	4.69
Parameters					Both (5%+5%)					
D	Conc.	α	β	γ	*ops.*	T	T_s	$\epsilon_{1(\%)}$	$\epsilon_{2(\%)}$	$\epsilon_{3(\%)}$
uniform	n=4	0.1	2	0.01	80	1000	220	2.22	2.71	0.84
	n=1 to m=5	0.4	2	0.0	80	1000	290	3.90	4.39	1.58
non-uniform	n=4	0.1	2	0.01	80	1000	650	2.93	3.21	1.90
	n=1 to m=5	0.4	2	0.0	80	1500	990	3.65	4.06	1.53

Table 4.4: Results of experiments with Bridge Inspection domain noisy data. D is the type of distribution, *Conc.* is the type of concurrency of actions used, α, β and γ are the ML algorithm's error handling parameters, *ops.* the number of operators in the domains, range T_s to T is the total number of transactions within which the error means are calculated and ϵ_1, ϵ_2 and ϵ_3 the measured errors.

These results were first graphically compared to the cases with noiseless data. The Bridge Inspection domain, for the case of a non-uniform distribution and a varying number of concurrent actions using noiseless data, is shown in Figure 4.1. By comparing it to the noisy data case (see Figure 4.2), we can clearly observe an increase in the errors.

We observed that when noise is present, no fixed point T_s exists for all of the errors we calculated (see Section 4.5 for a description of these). However in the cases of errors of commission only, both the Mars Rover and Elevator domains maintain their T_s in respect to the mean number of oracle operators not correctly identified (ϵ_3). The Bridge Inspection domain, in the same conditions, does not retain the same fixed point for ϵ_3 but nevertheless does have one that occurs much later. Because in all other cases no T_s exists that we may use for comparison, we instead use average error rates, as measured between the original noiseless T_s and the maximum T. The average errors for the Elevator, Mars Rover and Bridge Inspection domains are

shown in the Tables 4.2, 4.3 and 4.4 respectively.

From these results we can see that in general the algorithm is robust to noisy data. All of the average error rates that we have calculated are less than the injected error itself. Of the 36 experiments only 3 show error rates above that of the 10% injected noise. All instances occur in the Mars Rover domain. The error rates are $\epsilon_2 = 10.03\%$ (uniform, $n = 4$), $\epsilon_1 = 10.22\%$ (non-uniform, $n = 4$) and $\epsilon_2 = 11.85\%$ (non-uniform, $n = 4$) respectively.

We can also see that errors of omission have the greatest negative impact. All errors of omission are larger than those of commission and less than the combination of both types of errors. We also attempted to find out if any additional correlation between the actions distribution, number of concurrent action or even the number of operators and the error rates exist. However none was found.

4.8 Concluding Remarks

The experimental results show that combining co-occurring relations can be used to successfully identify simple planning operators when the data has no noise. When the data has errors of commission we are still able to learn all of the oracle's simplified operators. However, errors ϵ_1 and ϵ_2 indicate that a number conjunctions were identified as planning operators when they should have not been (threshold frequency β was too low). Note however that this is only true when the planning domain does not have overlap operators. Further investigation showed that this occurs because the algorithm cannot determine the operator's components and therefore cannot combine them correctly. It does however extend several of the components into overlap operators and the greedy covering algorithm therefore fails to generate the correct set of overlap components.

The errors of omission do not allow us to identify all of the oracle's simplified operators. Even so most of the errors we observe are well below the 5% mark. The only exception being the case of the Mars Rover domain with an error of $\epsilon_3 = 5.51\%$. We also observed that the errors of omission have the greatest impact on the final results. We can observe that all omission errors for the case of 10% noise are larger

than the combination of 5% omission and 5% commission errors.

Overall the algorithm is robust to noise showing a maximum error of $\epsilon_2 = 11.85\%$ for the Mars Rover domain in the case of 10% omission errors. We believe that such an error is because the heuristic search for a conjunction, only extends the seed with a single[20] co-occurring literals instead of considering all possible extensions. In the latter case one could then select the resulting conjunction with the highest support thereby avoiding the creation of non-consistent candidates.

[20]In an attempt to reduce the exponential cost of the search.

Chapter 5

Conclusions

5.1 Introduction

The goal of this thesis is to provide a means of learning the constituent parts of ad-hoc processes by non-obstructive observation of their execution. To this end we propose the identification of planning operators that can then be used to automatically compose processes using AI automated planning techniques. Specifically this work allows the learning of planning rules that can be used to generate Strong Cyclic Plans via a planning system based on formal Model checking techniques. We have successfully developed a Machine Learning algorithm that can learn planning operators using data generated by an experimentation platform. This platform allowed us to test several planning domains under various conditions including a number of noise settings to simulate sensing and sampling errors.

5.2 Contributions

Although the main focus of the work was the development and testing of the ML algorithm, our first contribution is the formalization and the description of the problem itself. First we point out that the observation of the state transitions allow us only to determine simple operator rules. However several of these rules can successfully describe more complex operators, such as actions with non-deterministic or conditional effects, which can be used by model checking-based planners. In fact

the ML algorithm can learn complex operators with disjunctive pre-conditions as a set of simple operator rules. It is important to note however that the simple rules that are automatically acquired by the ML algorithm can be used by a planning system without any change.

Our second contribution is the parametrization used in the generation of the experimental dataset. It consists of: the uniform and non-uniform distributions of operator instantiation, using a given (fixed or variable) number of concurrently executing actions and injecting several combinations of noise types into the dataset. Noise includes errors of omission or commission only or a combination of both. We have shown experimentally that the errors of omission are the most difficult ones do deal with and have explained why that happens.

5.2.1 Formalizing the Problem

Our third, and arguably most valuable contribution, is expressing the Machine Learning problem as a specific instance of the general Minimal Consistent Subset Cover problem. In this formulation the dataset consists of a list of transactions. Each transaction is a set of qualified literals that represents a single state transition. A subset of these qualified literals expresses a rule that covers (explains) part of a state transition (supported by evidence). We therefore require a minimum of these rules to cover all of the transactions. Consistency is guaranteed by imposing the following constraints on covering subsets of qualified literals: must consist of the largest possible set of co-occurring literals (maximals) only; must have maximum support; must be correctly qualified in order to express a rule; and the rule must express a valid planning operator. The latter condition means that a negative literal of the rule's consequent (operator effect) can only exist if it is also present in the antecedent (operator's pre-condition) and that a positive literal of the rule's consequent can only exist if it is not present in its antecedent.

We have found out that the above formulation allows us to determine operator definitions of various planning domains used in the planning competitions. However a set of operators specifically developed for the case of a workflow process shows that the co-occurrence condition may be too restrictive. More specifically operator

definitions may consist of combinations of two or more consistent conjunctions. The resulting operators are referred to as overlap[1] operators. The problem is then how one should go about combining these maximals without causing the overfitting of the result. To this end we have also determined an additional set of conditions that allows us to solve this problem irrespective of whether or not the domain contains overlap operators. More concretely, in order to determine the definitions of the overlap operators, we must attempt to extend all consistent conjunctions with one or more other consistent conjunctions if it does not represent a valid operator. If at least two such extensions represent correct operators and ensure the same coverage, then such extended conjunctions are recorded as overlap operators.

5.2.2 Heuristic Algorithm

Solving the MCSC problem is NP-complete. As with any other problem that has an exponential cost to solve, we have also turned to the use of heuristics. Greedy cover algorithms were used in order to determine both the minimal set of maximals and to combine those maximals into overlap operators. This is done by attempting to efficiently generate candidate maximals that have the highest possible support values and adhere to the constraints referred to in the previous section. In addition to this the base algorithm is also extended in order to deal with noisy data.

The challenges here consist in determining the order in which the maximals are generated and pruned. For example, overlap operators consist of maximals which are not valid rules. It is therefore necessary to proceed with the pruning of maximals, which do not represent rules, only after the check for overlap actions. Another issue is how to guarantee convergence of the algorithm when noisy data is used. In such a case several literals may not be covered by a rule due to errors of omission or commission. Our fourth and final contribution was therefore the design and implementation of a robust ML algorithm that can solve the maximal consistent subset cover of noisy datasets.

[1]Overlap in the sense that the various maximals that are combined do not co-occur but share support of the transactions.

5.2.3 Assessment of the ML Algorithm

Several experiments were executed based on four planning domains. Three of these domains are obtained from a planning competition, having been slightly altered in order to avoid the use of constants. The fourth domain was designed to specifically test the learning capabilities of non-deterministic and overlap operators. The datasets were generated according to several parameters, the most significant among them being the number of simultaneously executing actions and the type and amount of noise injected into the datasets.

Experimental evaluation has shown that irrespective of the type of operators and number of simultaneously executing actions, if the dataset is noiseless then the number of operators and their definitions can be correctly determined. If however the dataset contains only errors of commission, then an additional number of operators may be inadvertently included in the result. However all of the oracles operators will always be correctly determined. In the case errors of omission (independently of whether or not it is combined with errors of commission), it is not possible to ensure that all of the oracle's operator definitions can be identified. However in such cases, the ML algorithm has shown to be robust to noise, generating results with low error rates.

5.3 Future Research

Although we strove to avoid the exponential cost of the MCSC problem, the use of relations means that the algorithm still has a worst case exponential cost. The experiments with the Woodwork domain show that this algorithm is intractable for transactions consisting of more than 30 literals (compare values of $\Delta|T_i|$ and $\Delta|I_i|$ in Table 4.1). To understand why, assume we express all relations and variables of a conjunction as a graph.[2]. This means that each transaction and any operator may be expressed as a graph If we use the non-optimal greedy set covering strategy ([Vaz01]) described, then marking the coverage of an operator is equivalent to the

[2]Various representation are possible.

sub-graph isomorphism problem, which is NP-complete ([Sol10]). Note also that because a single transaction may represent multiple instances of the same operator, we must enumerate all such sub-graph isomorphisms which is at least NP-complete ([LVG09]).

Besides marking the coverage of an operator, we must first determine what constitutes such an operator. This is akin to the maximum common sub-graph isomorphism, which is also NP-complete ([CGS03]). This is a difficult problem whose approximations are still NP-hard ([SAKZ+05]). Moreover it must be extended to deal with a set of graphs (transactions). This problem is also referred to as the multiple largest common sub-graph (MLCS in [BSJL92]).

Future work will therefore focus on the use of efficient algorithms for determining the maximum common (co-occurring) sub-graphs. Currently such algorithms are based either on identifying the maximal clique ([PX94]) or the use of intelligent enumerative backtracking algorithms ([KH04]). Various maximal clique based algorithms exist including those that use a SAT solver ([LQ10]), dynamic programming ([Ö02]) and branch and bound algorithms ([TK07]). We are also considering the reorganization of the ML algorithm so that we may take advantage of parallel processing (identifying each operator may be done independently; once an operator has been determined its complete coverage may be tested independently) and caching (can we use previous tests of maximum sub-graph isomorphism to speed up the search). Finally we may consider the use of approximation algorithms that may be used in cases of noiseless data ([ZYJ10]) or the decomposition of the problem into smaller manageable parts (for example identify operator's pre-conditions and effects separately).

References

[AC08] Eyal Amir and Allen Chang. Learning partially observable deterministic action models. volume 33, pages 349–402, USA, November 2008. AI Access Foundation.

[ACBC+04] Mitchell Ai-Chang, John Bresina, Len Charest, Adam Chase, Jennifer Cheng jung Hsu, Ari Jonsson, Bob Kanefsky, Paul Morris, Kanna Rajan, Jeffrey Yglesias, Brian G. Chafin, William C. Dias, and Pierre F. Maldague. Mapgen: Mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems*, 19(1):8–12, 2004.

[AGF06] Fabiano S. R. Alves, Kairon F. Guimaraes, and Marcia A. Fernandes. Modeling workflow systems with genetic planner and scheduler. In *ICTAI '06: Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, pages 381–388, Washington, DC, USA, 2006. IEEE Computer Society.

[AGL98] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In Hans-Jrg Schek, Flix Saltor, Isidro Ramos, and Gustavo Alonso, editors, *Advances in Database Technology. Proceedings of the 6th International Conference on Extending Database Technology*, pages 469–483, Valencia, 1998.

[AKYG07] N. F. Ayan, U. Kuter, F. Yaman, and R. Goldman. HOTRiDE:

Hierarchical ordered task replanning in dynamic environments. In *Proceedings of the ICAPS-07 Workshop on Planning and Plan Execution for Real-World Systems – Principles and Practices for Planning in Execution*, pages 31–36, Providence, Rhode Island, USA, September 2007.

[Ami04] Eyal Amir. Learning partially observable action models. In 4^{th} *international workshop on cognitive robotics (CogRob'04)*, pages 46–52, 2004.

[BCP+01] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a model based planner. In *Proceeding of ICAI-2001 workshop on Planning under Uncertainty and Incomplete Information*, pages 93–97, Seattle, WA, 2001.

[BCRT06] Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Strong planning under partial observability. *Artif. Intell.*, 170(4):337–384, 2006.

[BD99] P.M. Berry and B. Drabble. SWIM: An AI-based system for workflow enabled reactive control. In *Proceedings of the IJCAI Workshop on Workflow and Process Management held as part of IJCAI-99*. IJCAI, Stockholm, Sweden, August 1999.

[BD00] P.M. BERRY and B. Drabble. SWIM: An AI-based system for organizational management. In *In proceedings of the 2nd NASA Intl. workshop on Planning and Scheduling for Space*. NASA, March 2000.

[Ben95a] Scott Benson. Action model learning and action execution in a reactive agent. In *Proceedings of the 1995 International Joint Conference on AI*, August 1995.

[Ben95b] Scott Benson. Inductive learning of reactive action models. In *Proceedings of the Twelfth International Conference on Ma-*

chine Learning (ICML), pages 47–54, Tahoe City, California, USA, July 9-12 1995. Morgan Kaufmann.

[Ber00] Alfs. T. Berztiss. Knowledge and workflow systems. In A. Min Tjoa, R.R. Wagner, and A. Al-Zobaidie, editors, *Proceedings of the 11th International Workshop on Database and Expert Systems Applications (DEXA '00)*, pages 1102–1108, London, UK, 2000. IEEE Computer Society.

[BF97] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artif. Intell.*, 90(1-2):281–300, 1997.

[BG01] B. Bonet and H. Geffner. GPT: a tool for planning with uncertainty and partial information. In A. Cimatti, H. Geffner, E. Giunchiglia, and J. Rintanen, editors, *Proc. IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*, pages 82–87, Seattle, WA, 2001.

[BJ03] B. Srivastava and J. Koehler. Web Service Composition - Current Solutions and Open Problems. In *ICAPS 2003 Workshop on Planning for Web Services*, pages 28–35, 2003.

[BK98] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22(1-2):5–27, 1998.

[BKS08] Daniel Bryce, Subbarao Kambhampati, and David E. Smith. Sequential monte carlo in reachability heuristics for probabilistic planning. *Artif. Intell.*, 172(6-7):685–715, 2008.

[BP84] Giampio Bracchi and Barbara Pernici. The design requirements of office systems. *ACM Transaction on Office Information Systems*, 2(2):151–170, 1984.

[BPNG09] Michele Berlingerio, Fabio Pinelli, Mirco Nanni, and Fosca Giannotti. Temporal mining for interactive workflow data anal-

ysis. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 109–118, New York, NY, USA, 2009. ACM.

[BR96] Marie-Claude Boudreau and Daniel Robey. Coping with contradictions in business process re-engineering. *Information Technology & People*, 9(4):40–57, 1996.

[BSJL92] Denis M. Bayada, Richard W. Simpson, A. Peter Johnson, and Claude Laurenco. An algorithm for the multiple common subgraph problem. *J. Chem. Inf. Comput. Sci.*, 32(6):680–685, 1992.

[Car97] Steinar Carlsen. *Conceptual Modeling and Composition of Flexible Workflow Models*. PhD thesis, Department of Computer and Information Science, Faculty of Physics, Informatics and Mathematics, Norwegian University of Science and Technology, Trondheim, Norway, December 1997.

[CC06] Gabriella Cortellessa and Amedeo Cesta. Evaluating mixed-initiative systems: An experimental approach. In Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey, editors, *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pages 172–181, Cumbria, UK, June 6-10 2006. AAAI.

[CGS03] D. Conte, C. Guidobaldi, and C. Sansone. A comparison of three maximum common subgraph algorithms on a large database of labeled graphs. In *Proceedings of the 4th IAPR international conference on Graph based representations in pattern recognition*, GbRPR'03, pages 130–141, Berlin, Heidelberg, 2003. Springer-Verlag.

[CPRT03] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak,

strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2):35–84, 2003.

[CRT98] Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Strong planning in non-deterministic domains via model checking. In *AIPS*, pages 36–43, 1998.

[DBS06] Subbarao Kambhampati Daniel Bryce and David E. Smith. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006.

[dMWvdA07] A. de Medeiros, A. Weijters, and W. van der Aalst. Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery*, 14:245–304, 2007.

[ECA+03] T. Estlin, R. Castano, B. Anderson, D. Gaines, and M. Judd F. Fisher. Learning and planning for mars rover science. In *International Joint Conference on Artificial Intelligence Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modeling, planning, learning, and communicating.*, Acapulco, August 2003. IJCAI.

[ENS95] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell.*, 76(1-2):75–88, 1995.

[ESBPDWJHDN04] Evren Sirin; Bijan Parsia; Dan Wu; James Hendler and Dana Nau. HTN planning for Web Service Composition using SHOP2. In *Web Semantics: Science, Services and Agents on the World Wide Web*, volume 1, pages 377–396. International Semantic Web Conference 2003, October 2004.

[Fer06] D.R. Ferreira, H.M.; Ferreira. An integrated life cycle for workflow management based on learning and planning. *International Journal of Cooperative Information Systems.*, 15(4):485–505, December 2006.

[FGLS06] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey, editors, *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pages 212–221, Cumbria, UK, June 6-10 2006. AAAI.

[Fis02] Layna Fischer, editor. *The Workflow Handbook 2002*, chapter Introduction to Workflow: A Vendor-independent tutorial, pages 19–38. Future Strategies Inc, 2002.

[FL03] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

[GABG05] Walid Gaaloul, Sadek Alaoui, Karim Baina, and Claude Godart. Mining workflow patterns through event-data analysis. In *SAINT-W '05: Proceedings of the 2005 Symposium on Applications and the Internet Workshops*, pages 226–229, Washington, DC, USA, 2005. IEEE Computer Society.

[GBG08] Walid Gaaloul, Karim Bana, and Claude Godart. Log-based mining techniques applied to web service composition reengineering. *Service Oriented Computing and Applications*, 2:93–110, 2008.

[GDB+04] Yolanda Gil, Ewa Deelman, Jim Blythe, Carl Kesselman, and Hongsuda Tangmunarunkit. Artificial intelligence and grids: Workflow planning and beyond. *IEEE Intelligent Systems*, 19(1):26–33, 2004.

[GEC+07] Byron J. Gao, Martin Ester, Jin-Yi Cai, Oliver Schulte, and Hui Xiong. The minimum consistent subset cover problem and its applications in data mining. In *KDD '07: Proceedings of*

the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 310–319, New York, NY, USA, 2007. ACM.

[GMB00] Ramón García-Martínez and Daniel Borrajo. An integrated approach of learning, planning, and execution. *J. Intell. Robotics Syst.*, 29(1):47–78, 2000.

[GMB+07] Stijn Goedertier, David Martens, Bart Baesens, Raf Haesen, and Jan Vanthienen. A new approach for discovering business process models from event logs. Open Access publications from Katholieke Universiteit Leuven urn:hdl:123456789/120460, Katholieke Universiteit Leuven, 2007.

[GMVB09] Stijn Goedertier, David Martens, Jan Vanthienen, and Bart Baesens. Robust process discovery with artificial negative events. *Journal of Machine Learning Research*, 10:1305–1340, 2009.

[GT97] Dimitrios Georgakopoulos1 and Aphrodite Tsalgatidou. Technology and Tools for Comprehensive Business Process Lifecycle Management. In Doa Asuman ; Kalinichenko Leonid ; zsu M. Tamer ; Sheth Amit, editor, *NATO ASI series. Series F : computer and system sciences*, volume 164, pages 356–395, Istanbul, Turkey, 12-21 August 1997. Workshop management systems and interoperability: NATO advanced study institute on workflow management systems, Plenum, New York, NY, ALLEMAGNE (19) (Revue); Springer, Berlin, ALLEMAGNE (1998) (Monograph).

[HK04] Joachim Herbst and Dimitris Karagiannis. Workflow mining with inwolve. *Comput. Ind.*, 53(3):245–264, 2004.

[Hog90] Christopher John Hogger. *Essentials of logic programming*. Oxford University Press, Inc., New York, NY, USA, 1990.

[HS07] David Hildum and Stephen F. Smith. Constructing conflict-free schedules in space and time. In Mark Boddy, Maria Fox, and Sylvie Thi'ebaux, editors, *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 184–191, Providence, Rhode Island, USA, September 2007. AAAI Press.

[IG06] Jon Espen Ingvaldsen and Jon Atle Gulla. Model-based business process mining. *Information Systems Management*, 23(1):19–31, 2006.

[JB98] Peter Jonsson and Christer Bäckström. Tractable plan existence does not imply tractable plan generation. *Annals of Mathematics and Artificial Intelligence*, 22(3-4):281–296, 1998.

[Jør04] Håvard D. Jørgensen. *Interactive Process Models*. PhD thesis, Department of Computer and Information Science, Faculty of Information Technology, Norwegian University of Science and Technology, Trondheim, Norway, January 2004.

[JSM+99] Peter Jarvis, Jussi Stader, Ann Macintosh, Jonathan Moore, and Paul Chung. What right do you have to do that? Infusing Adaptive Workflow Technology with Knowledge about the Organisational and Authority Context of a Task. In *Proceedings of the First International Conference on Enterprise Information Systems (ICEIS-99)*, pages 240–247, Setubal, Portugal, March 1999.

[JV07] Rune M. Jensen and Manuela M. Veloso. Learning non-deterministic multi-agent planning domains. Workshop on Artificial Intelligence Planning and Learning. In conjunction with the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS-07), `http://www.cs.umd.edu/~ukuter/icaps07aipl/`

`proceedings/paper4.pdf`, 22 September 2007. Online; accesed 20 September 2010.

[Kam07] Subbarao Kambhampati. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In Robert C. Holte and Adele Howe, editors, *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 1601–1604, Vancouver, British Columbia, Canada, 2007. AAAI Press.

[KH04] Evgeny B. Krissinel and Kim Henrick. Common subgraph isomorphism detection by backtracking search. *Softw. Pract. Exper.*, 34:591–607, May 2004.

[KL06] Tolga Könik and John E. Laird. Learning goal hierarchies from structured observations and expert annotations. *Mach. Learn.*, 64(1-3):263–287, 2006.

[KN87] Deepak Kapur and Paliath Narendran. Matching, unification and complexity. *SIGSAM Bull.*, 21:6–9, November 1987.

[KS99] Henry A. Kautz and Bart Selman. Unifying sat-based and graph-based planning. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 318–325, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[KS00] Jana Koehler and Kilian Schuster. Elevator control as a planning problem. In S. Chien, S. Kambhampati, and C. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pages 331–338, Menlo Park, Calif., 2000. AAAI Press.

[LADM07] Jos J. Prez-Alczar Luciano A. Digiampietri and Claudia B. Medeiros. AI Planning in Web Services Composition: a review

of current approaches and a new solution. In *Proceedings of the XXVII Conference Brazilian Computer Science (CSBC2007)*, pages 983–992, Rio de Janeiro, Brazil, July 3-7 2007. VI ENIA.

[LAP06] Alexander Lazovik, Marco Aiello, and Mike Papazoglou. Planning and monitoring the execution of web service requests. In *International Journal on Digital Libraries*, volume 6, pages 235–246. Springer Berlin, June 2006.

[LD06] Geoffrey Levine and Gerald DeJong. Explanation-based acquisition of planning operators. In Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey, editors, *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pages 152–161, Cumbria, UK, June 6-10 2006. AAAI.

[LGM98] Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.

[LPT02] Ugo Dal Lago, Marco Pistore, and Paolo Traverso. Planning with a language for extended goals. In *Eighteenth national conference on Artificial intelligence*, pages 447–454, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[LQ10] Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *AAAI*, 2010.

[LVG09] V. Lipets, N. Vanetik, and E. Gudes. Subsea: an efficient heuristic algorithm for subgraph isomorphism. *Data Mining and Knowledge Discovery*, 19:320–350, 2009.

[MB99a] K.L. Myers and P.M. Berry. At the Boundary of Workflow

and AI. In *Proceedings of the AAAI-99 Workshop on Agent-Based Systems in The Business Context held as part of AAAI-99, Technical Report WS-99-02*, pages 41–49. AAAI, Orlando, Florida, July 1999.

[MB99b] K.L. Myers and P.M. Berry. Workflow Management Systems: An AI Perspective. Technical report, AIC, SRI International, Jan. 1999.

[MF90] Stephen Muggleton and Cao Feng. Efficient induction of logic programs. In *Proceedings of the First International Workshop on Algorithmic Learning Theory—ALT-90*, pages 368–381, 1990.

[MLR06] Zhen Liu Marc Lelarge and Anton V. Riabov. *Autonomic and Trusted Computing*, volume Volume 4158/2006 of *Lecture Notes in Computer Science*, chapter Automatic Composition of Secure Workflows, pages 322–331. Springer Berlin / Heidelberg, October 2006.

[MOJ+07] S. Mahadevan, S. Osentoski, J. Johns, K. Ferfuson, and C. Wang. Learning to plan using harmonic analysis of diffusion models. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS-07)*, pages 224–232, Providence, Rhode Island, USA, September 2007.

[Nau07] Dana S. Nau. Current trends in automated planning. *AI Magazine*, 28(4):43, 2007.

[NCdW95] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. The subsumption theorem in inductive logic programming: Facts and fallacies. In *Advances in Inductive Logic Programming. IOS*, pages 265–276. Press, 1995.

[NK01] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In Bernhard Nebel, editor, *IJCAI*, pages 459–466. Morgan Kaufmann, 2001.

[NLFL07] M. A. H. Newton, J. Levine, M. Fox, and D. Long. Learning macro-actions for arbitrary planners and domains. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 07)*, pages 256–265, Providence, Rhode Island, USA, September 2007.

[Ö02] Patric R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Appl. Math.*, 120:197–207, August 2002.

[OC96a] Tim Oates and Paul R. Cohen. Learning planning operators with conditional and probabilistic effects. In *Planning with Incomplete Information for Robot Problems: Papers from the 1996 AAAI Spring Symposium*, pages 86–94. AAAI Press, Menlo Park, California, 1996.

[OC96b] Tim Oates and Paul R. Cohen. Searching for planning operators with context-dependent and probabilistic effects. In Howard Shrobe and Ted Senator, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, Vol. 2*, pages 865–868, Menlo Park, California, 1996. AAAI Press.

[PS92] Mark A. Peot and David E. Smith. Conditional nonlinear planning. In *Proceedings of the first international conference on Artificial intelligence planning systems*, pages 189–197, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[PW92] J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In Bernhard Nebel,

Charles Rich, and William Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 103–114, San Mateo, California, 1992. Morgan Kaufmann.

[PX94] Panos M. Pardalos and Jue Xue. The maximum clique problem. *Journal of Global Optimization*, 4:301–328, 1994.

[PZK04] Hanna Pasula, Luke S. Zettlemoyer, and Leslie Pack Kaelbling. Learning probabilistic relational planning rules. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *ICAPS*, pages 73–82. AAAI, 2004.

[Qui87] J.R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[Rei92] Wolfgang Reisig. *A Primer in Petri Net Design.* Springer Compass International, 1992.

[RH01] Jussi Rintanen and Jörg Hoffmann. An overview of recent algorithms for AI planning. *KI*, 15(2):5–11, 2001.

[RIAS97] Luc De Raedt, Peter Idestam-Almquist, and Gunther Sablon. Theta-subsumption for structural matching. In *Proceedings of the 9th European Conference on Machine Learning*, pages 73–84, London, UK, 1997. Springer-Verlag.

[Rin04] J. Rintanen. Conditional planning in the discrete belief space. Technical Report 205, Albert-Ludwigs-Universitt Freiburg, Institut fr Informatik, 2004.

[RKM06] J. Rao, P. Kungas, and M. Matskin. Composition of Semantic Web Services using Linear Logic Theorem Proving. *Information Systems Journal - Special Issue on the Semantic Web and Web Services*, 31(4-5):229–296, June-July 2006.

[RMBCO07] María Dolores R-Moreno, Daniel Borrajo, Amedeo Cesta, and Angelo Oddi. Integrating planning and scheduling in workflow domains. *Expert Syst. Appl.*, 33(2):389–406, 2007.

[RMBM00a] M.D. R-Moreno, D. Borrajo, and D. Meziat. Process modeling and AI planning Techniques: A new approach. In *Second International Workshop on Information Integration and Web-based Applications and Services*, pages 231–242, Yogyakarta (Indonesia), 2000. IIWAS2000.

[RMBM00b] M.D. R-Moreno, D. Borrajo, and D. Meziat. Transforming Business Processes modeling into Planning Problems. In *PLANET News Letter*, pages 10–12, 2000.

[RMBM01] M.D. R-Moreno, D. Borrajo, and D. Meziat. Planning based generation of process models. In Lee McCluskey and Alfredo Milani, editors, *New Methods of Electronic Mobile and Collaborative Work*, pages 22–34, Toledo (Spain), September 2001. ECP01 Workshop on Planning and Scheduling Technologies.

[RMBO+05] M.D. R-Moreno, D. Borrajo, A. Oddi, A. Cesta, and D. Meziat. Planning and scheduling for workflow domains. In *Planning, scheduling and constraint satisfaction: from theory to practice*, Frontiers in Artificial Intelligence and Applications, pages 169–170. IOS Press., 2005.

[RMK01] M.D. R-Moreno and P. Kearney. Let's see what happen if we integrate AI Planning with Workflow Management System. In *21st SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, pages 299–312, Cambridge (U.K.), 2001.

[RMK02] M.D. R-Moreno and P. Kearney. Integrating AI Planning with Workflow Management System. *International Journal of Knowledge-Based Systems, Elsevier*, 15:285–291, 2002.

[RMKM00] M.D R-Moreno, P. Kearney, and D. Meziat. A Case Study: Using Workflow and AI Planners. In *The 19th Workshop of the UK Planning and Scheduling (PLANSIG2000)*, Milton Keynes (U.K.), 2000.

[RMOB+04] M.D. R-Moreno, A. Oddi, D. Borrajo, A. Cesta, and D. Meziat. Planning and scheduling for workflow domains. In *Planning and Scheduling: Bridging Theory to Practice*, pages 75–84, Valencia (Spain), August 2004. 16th European Conference in Artificial Intelligence (ECAI 2004) workshop on Planning and Scheduling.

[RSDC99] J. Ritchie, J. Simmons, R. Dewar, and I. Carpenter. A methodology for eliciting expert knowledge in virtual engineeringenvironments. In *Management of Engineering and Technology*, volume vol.1, page 202, Portland, OR, USA, July 1999. PICMET '99. Portland International Conference on Technology and Innovation Management.

[RZV+09] A. Rozinat, S. Zickler, M. Veloso, WMP van der Aalst, and C. McMillen. Analyzing Multi-agent Activity Logs Using Process Mining Techniques. In *Distributed Autonomous Robotic System 8*, page 251. Springer, 2009.

[SA06] Dafna Shahaf and Eyal Amir. Learning partially observable action schemas. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, pages 913–919, Boston, Massachusetts, USA, July 16-20 2006. AAAI Press.

[SAKZ+05] W. Henry Suters, Faisal N. Abu-Khzam, Yun Zhang, Christopher T. Symons, Nagiza F. Samatova, and Michael A. Langston. A new approach and faster exact methods for the maximum common subgraph problem. In Lusheng Wang, editor, *Computing and Combinatorics*, volume 3595 of *Lecture*

Notes in Computer Science, pages 717–727. Springer Berlin / Heidelberg, 2005.

[SCA06] Dafna Shahaf, Allen Chang, and Eyal Amir. Learning partially observable action models: efficient algorithms. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, pages 920–926. AAAI Press, 2006.

[Sch04] Guido Schimm. Mining exact models of concurrent workflows. *Comput. Ind.*, 53:265–281, April 2004.

[SHH95] Bill Schwartz, Betty W. Hwang, and C. Jinshong Hwang. A workplan for business process reengineering and a challenge for information science and technology. In *Proceedings of the 1995 ACM 23rd annual conference on Computer science*, pages 178 – 194, Nashville, Tennessee, United States, 1995. ACM Annual Computer Science Conference.

[SM10] Jose Santos and Stephen Muggleton. Subsumer: A prolog theta-subsumption engine. In Manuel Hermenegildo and Torsten Schaub, editors, *Technical Communications of the 26th International Conference on Logic Programming*, volume 7 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 172–181, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Sol10] Christine Solnon. Alldifferent-based filtering for subgraph isomorphism. *Artif. Intell.*, 174:850–864, August 2010.

[Ste01] Dick Stenmark. Leveraging tacit organizational knowledge. *J. Manage. Inf. Syst.*, 17(3):9–24, 2001.

[SW98] David E. Smith and Daniel S. Weld. Conformant graphplan. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applica-*

tions of artificial intelligence, pages 889–896, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[TCH99] Kang Soo Tae, Diane J. Cook, and Lawrence B. Holder. Experimentation-driven knowledge acquisition for planning. *Computational Intelligence*, 15:252–279, 1999.

[TK07] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *J. of Global Optimization*, 37:95–111, January 2007.

[Tra04] Dana Nau; Malik Ghallab; Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[vdA10] Wil M. P. van der Aalst. Process discovery: capturing the invisible. *Comp. Intell. Mag.*, 5:28–41, February 2010.

[vdARV+09] W. van der Aalst, V. Rubin, H. Verbeek, B. van Dongen, E. Kindler, and C. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 9(1):87–111, January 2009.

[vdAvDH+03] Wil M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

[vdKdW04] Roman P.J. van der Krogt and Mathijs M. de Weerdt. The two faces of plan repair. In *Proceedings of the BNAIC (BNAIC-04)*, pages 147–154, 2004.

[vdKdWW03] Roman van der Krogt, Mathijs de Weerdt, and Cees Witteveen. A resource based framework for planning and replanning. *Web Intelli. and Agent Sys.*, 1(3-4):173–186, 2003.

[Wan95] Xuemei Wang. Learning by observation and practice: An incremental approach for planning operator acquisition. In *In Proceedings of the 12th International Conference on Machine Learning (ICML)*, pages 549–557, 1995.

[Wan96a] Xuemei Wang. A multistrategy learning system for planning operator acquisition. In *Third International Workshop on Multistrategy Learning*, pages 229–236, Harpers Ferry, West Virginia., 23-25 May 1996.

[Wan96b] Xuemei Wang. Planning while learning operators. In *AIPS*, pages 229–236, 1996.

[WAS98] Daniel S. Weld, Corin R. Anderson, and David E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/-tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 897–904, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[WC94] Xuemei Wang and Jaime Carbonell. Learning by observation and practice: Towards real applications of planning systems. In *AAAI Fall Symposium on Planning and Learning: On to Real Applications*, pages 156–160, 1994.

[Wel94] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[Wel99] Daniel S. Weld. Recent advances in ai planning. *AI Magazine*, 20(2):93–123, 1999.

[WSK+05] David E. Wilkins, Stephen Smith, Laurence Kramer, Thomas J. Lee, and Timothy W. Rauenbusch. Execution monitoring and replanning with incremental and collaborative scheduling. In *Workshop on Multiagent Planning and Scheduling, The 15th International Conference on Automated Planning & Scheduling*, pages 29–35, June 2005.

[WWA+09] Lijie Wen, Jianmin Wang, Wil M. Aalst, Biqing Huang, and Jiaguang Sun. A novel approach for process mining based on event types. *J. Intell. Inf. Syst.*, 32:163–190, April 2009.

[WWvdA+10] Lijie Wen, Jianmin Wang, Wil M. P. van der Aalst, Biqing Huang, and Jiaguang Sun. Mining process models with prime invisible tasks. *Data Knowl. Eng.*, 69:999–1021, October 2010.

[YFG06] Sung Wook Yoon, Alan Fern, and Robert Givan. Learning heuristic functions from relaxed plans. In Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey, editors, *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pages 162–171, Cumbria, UK, June 6-10 2006. AAAI.

[YWJ07] Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted max-sat. *Artif. Intell.*, 171(2-3):107–143, 2007.

[ZGLL03] Shao-Hua Zhang, Ning Gu, Jie-Xin Lian, and Sai-Han Li. Workflow process mining based on machine learning. In *International Conference on Machine Learning and Cybernetics*, volume 4, pages 2319–2323, 2-5 Nov 2003.

[zM99] Jrg zur Muehlen, Michael; Becker. WPDL - State-of-the-Art and directions of a meta-language for workflow. In L. et al. Bading, editor, *Proceedings of the 1st KnowTech Forum*, Potsdam, September 17th-19th 1999.

[zM04] Michael zur Muehlen. Organizational Management in Workflow Applications - Issues and Perspectives. *Information Technology and Management Journal.*, 5(3-4):271–291, July - October 2004. Kluwer Academic Publishers.

[ZnAY11] Hankz Hankui Zhuo, Hector Mu noz Avila, and Qiang Yang. Learning action models for multi-agent planning. In Stone Yolum, Tumer and Sonenberg, editors, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Taipei, Taiwan, May 2011. International Foundation for Autonomous Agents and Multiagent Systems.

[ZYHL10] Hankz Hankui Zhuo, Qiang Yang, Derek Hao Hu, and Lei Li. Learning complex action models with quantifiers and logical implications. *Artif. Intell.*, 174:1540–1569, December 2010.

[ZYJ10] Shijie Zhang, Jiong Yang, and Wei Jin. Sapper: Subgraph indexing and approximate matching in large graphs. *PVLDB*, 3(1):1185–1194, 2010.

Appendix A

Domains

A.1 Case Studies Definitions

A.1.1 Elevator

Listing A.1: Elevator Domain.

```
1  : move_up_slow(Lift:elevator, F1:count,  F2:count )
2    lift_at(Lift, F1) * above(F1, F2) * reachable_floor(Lift, F2) * is_slow(Lift)
3    =>
4    lift_at(Lift, F2) * !lift_at(Lift, F1) *
5    increase(Total_cost:cost, travel_slow(F1, F2)).
6
7  : move_down_slow(Lift:elevator, F1:count, F2:count )
8    lift_at(Lift, F1) * above(F2, F1) * reachable_floor(Lift, F2) * is_slow(Lift)
9    =>
10   lift_at(Lift, F2) * !lift_at(Lift, F1) *
11   increase(Total_cost:cost, travel_slow(F2, F1)).
12
13 : move_up_fast(Lift:elevator, F1:count,  F2:count )
14   lift_at(Lift, F1) * above(F1, F2) * reachable_floor(Lift, F2) * is_fast(Lift)
15   =>
16   lift_at(Lift, F2) * !lift_at(Lift, F1) *
17   increase(Total_cost:cost, travel_fast(F1, F2)).
18
19 : move_down_fast(Lift:elevator, F1:count, F2:count )
20   lift_at(Lift, F1) * above(F2, F1) * reachable_floor(Lift, F2) * is_fast(Lift)
21   =>
22   lift_at(Lift, F2) * !lift_at(Lift, F1) *
23   increase(Total_cost:cost, travel_fast(F2, F1)).
24
```

```
: board(P:passenger, Lift:elevator, F:count, N1:count, N2:count)
  lift_at(Lift, F) * passenger_at(P, F) * passengers(Lift, N1) * next(N1, N2) *
  can_hold(Lift, N2)
  =>
  !passenger_at(P, F) * boarded(P, Lift) * !passengers(Lift, N1) *
  passengers(Lift, N2).

: leave(P:passenger, Lift:elevator, F:count, N1:count, N2:count)
  lift_at(Lift, F) * boarded(P, Lift) * passengers(Lift, N1) * next(N2, N1)
  =>
  passenger_at(P, F) * !boarded(P, Lift) * !passengers(Lift, N1) *
  passengers(Lift, N2).
```

A.1.2 Mars Rover

Listing A.2: Mars Rover Domain.

```
: navigate(X:rover, Y:waypoint, Z:waypoint)
 can_traverse(X, Y, Z) * available(X) * at(X, Y) * visible(Y, Z)
 =>
 !at(X, Y) * at(X, Z).

: sample_soil(X:rover, S:store, P:waypoint)
 at(X, P) * at_soil_sample(P) * equipped_for_soil_analysis(X) *
 store_of(S, X) * empty(S)
 =>
 !empty(S) * full(S) * have_soil_analysis(X, P) * !at_soil_sample(P).

: sample_rock(X:rover, S:store, P:waypoint)
 at(X, P) * at_rock_sample(P) * equipped_for_rock_analysis(X) *
 store_of(S, X) * empty(S) =>
 !empty(S) * full(S) * have_rock_analysis(X, P) * !at_rock_sample(P).

: drop(X:rover, Y:store)
 store_of(Y, X) * full(Y) => !full(Y) * empty(Y).

: calibrate(R:rover, I:camera, T:objective, W:waypoint)
 equipped_for_imaging(R) * calibration_target(I, T) * at(R, W) *
 visible_from(T, W) * on_board(I, R) =>
 calibrated(I, R).

: take_image(R:rover, P:waypoint, O:objective, I:camera, M:mode)
 calibrated(I, R) * on_board(I, R) * equipped_for_imaging(R) *
 supports(I, M) * visible_from(O, P) * at(R, P)
 =>
 have_image(R, O, M) * !calibrated(I, R).
```

```
: communicate_soil_data(R:rover, L:lander, P:waypoint, X:waypoint, Y:waypoint)
 at(R, X) * at_lander(L, Y) * have_soil_analysis(R, P) * visible(X, Y) *
 available(R) * channel_free(L)
 =>
 communicated_soil_data(P).

: communicate_rock_data(R:rover, L:lander, P:waypoint, X:waypoint, Y:waypoint)
 at(R, X) * at_lander(L, Y) * have_rock_analysis(R, P) * visible(X, Y) *
 available(R) * channel_free(L)
 =>
 communicated_rock_data(P).

: communicate_image_data(R:rover, L:lander, O:objective, M:mode, X:waypoint,
                         Y:waypoint)
 at(R, X) * at_lander(L, Y) * have_image(R, O, M) * visible(X, Y) *
 available(R) * channel_free(L)
 =>
 communicated_image_data(O, M).
```

A.1.3 Woodwork

Listing A.3: Woodwork Domain.

```

: do_immersion_varnish(X:part, M:immersion_varnisher, Newcolour:acolour,
                       Surface:surface)
  available(X) * has_colour(M, Newcolour) * surface_condition(X, Surface) *
  is_smooth(Surface) * untreated(X) * colour_natural(X)
  =>
  increase(Total_cost:cost, immersion_varnish_cost(X)) * !untreated(X) *
  varnished(X) * !colour_natural(X) * colour(X, Newcolour).

: do_spray_varnish(X:part, M:spray_varnisher, Newcolour:acolour,
                   Surface:surface)
  available(X) * has_colour(M, Newcolour) * surface_condition(X, Surface) *
  is_smooth(Surface) * untreated(X) * colour_natural(X)
  =>
  increase(Total_cost:cost, spray_varnish_cost(X)) * !untreated(X) *
  varnished(X) * !colour_natural(X) * colour(X, Newcolour).

: do_glaze(X:part, M:glazer, Newcolour:acolour)
  available(X) * has_colour(M, Newcolour) * untreated(X) * colour_natural(X)
  =>
  increase(Total_cost:cost, glaze_cost(X)) * !untreated(X) *
  glazed(X) * !colour_natural(X) * colour(X, Newcolour).

: do_grind(X:part, M:grinder, Oldsurface:surface, Oldcolour:acolour,
```

```
25            Oldtreatment:treatmentstatus, Newtreatment:treatmentstatus)
26    available(X) * surface_condition(X, Oldsurface) * is_smooth(Oldsurface) *
27    colour(X, Oldcolour) * treatment(X, Oldtreatment) *
28    grind_treatment_change(Oldtreatment, Newtreatment)
29    =>
30    increase(Total_cost:cost, grind_cost(X)) *
31    !surface_condition(X, Oldsurface) *
32    surface_condition(X, Newsurface:surface) * is_smooth(Newsurface) *
33    !treatment(X, Oldtreatment) *
34    treatment(X, Newtreatment) * !colour(X, Oldcolour) * colour_natural(X).
35
36  : do_plane(X:part, M:planer, Oldsurface:surface, Oldcolour:acolour,
37            Oldtreatment:treatmentstatus)
38    available(X) * surface_condition(X, Oldsurface) *
39    treatment(X, Oldtreatment) * colour(X, Oldcolour)
40    =>
41    increase(Total_cost:cost, plane_cost(X)) *
42    !surface_condition(X, Oldsurface) *
43    surface_condition(X,Newsurface:surface) * is_smooth(Newsurface) *
44    !treatment(X, Oldtreatment) * untreated(X) * !colour(X, Oldcolour) *
45    colour_natural(X).
46
47  : load_highspeed_saw(B:board, M:highspeed_saw)
48    empty(M) * available(B) =>
49    increase(Total_cost:cost, load_cost(B)) * !available(B) * !empty(M) *
50    in_highspeed_saw(B, M).
51
52  : unload_highspeed_saw(B:board, M:highspeed_saw)
53    in_highspeed_saw(B, M) =>
54    increase(Total_cost:cost, unload_cost(B)) * available(B) *
55    !in_highspeed_saw(B, M) * empty(M).
56
57  : cut_board_small(B:board, P:part, M:highspeed_saw, W:awood, Surface:surface,
58                    Size_before:aboardsize, Size_after:aboardsize)
59    unused(P) * goalsize_small(P) * in_highspeed_saw(B, M) * wood(B, W) *
60    surface_condition(B, Surface) * boardsize(B, Size_before) *
61    boardsize_successor(Size_after, Size_before)
62    =>
63    increase(Total_cost:cost, cut_small_cost(P)) * !unused(P) * available(P) *
64    wood(P, W) * surface_condition(P, Surface) * colour_natural(P) * untreated(P) *
65    boardsize(B, Size_after).
66
67  : cut_board_medium(B:board, P:part, M:highspeed_saw, W:awood, Surface:surface,
68    Size_before:aboardsize, S1:aboardsize, Size_after:aboardsize)
69    unused(P) * goalsize_medium(P) * in_highspeed_saw(B, M) * wood(B, W) *
70    surface_condition(B, Surface) * boardsize(B, Size_before) *
71    boardsize_successor(Size_after, S1) * boardsize_successor(S1, Size_before)
```

```
72    =>
73    increase(Total_cost:cost, cut_medium_cost(P)) * !unused(P) * available(P) *
74    wood(P, W) * surface_condition(P, Surface) * colour_natural(P) * untreated(P) *
75    boardsize(B, Size_after).
76
77  : cut_board_large(B:board, P:part, M:highspeed_saw, W:awood, Surface:surface,
78    Size_before:aboardsize, S1:aboardsize, S2:aboardsize, Size_after:aboardsize)
79    unused(P) * goalsize_large(P) * in_highspeed_saw(B, M) * wood(B, W) *
80    surface_condition(B, Surface) * boardsize(B, Size_before) *
81    boardsize_successor(Size_after, S1) * boardsize_successor(S1, S2) *
82    boardsize_successor(S2, Size_before)
83    =>
84    increase(Total_cost:cost, cut_large_cost(P)) * !unused(P) * available(P) *
85    wood(P, W) * surface_condition(P, Surface) * colour_natural(P) * untreated(P) *
86    boardsize(B, Size_after).
87
88  : do_saw_small(B:board, P:part, M:saw, W:awood, Surface:surface,
89    Size_before:aboardsize, Size_after:aboardsize)
90    unused(P) * goalsize_small(P) * available(B) * wood(B, W) *
91    surface_condition(B, Surface) * boardsize(B, Size_before) *
92    boardsize_successor(Size_after, Size_before)
93    =>
94    increase(Total_cost:cost, saw_small_cost(P)) * !unused(P) * available(P) *
95    wood(P, W) * surface_condition(P, Surface) * colour_natural(P) *
96    untreated(P) * boardsize(B, Size_after).
97
98  : do_saw_medium(B:board, P:part, M:saw, W:awood, Surface:surface,
99    Size_before:aboardsize, S1:aboardsize, Size_after:aboardsize)
100   unused(P) * goalsize_medium(P) * available(B) * wood(B, W) *
101   surface_condition(B, Surface) * boardsize(B, Size_before) *
102   boardsize_successor(Size_after, S1) * boardsize_successor(S1, Size_before)
103   =>
104   increase(Total_cost:cost, saw_medium_cost(P)) * !unused(P) * available(P) *
105   wood(P, W) * surface_condition(P, Surface) * colour_natural(P) *
106   untreated(P) * boardsize(B, Size_after).
107
108 : do_saw_large(B:board, P:part, M:saw, W:awood, Surface:surface,
109   Size_before:aboardsize, S1:aboardsize, S2:aboardsize, Size_after:aboardsize)
110   unused(P) * goalsize_large(P) * available(B) * wood(B, W) *
111   surface_condition(B, Surface) * boardsize(B, Size_before) *
112   boardsize_successor(Size_after, S1) * boardsize_successor(S1, S2) *
113   boardsize_successor(S2, Size_before)
114   =>
115   increase(Total_cost:cost, saw_large_cost(P)) * !unused(P) * available(P) *
116   wood(P, W) * surface_condition(P, Surface) * colour_natural(P) *
117   untreated(P) * boardsize(B, Size_after).
```

A.1.4 Bridge inspection

Listing A.4: Bridge Inspection Domain.

```
/* riding surface */
: check_riding_surface(B:bridge)
  at(B) * part_of(deck(S1:riding_surface),B) * checked_surface(B, deck(S1)) *
  part_of(roadway(S2:riding_surface),B) * checked_surface(B, roadway(S2)) *
  part_of(parapet(S3:riding_surface),B) * checked_surface(B, parapet(S3)) *
  part_of(deck_joints(S4:riding_surface),B) * checked_surface(B, deck_joints(S4))
  =>
  checked_riding_surface(B).

: check_surface_1(B:bridge, deck(S:riding_surface))
  checked_for_potholes(B, S) *
  checked_for_cracking(B, S) *
  checked_for_excessive_ware(B, S) *
  checked_for_seepage(B, S) *
  sounded_for_hollow_areas(B, S)
  =>
  checked_surface(B, deck(S)).

: check_surface_2(B:bridge, roadway(S:riding_surface))
  checked_for_potholes(B, S) *
  checked_for_cracking(B, S) *
  checked_for_excessive_ware(B, S) *
  checked_for_seepage(B, S) *
  sounded_for_hollow_areas(B, S)
  =>
  checked_surface(B, roadway(S)).

: check_surface_3(B:bridge, parapet(S:riding_surface))
  checked_for_cracking(B, S) *
  checked_for_excessive_ware(B, S) *
  sounded_for_hollow_areas(B, S)
  =>
  checked_surface(B, parapet(S)).

: check_surface_4(B:bridge, deck_joints(S:riding_surface))
  checked_for_loose_armour(B, S) *
  checked_for_correct_expansion(B, S)
  =>
  checked_surface(B, deck_joints(S)).

/* non-deterministic */
: check_for_potholes_1(B:bridge, S:riding_surface)
  !checked_for_potholes(B, S)
  =>
```

```
45    checked_for_potholes(B, S) * no_potholes(B, S).
46
47  : check_for_potholes_2(B:bridge, S:riding_surface)
48    !checked_for_potholes(B, S)
49    =>
50    checked_for_potholes(B, S) * has_potholes(B, S).
51
52  : check_for_cracking_1(B:bridge, S:riding_surface)
53    !checked_for_cracking(B, S)
54    =>
55    checked_for_cracking(B, S) * no_cracking(B, S).
56
57  : check_for_cracking_2(B:bridge, S:riding_surface)
58    !checked_for_cracking(B, S)
59    =>
60    checked_for_cracking(B, S) * has_cracking(B, S).
61
62  : check_for_excessive_ware_1(B:bridge, S:riding_surface)
63    !checked_for_excessive_ware(B, S)
64    =>
65    checked_for_excessive_ware(B, S) * no_excessive_ware(B, S).
66
67  : check_for_excessive_ware_2(B:bridge, S:riding_surface)
68    !checked_for_excessive_ware(B, S)
69    =>
70    checked_for_excessive_ware(B, S) * has_excessive_ware(B, S).
71
72  : check_for_seepage_1(B:bridge, S:riding_surface)
73    !checked_for_seepage(B, S)
74    =>
75    checked_for_seepage(B, S) * no_seepage(B, S).
76
77  : check_for_seepage_2(B:bridge, S:riding_surface)
78    !checked_for_seepage(B, S)
79    =>
80    checked_for_seepage(B, S) * has_seepage(B, S).
81
82  : check_for_loose_armour_1(B:bridge, S:riding_surface)
83    !checked_for_loose_armour(B, S)
84    =>
85    checked_for_loose_armour(B, S) * no_loose_armour(B, S).
86
87  : check_for_loose_armour_2(B:bridge, S:riding_surface)
88    !checked_for_loose_armour(B, S)
89    =>
90    checked_for_loose_armour(B, S) * has_loose_armour(B, S).
91
```

```
92  : check_for_correct_expansion_1(B:bridge, S:riding_surface)
93    !checked_for_correct_expansion(B, S)
94    =>
95    checked_for_correct_expansion(B, S) * no_correct_expansion(B, S).
96
97  : check_for_correct_expansion_2(B:bridge, S:riding_surface)
98    !checked_for_correct_expansion(B, S)
99    =>
100   checked_for_correct_expansion(B, S) * has_correct_expansion(B, S).
101
102 : sound_for_hollow_areas_1(B:bridge, S:riding_surface)
103   !sounded_for_hollow_areas(B, S)
104   =>
105   sounded_for_hollow_areas(B, S) * no_hollow_areas(B, S).
106
107 : sound_for_hollow_areas_2(B:bridge, S:riding_surface)
108   !sounded_for_hollow_areas(B, S)
109   =>
110   sounded_for_hollow_areas(B, S) * has_hollow_areas(B, S).
111   /* 19 */
112
113 /* super structure, beam is a girder */
114 : check_super_structure(B:bridge)
115   at(B) * part_of(beam(S1:support_structure),B) * checked_girder(B, beam(S1)) *
116   part_of(bearings(S2:support_structure),B) * checked_bearings(B, bearings(S2))
117   =>
118   checked_super_structure(B).
119
120 : check_girder_1(B:bridge, beam(S:support_structure))
121   made_of_wood(B, S) *
122   checked_for_woodrot(B, S) *
123   checked_for_crushing(B, S) *
124   checked_for_splitting(B, S) *
125   checked_for_cracking(B, S)
126   =>
127   checked_girder(B, beam(S)).
128
129 : check_girder_2(B:bridge, beam(S:support_structure))
130   made_of_concrete(B, S) *
131   checked_for_cracking(B, S) *
132   checked_for_spalling(B, S) *
133   sounded_for_hollow_areas(B, S)
134   =>
135   checked_girder(B, beam(S)).
136
137 : check_girder_3(B:bridge, beam(S:support_structure))
138   made_of_steel(B, S) *
```

```
139   checked_for_cracking(B, S) *
140   checked_for_corrosion(B, S) *
141   checked_for_peeling_paint(B, S)
142   =>
143   checked_girder(B, beam(S)).
144
145 : check_bearings(B:bridge, bearings(S:support_structure))
146   checked_for_excessive_deformation(B, S) *
147   checked_for_temperature_change_movement(B, S)
148   =>
149   checked_bearings(B, bearings(S)).
150
151 /* non-deterministic */
152 : check_for_woodrot_1(B:bridge, S:support_structure)
153   made_of_wood(B, S) * !checked_for_woodrot(B, S)
154   =>
155   checked_for_woodrot(B, S) * no_woodrot(B, S).
156
157 : check_for_woodrot_2(B:bridge, S:support_structure)
158   made_of_wood(B, S) * !checked_for_woodrot(B, S)
159   =>
160   checked_for_woodrot(B, S) * has_woodrot(B, S).
161
162 : sound_for_hollow_areas_1(B:bridge, S:support_structure)
163   !sounded_for_hollow_areas(B, S)
164   =>
165   sounded_for_hollow_areas(B, S) * no_hollow_areas(B, S).
166
167 : sound_for_hollow_areas_2(B:bridge, S:support_structure)
168   !sounded_for_hollow_areas(B, S)
169   =>
170   sounded_for_hollow_areas(B, S) * has_hollow_areas(B, S).
171
172 : check_for_crushing_1(B:bridge, S:support_structure)
173   made_of_wood(B, S) * !checked_for_crushing(B, S)
174   =>
175   checked_for_crushing(B, S) * no_crushing(B, S).
176
177 : check_for_crushing_2(B:bridge, S:support_structure)
178   made_of_wood(B, S) * !checked_for_crushing(B, S)
179   =>
180   checked_for_crushing(B, S) * has_crushing(B, S).
181
182 : check_for_splitting_1(B:bridge, S:support_structure)
183   made_of_wood(B, S) * !checked_for_splitting(B, S)
184   =>
185   checked_for_splitting(B, S) * no_splitting(B, S).
```

```
186
187 : check_for_splitting_2(B:bridge, S:support_structure)
188   made_of_wood(B, S) * !checked_for_splitting(B, S)
189   =>
190   checked_for_splitting(B, S) * has_splitting(B, S).
191
192 : check_for_cracking_1(B:bridge, S:support_structure)
193   !checked_for_cracking(B, S)
194   =>
195   checked_for_cracking(B, S) * no_cracking(B, S).
196
197 : check_for_cracking_2(B:bridge, S:support_structure)
198   !checked_for_cracking(B, S)
199   =>
200   checked_for_cracking(B, S) * has_cracking(B, S).
201
202 : check_for_spalling_1(B:bridge, S:support_structure)
203   made_of_concrete(B, S) * !checked_for_spalling(B, S)
204   =>
205   checked_for_spalling(B, S) * no_spalling(B, S).
206
207 : check_for_spalling_2(B:bridge, S:support_structure)
208   made_of_concrete(B, S) * !checked_for_spalling(B, S)
209   =>
210   checked_for_spalling(B, S) * has_spalling(B, S).
211
212 : check_for_corrosion_1(B:bridge, S:support_structure)
213   made_of_steel(B, S) * !checked_for_corrosion(B, S)
214   =>
215   checked_for_corrosion(B, S) * no_corrosion(B, S).
216
217 : check_for_corrosion_2(B:bridge, S:support_structure)
218   made_of_steel(B, S) * !checked_for_corrosion(B, S)
219   =>
220   checked_for_corrosion(B, S) * has_corrosion(B, S).
221
222 : check_for_peeling_paint_1(B:bridge, S:support_structure)
223   made_of_steel(B, S) * !checked_for_peeling_paint(B, S)
224   =>
225   checked_for_peeling_paint(B, S) * no_peeling_paint(B, S).
226
227 : check_for_peeling_paint_2(B:bridge, S:support_structure)
228   made_of_steel(B, S) * !checked_for_peeling_paint(B, S)
229   =>
230   checked_for_peeling_paint(B, S) * has_peeling_paint(B, S).
231
232 : check_for_excessive_deformation_1(B:bridge, S:support_structure)
```

```
233   !checked_for_excessive_deformation(B, S)
234   =>
235   checked_for_excessive_deformation(B, S) * no_excessive_deformation(B, S).
236
237 : check_for_excessive_deformation_2(B:bridge, S:support_structure)
238   !checked_for_excessive_deformation(B, S)
239   =>
240   checked_for_excessive_deformation(B, S) * has_excessive_deformation(B, S).
241
242 : check_for_temperature_change_movement_1(B:bridge, S:support_structure)
243   !checked_for_temperature_change_movement(B, S)
244   =>
245   checked_for_temperature_change_movement(B, S) * no_temperature_change_movement(B, S).
246
247 : check_for_temperature_change_movement_2(B:bridge, S:support_structure)
248   !checked_for_temperature_change_movement(B, S)
249   =>
250   checked_for_temperature_change_movement(B, S) * has_temperature_change_movement(B, S).
251  /* 45 */
252
253 /* sub structure */
254 : check_sub_structure(B:bridge)
255   at(B) *
256   part_of(pier_caps(S1:support_structure),B) * checked_pier_caps(B, pier_caps(S1)) *
257   part_of(columns(S2:support_structure),B) * checked_columns(B, columns(S2)) *
258   part_of(piles(S3:support_structure),B) * checked_piles(B, piles(S3))
259   =>
260   checked_sub_structure(B).
261
262 : check_pier_caps_1(B:bridge, pier_caps(S:support_structure))
263   made_of_wood(B, S) *
264   checked_for_woodrot(B, S) *
265   checked_for_crushing(B, S) *
266   checked_for_splitting(B, S) *
267   checked_for_cracking(B, S)
268   =>
269   checked_pier_caps(B, pier_caps(S)).
270
271 : check_pier_caps_2(B:bridge, pier_caps(S:support_structure))
272   made_of_concrete(B, S) *
273   checked_for_cracking(B, S) *
274   checked_for_spalling(B, S) *
275   sounded_for_hollow_areas(B, S)
276   =>
277   checked_pier_caps(B, pier_caps(S)).
278
279 : check_pier_caps_3(B:bridge, pier_caps(S:support_structure))
```

```
280   made_of_steel(B, S) *
281   checked_for_cracking(B, S) *
282   checked_for_corrosion(B, S) *
283   checked_for_peeling_paint(B, S)
284   =>
285   checked_pier_caps(B, pier_caps(S)).
286
287 : check_columns_1(B:bridge, columns(S:support_structure))
288   made_of_wood(B, S) *
289   checked_for_woodrot(B, S) *
290   checked_for_crushing(B, S) *
291   checked_for_splitting(B, S) *
292   checked_for_cracking(B, S)
293   =>
294   checked_columns(B, columns(S)).
295
296 : check_columns_2(B:bridge, columns(S:support_structure))
297   made_of_concrete(B, S) *
298   checked_for_cracking(B, S) *
299   checked_for_spalling(B, S) *
300   sounded_for_hollow_areas(B, S)
301   =>
302   checked_columns(B, columns(S)).
303
304 : check_columns_3(B:bridge, columns(S:support_structure))
305   made_of_steel(B, S) *
306   checked_for_cracking(B, S) *
307   checked_for_corrosion(B, S) *
308   checked_for_peeling_paint(B, S)
309   =>
310   checked_columns(B, columns(S)).
311
312 : check_piles_1(B:bridge, piles(S:support_structure))
313   made_of_wood(B, S) *
314   checked_for_woodrot(B, S) *
315   checked_for_crushing(B, S) *
316   checked_for_splitting(B, S) *
317   checked_for_cracking(B, S) *
318   checked_for_settlement(B, S) *
319   checked_for_scour(B, S)
320   =>
321   checked_piles(B, piles(S)).
322
323 : check_piles_2(B:bridge, piles(S:support_structure))
324   made_of_concrete(B, S) *
325   checked_for_cracking(B, S) *
326   checked_for_spalling(B, S) *
```

```
327   sounded_for_hollow_areas(B, S) *
328   checked_for_settlement(B, S) *
329   checked_for_scour(B, S)
330   =>
331   checked_piles(B, piles(S)).
332
333 : check_piles_3(B:bridge, piles(S:support_structure))
334   made_of_steel(B, S) *
335   checked_for_cracking(B, S) *
336   checked_for_corrosion(B, S) *
337   checked_for_peeling_paint(B, S) *
338   checked_for_settlement(B, S) *
339   checked_for_scour(B, S)
340   =>
341   checked_piles(B, piles(S)).
342
343 /* when piles sink into ground, dangerous */
344 : check_for_settlement_1(B:bridge, S:support_structure)
345   !checked_for_settlement(B, S) => checked_for_settlement(B, S) * no_settlement(B, S).
346
347 : check_for_settlement_2(B:bridge, S:support_structure)
348   !checked_for_settlement(B, S) => checked_for_settlement(B, S) * has_settlement(B, S).
349
350 /* when piles' surrounding soil is washed away or falls away, dangerous */
351 : check_for_scour_1(B:bridge, S:support_structure)
352   !checked_for_scour(B, S) => checked_for_scour(B, S) * no_scour(B, S).
353
354 : check_for_scour_2(B:bridge, S:support_structure)
355   !checked_for_scour(B, S) => checked_for_scour(B, S) * has_scour(B, S).
356   /* 60 */
357
358 /* recomendation */
359 : analyse_and_inspect(B:bridge)
360   reviewed_previous_inspection(B) *
361   checked_riding_surface(B) *
362   checked_super_structure(B) *
363   checked_sub_structure(B) *
364   checked_profile(B)
365   =>
366   analysed_and_inspected(B).
367
368 : check_profile_1(B:bridge)
369   !checked_profile(B) => checked_profile(B) * not_smooth(B).
370
371 : check_profile_2(B:bridge)
372   !checked_profile(B) => checked_profile(B) * is_smooth(B).
373
```

```
374 : generate_recomendation_1(B:bridge)
375   analysed_and_inspected(B:bridge) *
376   has_settlement(B, S:support_structure)
377   =>
378   generated_recomendation(B) * to_close_bridge(B) * do_emergency_repair(B).
379
380 : generate_recomendation_2(B:bridge)
381   analysed_and_inspected(B:bridge) *
382   has_scour(B, S:support_structure)
383   =>
384   generated_recomendation(B) * to_close_bridge(B) * do_emergency_repair(B).
385
386 : generate_recomendation_3(B:bridge)
387   analysed_and_inspected(B:bridge) *
388   not_smooth(B)
389   =>
390   generated_recomendation(B) * to_close_bridge(B) * do_emergency_repair(B).
391
392 : generate_recomendation_4(B:bridge)
393   analysed_and_inspected(B) *
394   has_potholes(B, S:riding_surface)
395   =>
396   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
397
398 : generate_recomendation_5(B:bridge)
399   analysed_and_inspected(B) *
400   has_cracking(B, S:riding_surface)
401   =>
402   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
403
404 : generate_recomendation_6(B:bridge)
405   analysed_and_inspected(B) *
406   has_excessive_ware(B, S:riding_surface)
407   =>
408   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
409
410 : generate_recomendation_7(B:bridge)
411   analysed_and_inspected(B) *
412   has_seepage(B, S:riding_surface)
413   =>
414   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
415
416 : generate_recomendation_8(B:bridge)
417   analysed_and_inspected(B) *
418   has_loose_armour(B, S:riding_surface)
419   =>
420   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
```

```
421
422 : generate_recomendation_9(B:bridge)
423   analysed_and_inspected(B) *
424   no_correct_expansion(B, S:riding_surface)
425   =>
426   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
427
428 : generate_recomendation_10(B:bridge)
429   analysed_and_inspected(B) *
430   has_hollow_areas(B, S:riding_surface)
431   =>
432   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
433
434 : generate_recomendation_11(B:bridge)
435   analysed_and_inspected(B) *
436   has_woodrot(B, S:support_structure)
437   =>
438   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
439
440 : generate_recomendation_12(B:bridge)
441   analysed_and_inspected(B) *
442   has_crushing(B, S:support_structure)
443   =>
444   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
445
446 : generate_recomendation_13(B:bridge)
447   analysed_and_inspected(B) *
448   has_splitting(B, S:support_structure)
449   =>
450   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
451
452 : generate_recomendation_14(B:bridge)
453   analysed_and_inspected(B) *
454   has_cracking(B, S:support_structure)
455   =>
456   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
457
458 : generate_recomendation_15(B:bridge)
459   analysed_and_inspected(B) *
460   has_spalling(B, S:support_structure)
461   =>
462   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
463
464 : generate_recomendation_16(B:bridge)
465   analysed_and_inspected(B) *
466   has_corrosion(B, S:support_structure)
467   =>
```

```
468   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
469
470 : generate_recomendation_17(B:bridge)
471   analysed_and_inspected(B) *
472   has_peeling_paint(B, S:support_structure)
473   =>
474   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
475
476 : generate_recomendation_18(B:bridge)
477   analysed_and_inspected(B) *
478   has_excessive_deformation(B, S:support_structure)
479   =>
480   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
481
482 /* 80 */
483 : generate_recomendation_19(B:bridge)
484   analysed_and_inspected(B) *
485   has_temperature_change_movement(B, S:support_structure)
486   =>
487   generated_recomendation(B) * to_coordinate_traffic(B) * do_routine_repair(B).
```

The Bridge Inspection domain describes the steps required to generate maintenance recommendations. Such recommendations may involve routine repair requiring only traffic rerouting or emergency repairs that needs the closing down of the bridge. This domain was inspired by a description of the process provided on-line by the Florida Department Of Transportation of the USA (Office of Maintenance, Structures Operations[1]), which we reproduce verbatim here only for reference purposes (see Figure A.1 and Section A.1.4). It is by no means complete and has not been tested by an AI planner. It serves only as a test-bed for the ML algorithm.

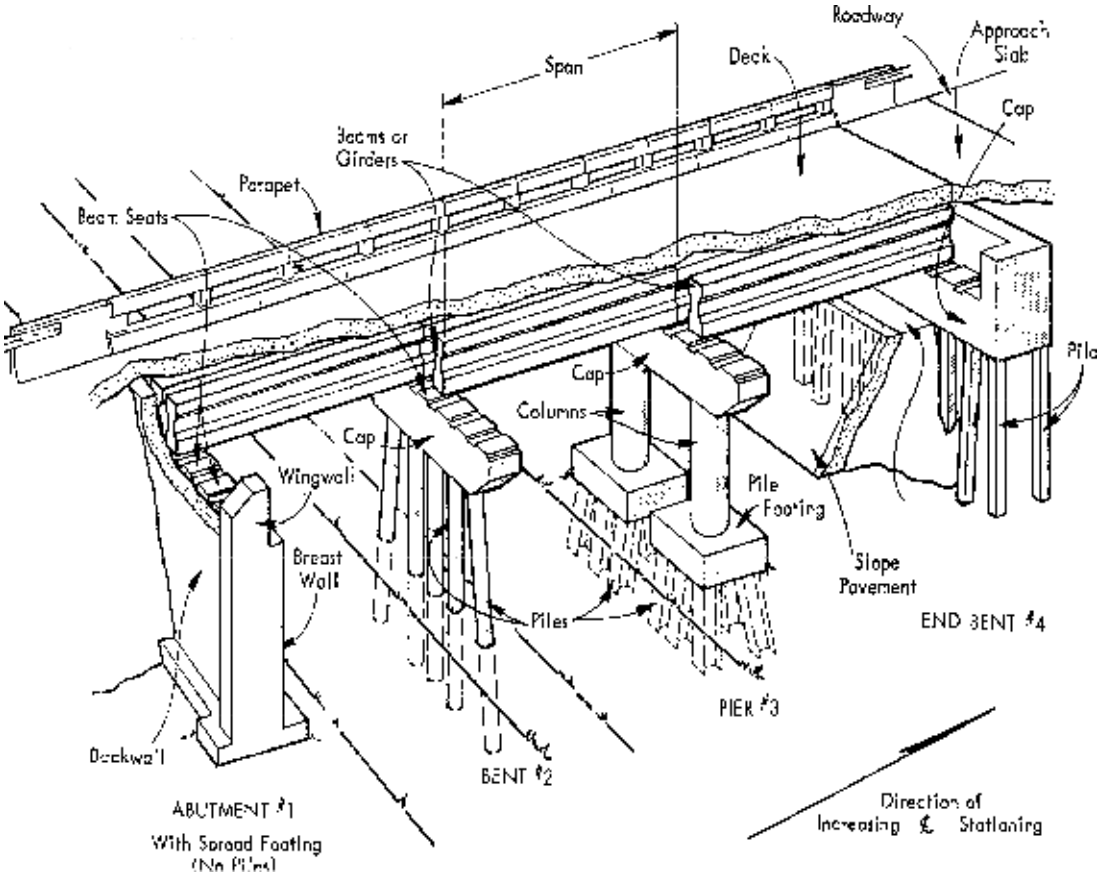
Bridge Inspection Process

"The bridge inspection process starts with the bridge inspectors reviewing the previous bridge inspection report and planning the inspection. The inspectors identify areas where defects were found in previous inspections. This allows them to determine if the defects previously identified have been repaired or have increased in size and severity. The inspectors coordinate traffic control and access equipment.

When the inspectors arrive at the bridge site they observe the bridge from a

[1] http://www.dot.state.fl.us/statemaintenanceoffice/CBR/Bridge\%20Inspection\%20Process2.pdf

Figure A.1: Bridge Inspection Terminology[1].



distance. Some major problems may be indicated if the profile of the bridge is not smooth, in other words the bridge will not look right to the experienced bridge inspector. The inspectors will then concentrate on discovering the cause and determining the extent of the problem. Depending on the exact nature of the problem emergency repair or immediate closure of the bridge may be required.

The inspectors use a systematic method to inspect the bridge, to ensure that the entire bridge is inspected. The exact order of the inspection will vary depending on the type of bridge being inspected.

The deck is the riding surface for traffic. The deck surface and road way barrier or parapet are looked at for potholes, cracking, excessive wear, and sounded for hollow areas. The deck joints are looked at for evidence of seepage, loose armor angles and if the deck joints are properly functioning to allow expansion and contraction as temperature changes.

The superstructure supports the deck and generally consists of beams or girders that may be constructed of timber, concrete or steel, and the bearings that connect the superstructure to the substructure. The inspectors pay close attention to areas of high stress and those prone to deterioration, but the entire superstructure is inspected. Timber members are inspected for wood rot, crushing, splitting and cracking. Concrete members are inspected for cracking, spalling and hollow areas. (Spalling is where a portion of the concrete has fallen away leaving a hole in the concrete.) Steel members are inspected for paint peeling, corrosion and cracking. The bearings serve to transmit loads from the superstructure to the substructure and allow the movement of the bridge that occur due to changes in temperature. The bearings are inspected for excessive deformation and evidence that they are functioning properly allowing the movements of the bridge due to temperature change.

The substructure supports the superstructure and transmits loads from the superstructure to the ground. The substructure generally consists of pier caps, columns and piles. The substructure may be constructed of timber, concrete or steel. Timber members are inspected for wood rot, crushing, splitting and cracking. Concrete members are inspected for cracking, spalling and hollow areas. Steel members are inspected for paint peeling, corrosion and cracking. In addition, the substructure is

inspected for evidence of settlement or scour. Settlement is elements of the substructure move downward due to soil conditions. Scour is the undermining of a structure due to water flow removing soil which supports the structure.

The inspectors' actions will vary depending on their findings. The inspectors will recommend immediate closure or emergency repair of the bridge if a critical condition is found that endangers the public. The inspectors will recommend a repair be performed quickly when a situation exists that if not addressed may lead to a condition that could endanger the public. The inspectors will recommend routine repairs or maintenance to correct defects that if not repaired could increase in size and severity and shorten the service life of the bridge."

A.2 Domain Statistics

The Table A.1 provides some basic statistics on the domains. The statistics includes the number of operators in the domain ($|O|$), average number of pre-condition literals (*Pre-condition*), average number of add effects literals (*Add*), the average number of delete effect literals (*Delete*) and the total number of literals (*All*) in the operators. We note that the Woodwork domain generates problems that have proven to be the hardest to solve. We can see that its operators consists of 12.6 literals on average, at least 68% more literals than any of the other domains. We also note that although the Bridge domain has by far many more literals than any other domain (more has 6 times), its processing time is reduced. This shows that processing time is greatly influenced by the transaction size (number of literals in the transactions).

Table A.1: Domain definition Statistics.

Domain	$\|O\|$	Pre-conditions	Add	Delete	All
Elevator	6	4.2	2.0	1.3	7.5
Mars Rover	9	5.0	1.2	1.0	7.0
Woodwork	13	5.9	5.1	1.6	12.6
Bridge Inspection	80	2.4	2.0	0.0	4.4

Appendix B

Results

B.1 Noiseless Data

B.1.1 Elevator

Figure B.1: Elevator: Uniform distribution, number of operators n=4, $T_s = 40$.

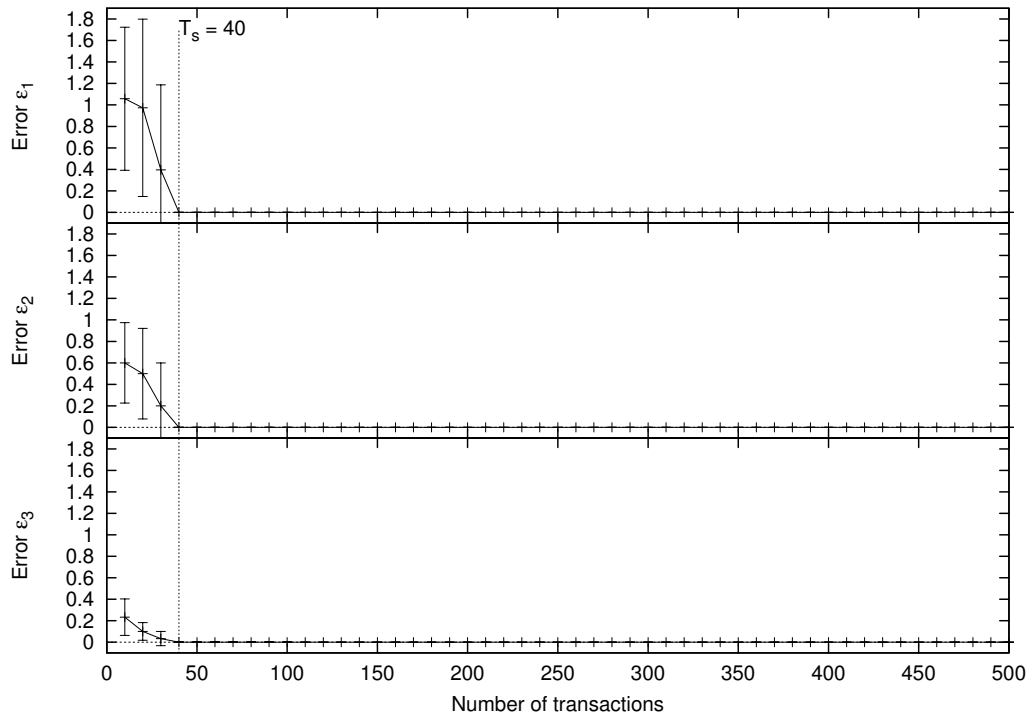


Figure B.2: Elevator: Uniform distribution, number of operators n=1..5, $T_s = 20$.

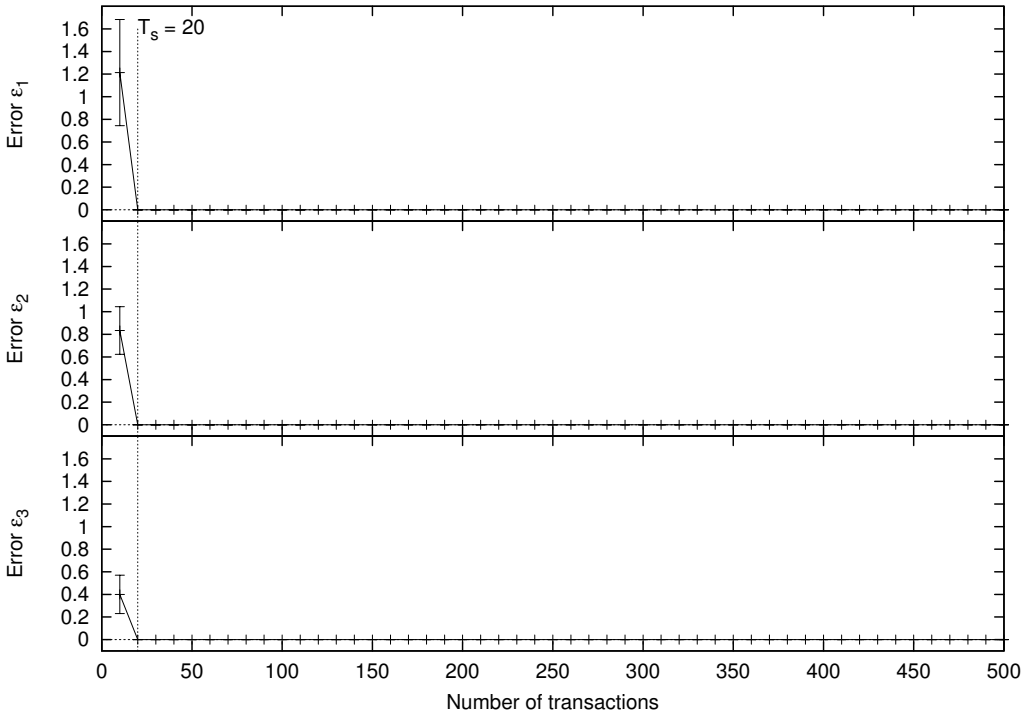


Figure B.3: Elevator: Non-uniform, number of operators n=4, $T_s = 70$.

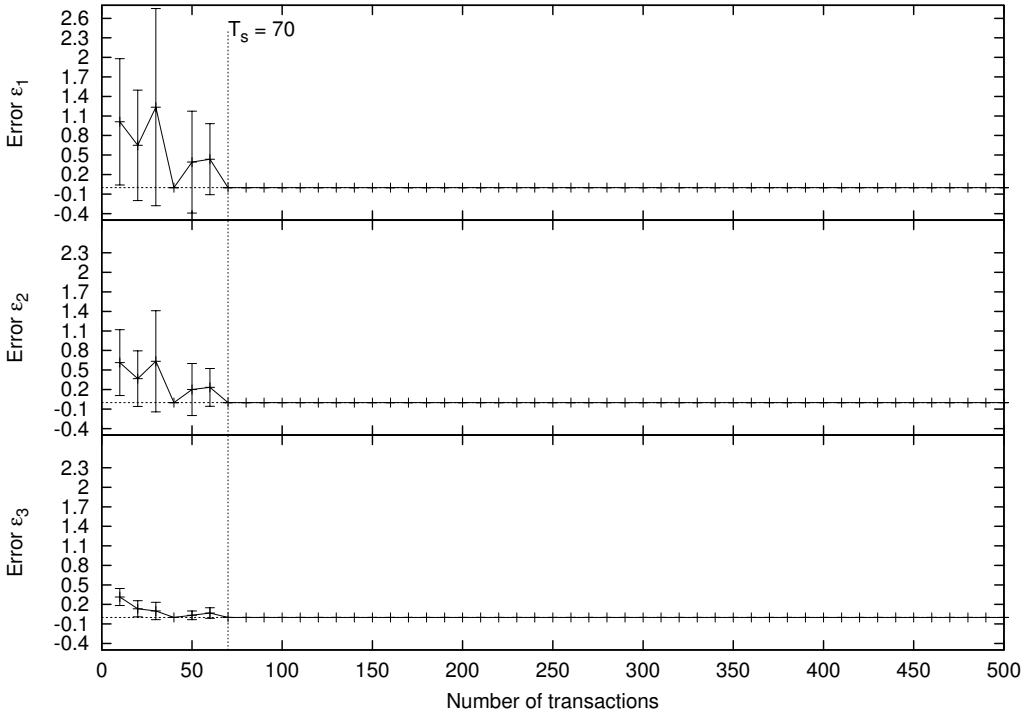
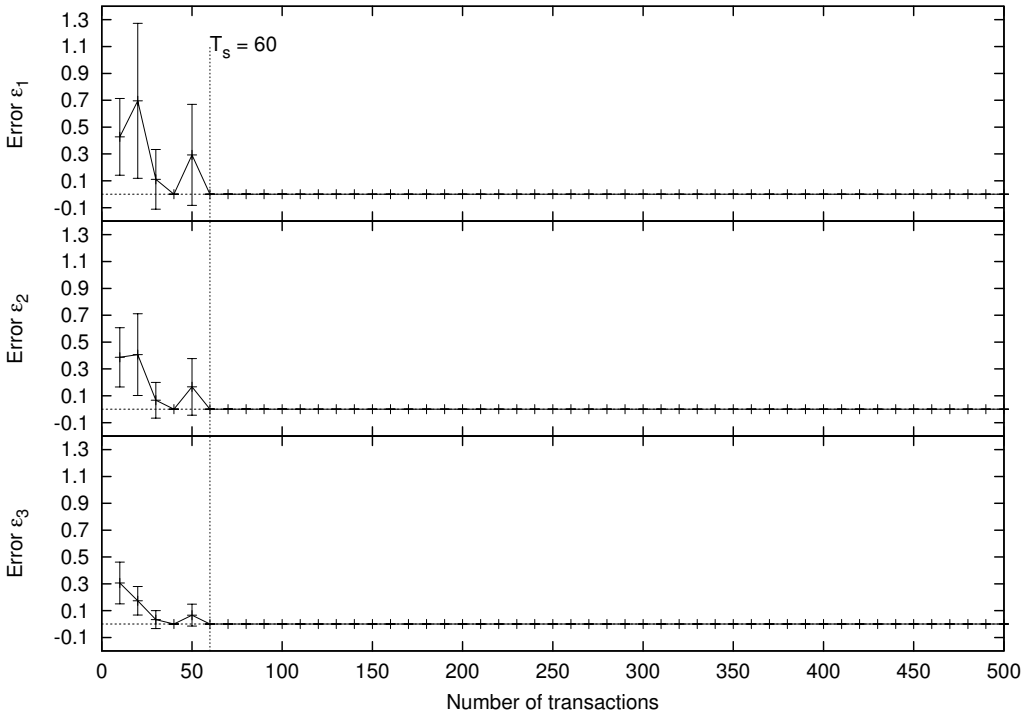


Figure B.4: Elevator: Non-uniform, number of operators n=1..5, $T_s = 60$.



B.1.2 Mars Rover

Figure B.5: Mars Rover: Uniform distribution, number of operators n=4, $T_s = 50$.

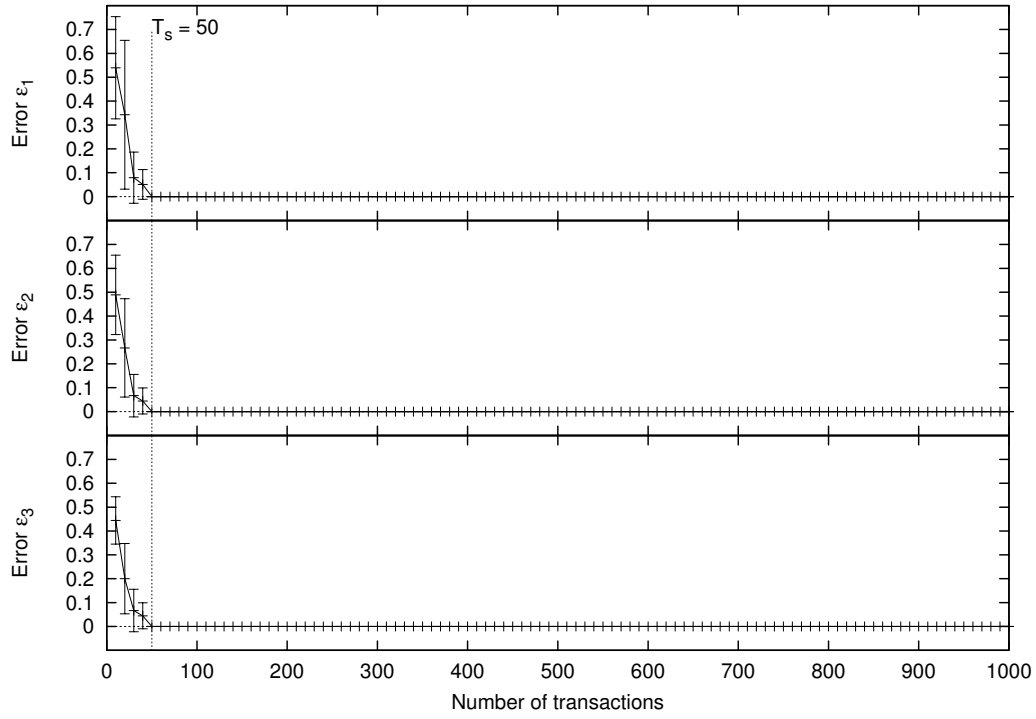


Figure B.6: Mars Rover: Uniform distribution, number of operators n=1..5, $T_s = 60$.

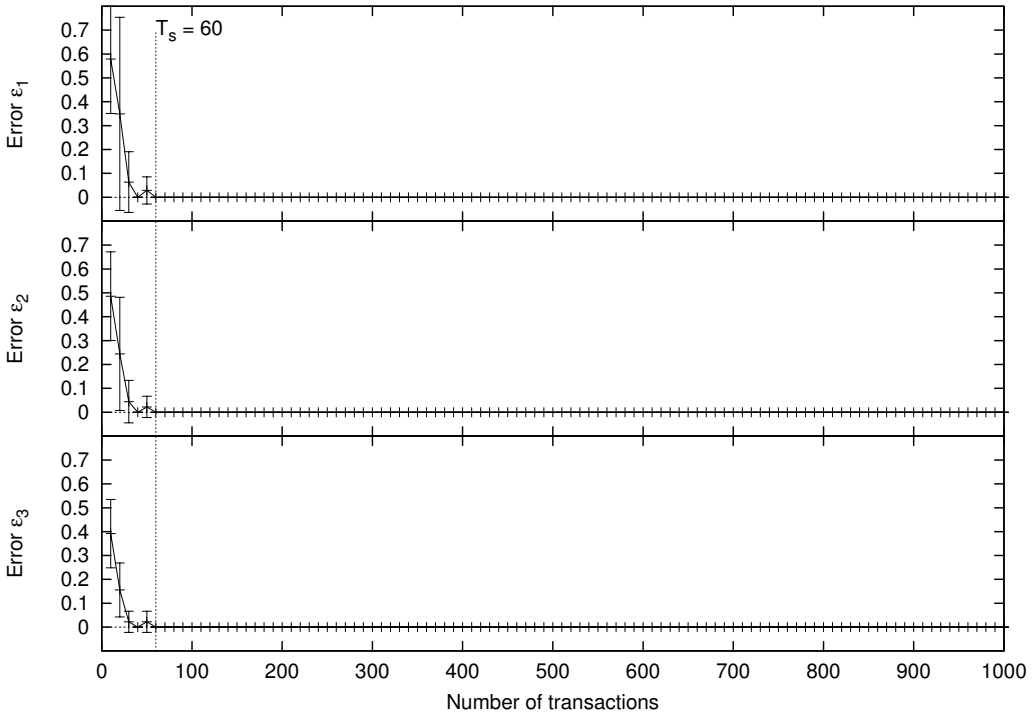


Figure B.7: Mars Rover: Non-uniform, number of operators n=4, $T_s = 340$.

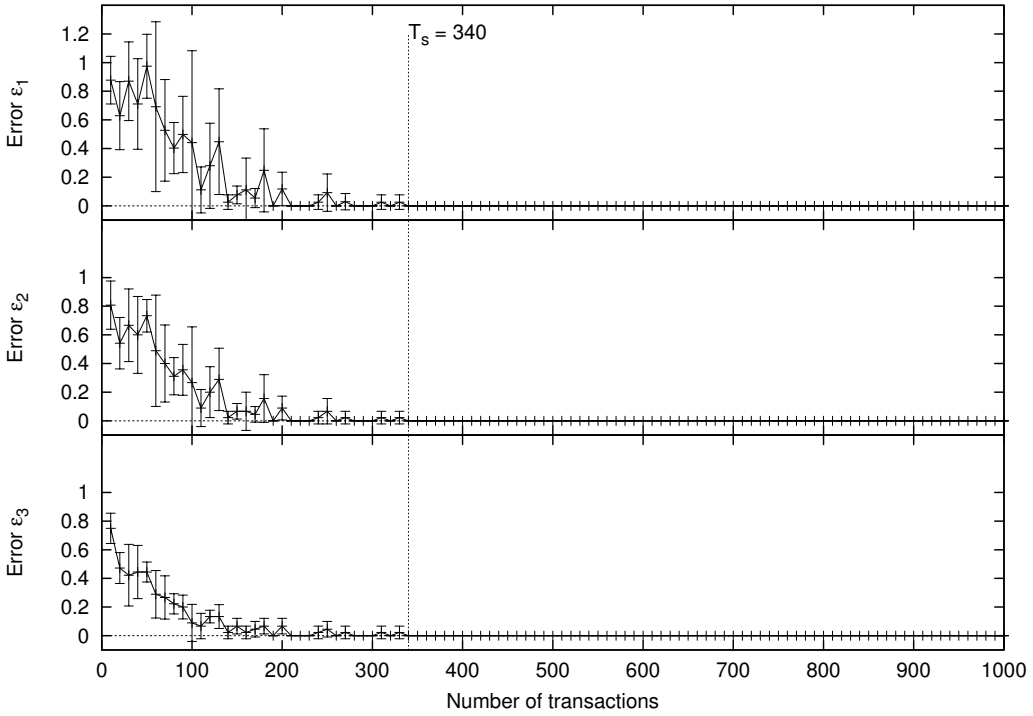
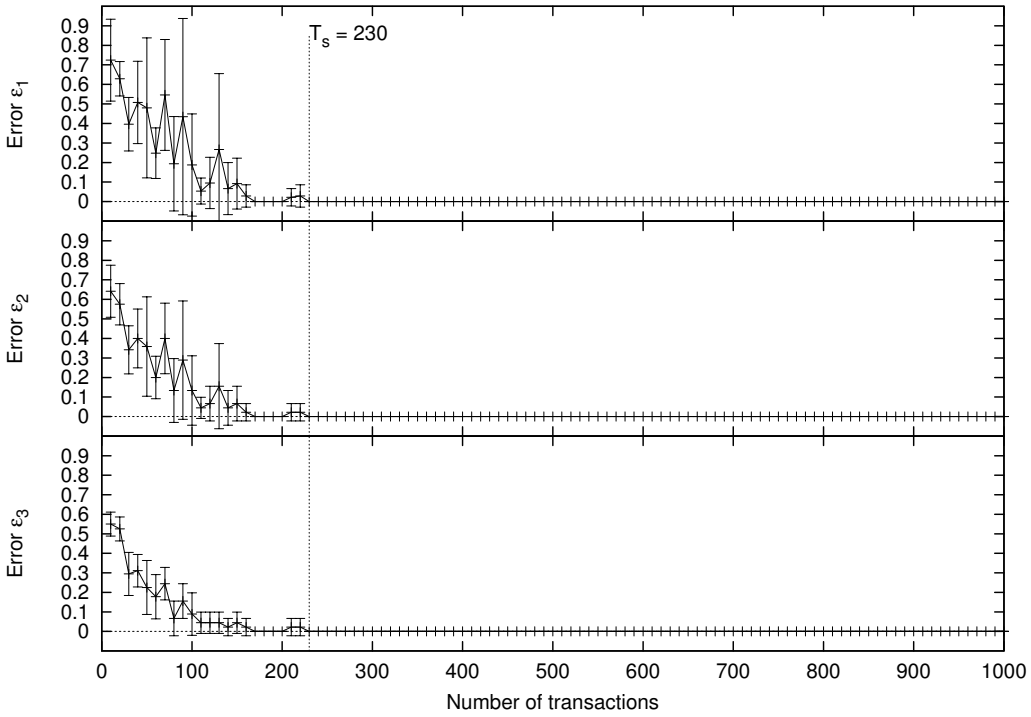


Figure B.8: Mars Rover: Non-uniform, number of operators n=1..5, $T_s = 230$.



B.1.3 Woodwork

Figure B.9: Woodwork: Uniform, number of operators n=4, $T_s = 60$.

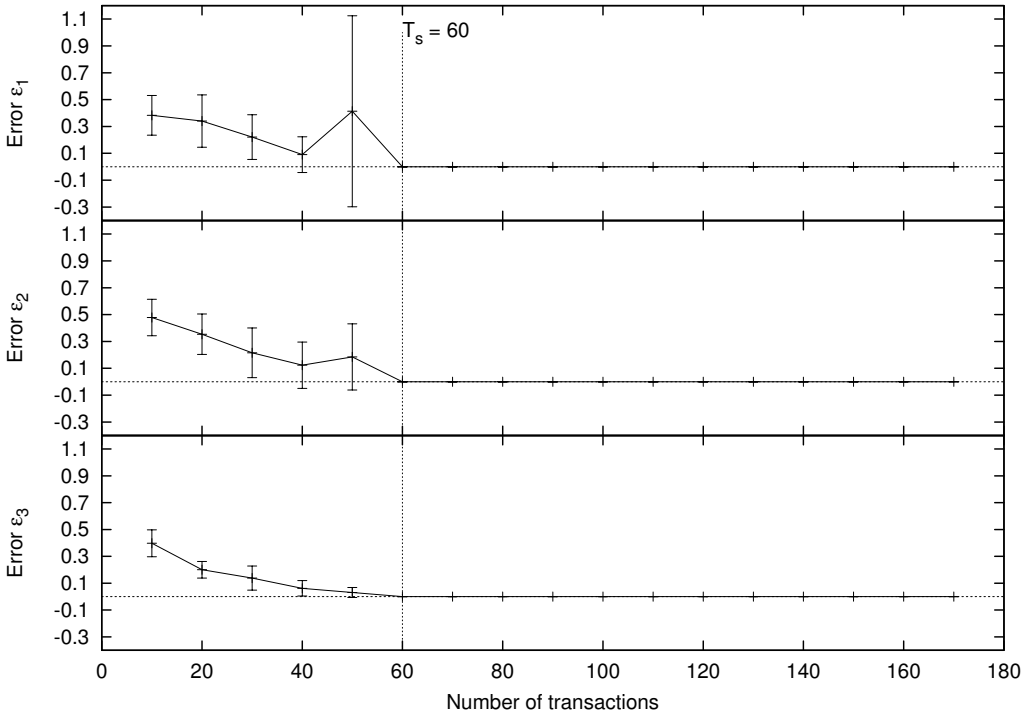


Figure B.10: Woodwork: Uniform, number of operators n=1..5, $T_s = 90$.

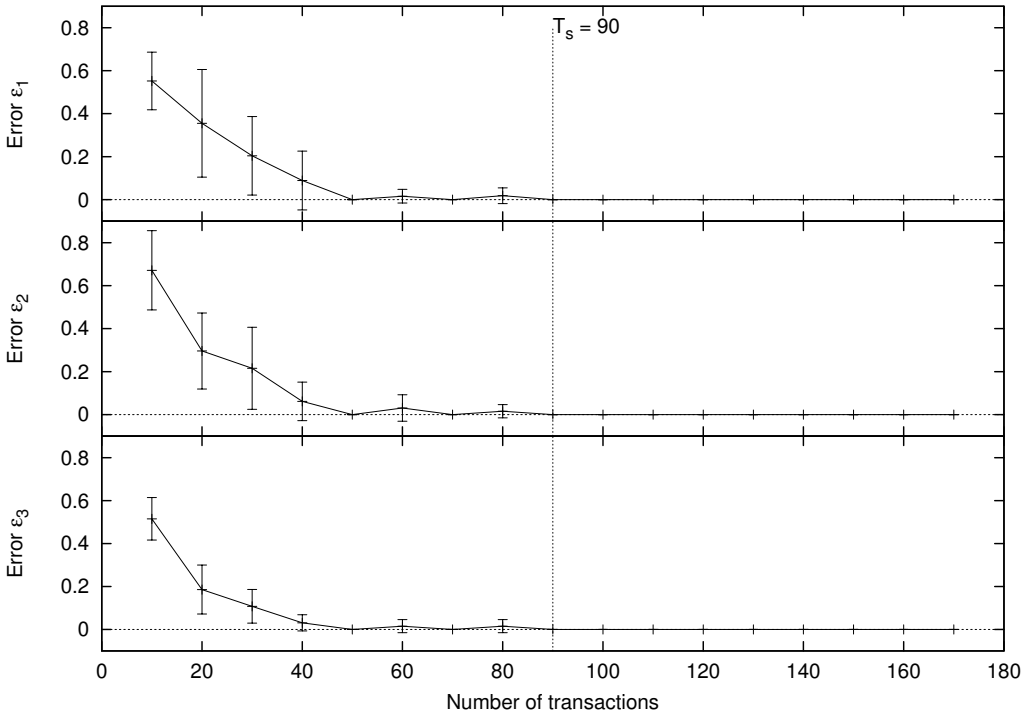


Figure B.11: Woodwork: Non-uniform, number of operators n=4, $T_s = 700$.

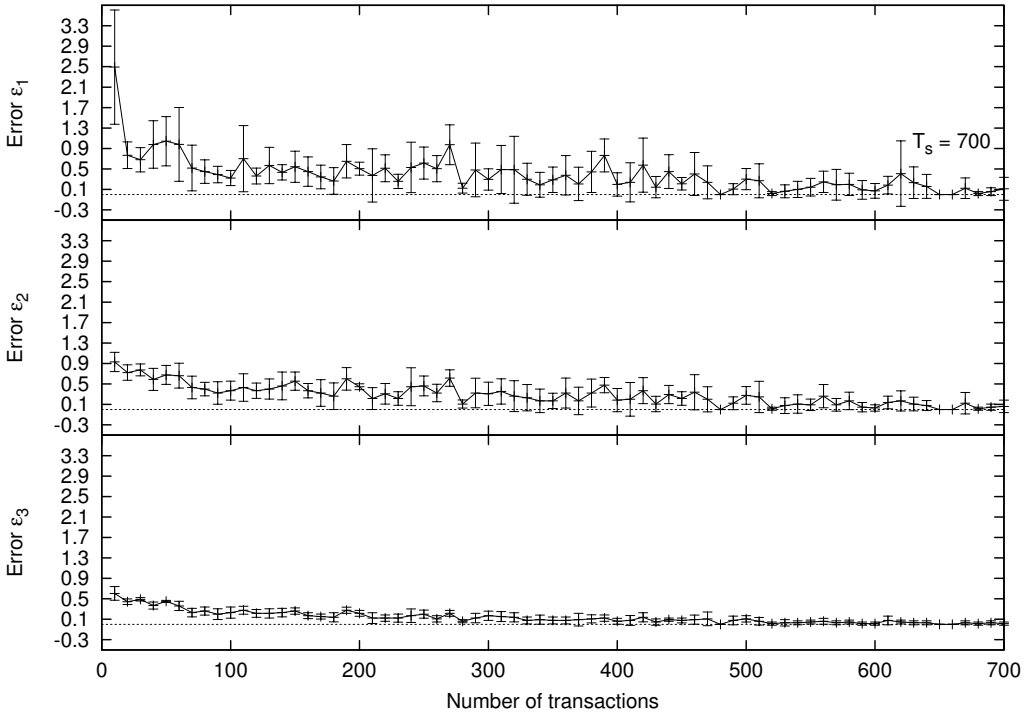
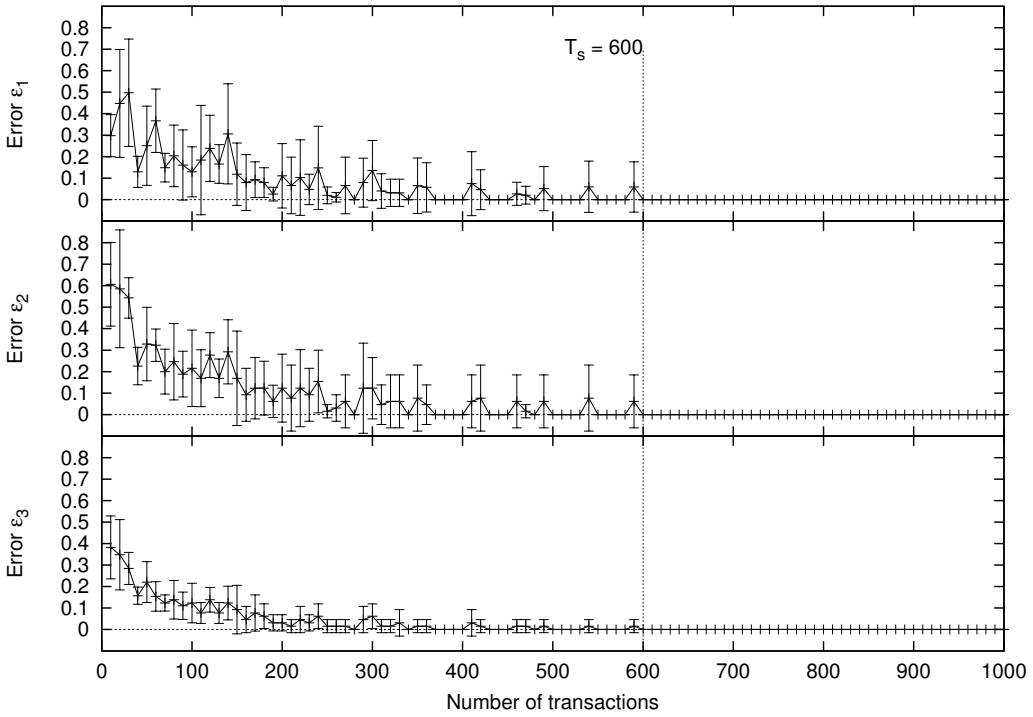


Figure B.12: Woodwork: Non-uniform, number of operators n=1..5, $T_s = 590$.



B.1.4 Bridge inspection

Figure B.13: Bridge Inspection: Uniform, number of operators n=4, $T_s = 220$.

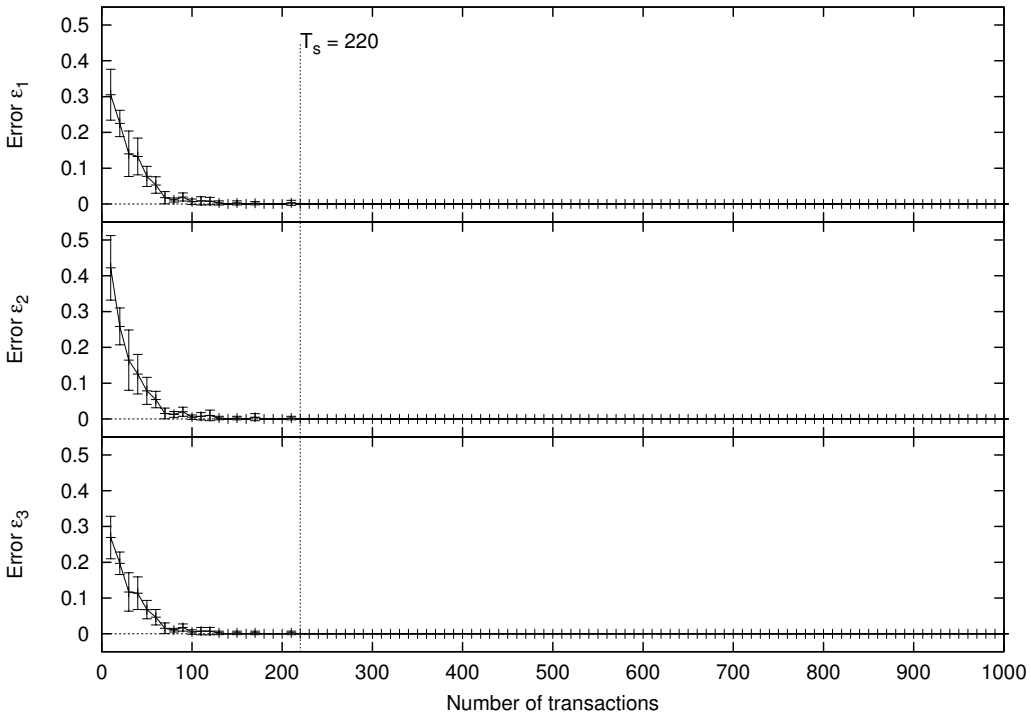


Figure B.14: Bridge Inspection: Uniform, number of operators n=1..5, $T_s = 290$.

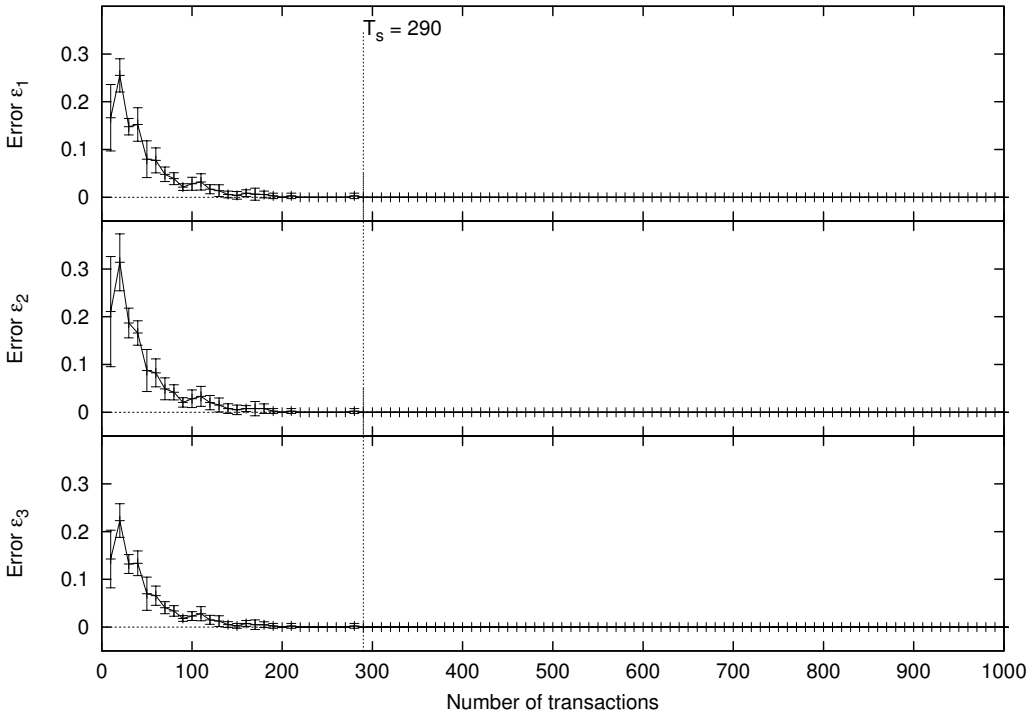


Figure B.15: Bridge Inspection: Non-uniform, number of operators n=4, $T_s = 650$.

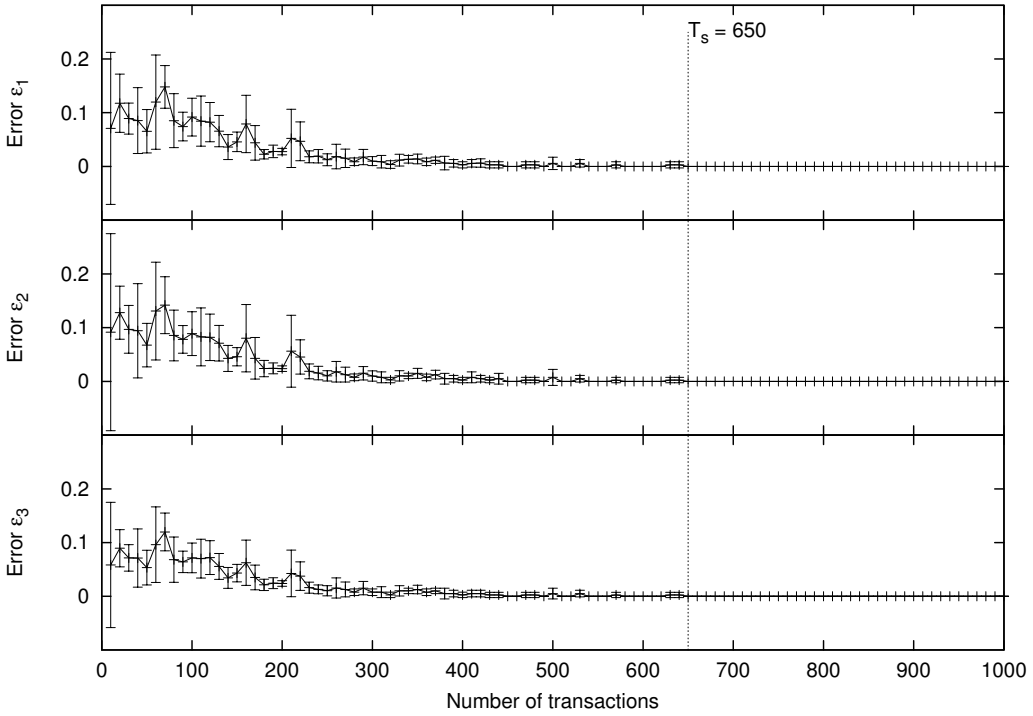
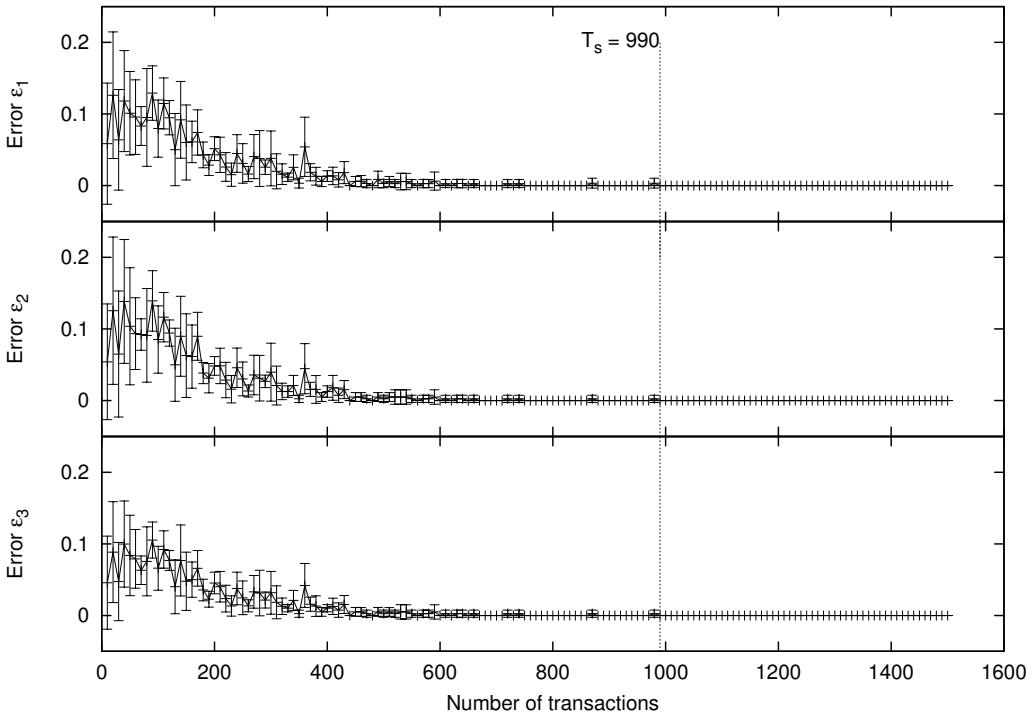


Figure B.16: Bridge Inspection: Non-uniform, number of operators n=1..5, T_s = 990.



B.2 Noisy Data

B.2.1 Elevator

Elevator with 10% Commission Noise

Figure B.17: Elevator: Uniform distribution, number of operators n=4, $T_s = 40$.

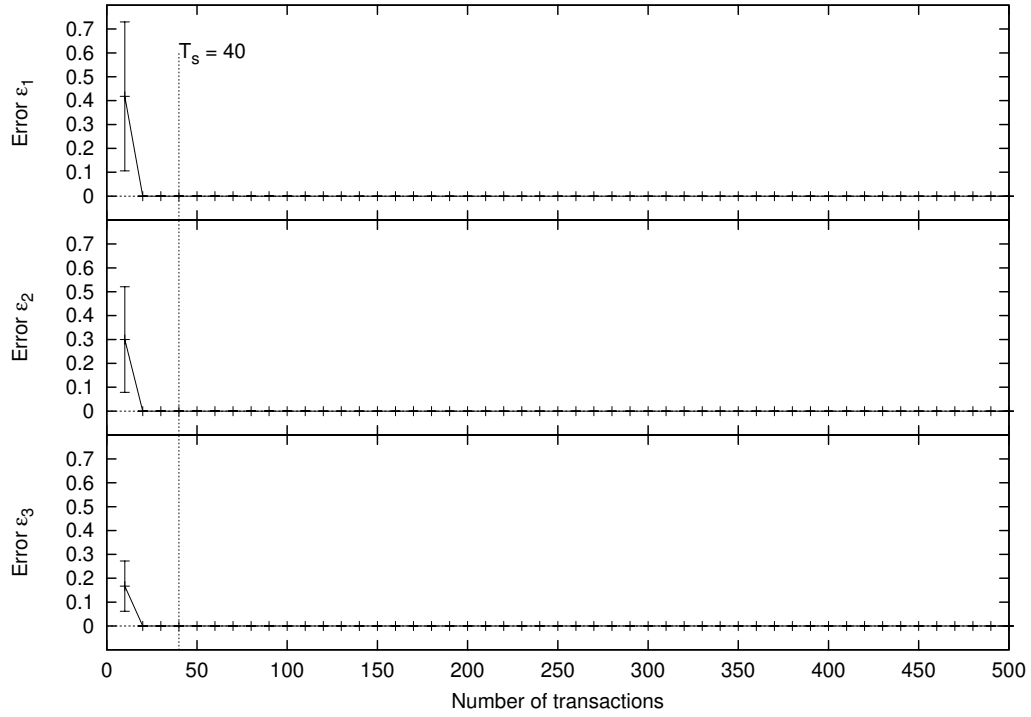


Figure B.18: Elevator: Uniform distribution, number of operators n=1..5, $T_s = 20$.

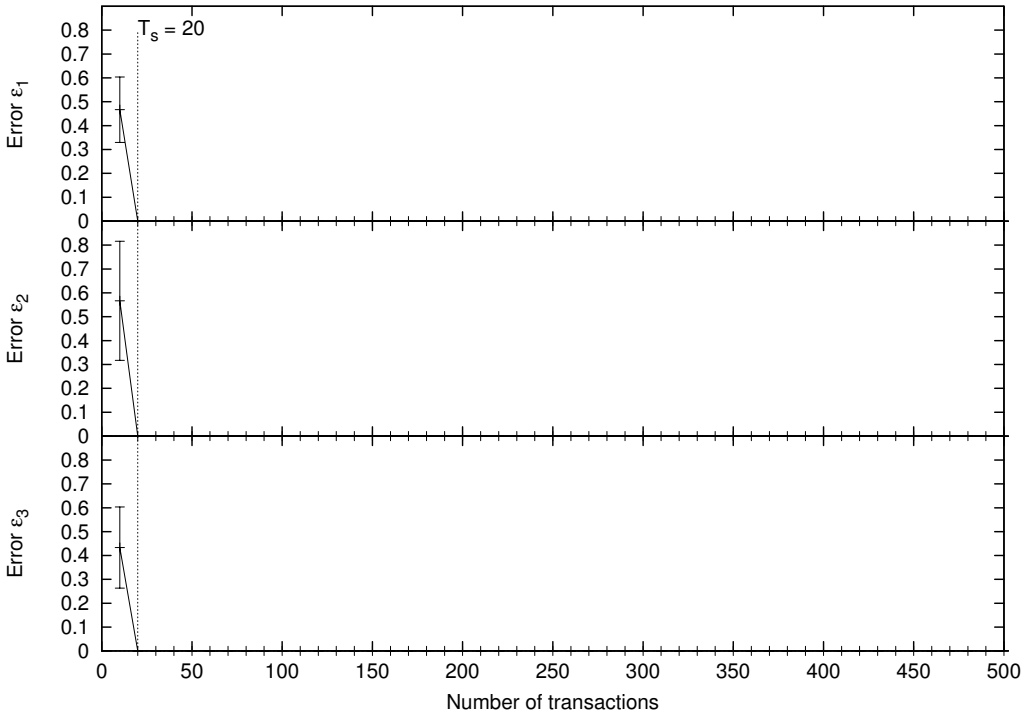


Figure B.19: Elevator: Non-uniform, number of operators n=4, $T_s = 70$.

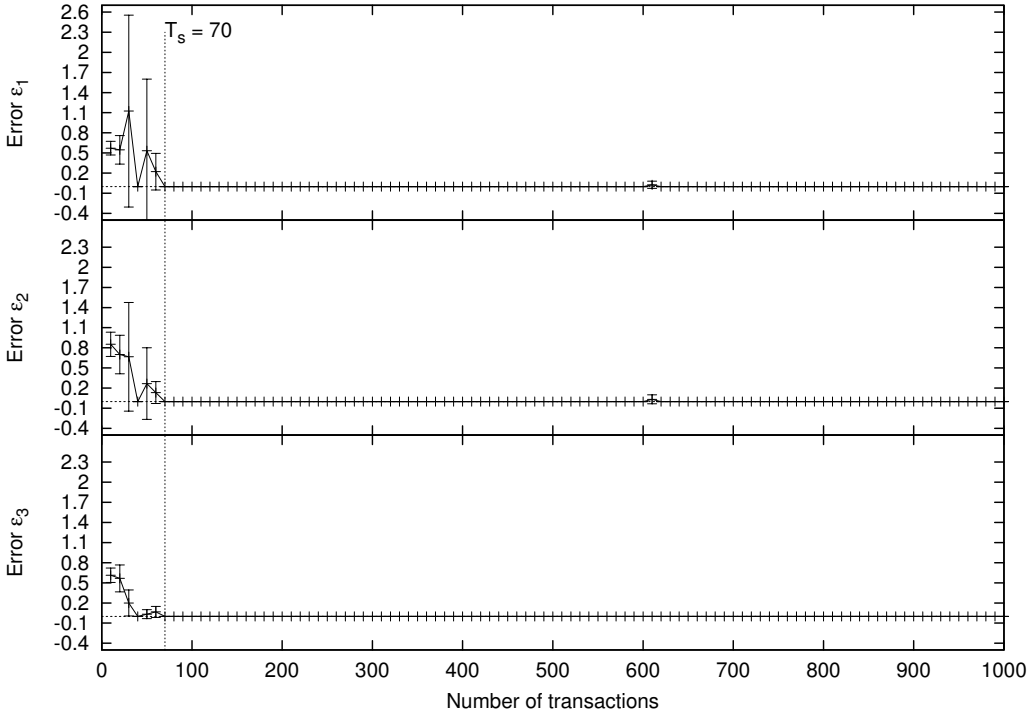
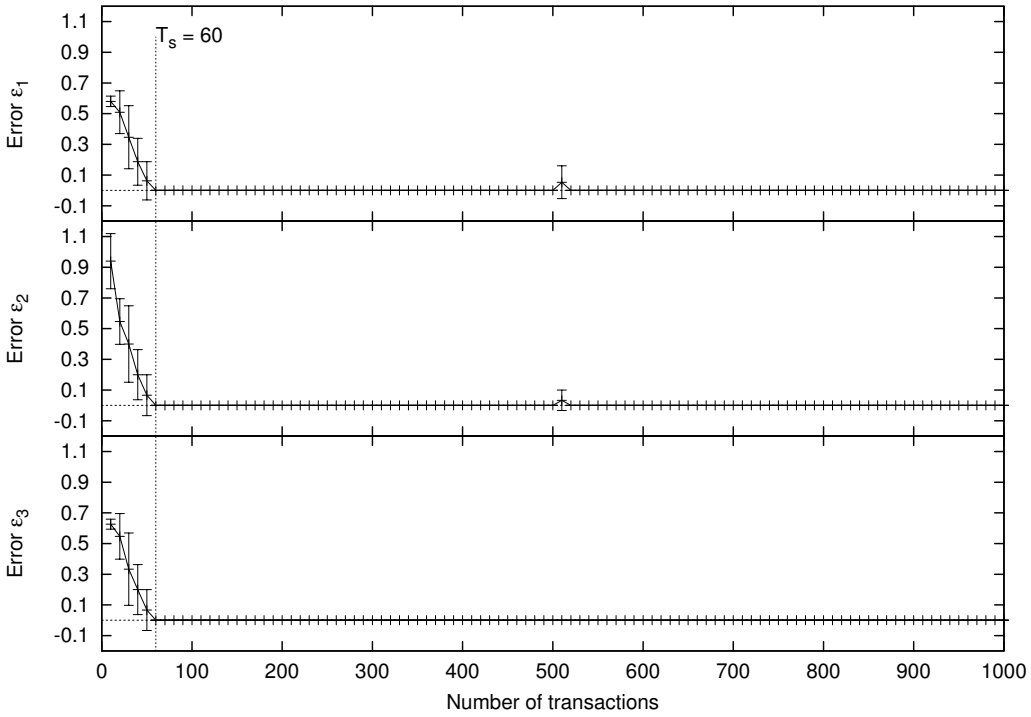


Figure B.20: Elevator: Non-uniform, number of operators n=1..5, $T_s = 60$.



Elevator with 10% Omission Noise

Figure B.21: Elevator: Uniform distribution, number of operators n=4, $T_s = 40$.

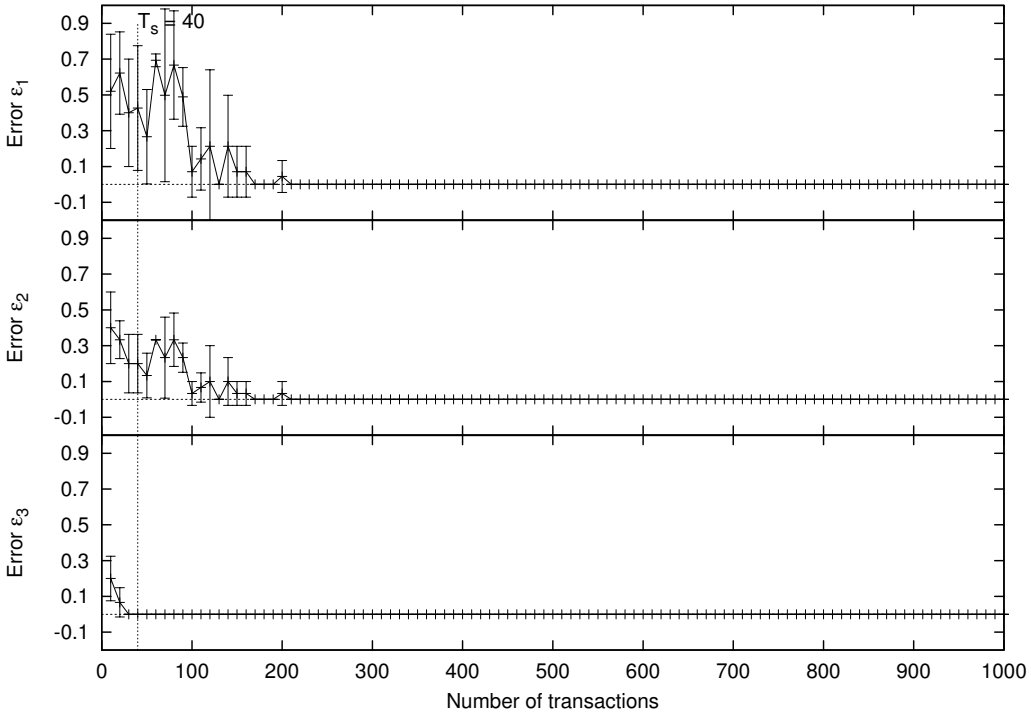


Figure B.22: Elevator: Uniform distribution, number of operators n=1..5, $T_s = 20$.

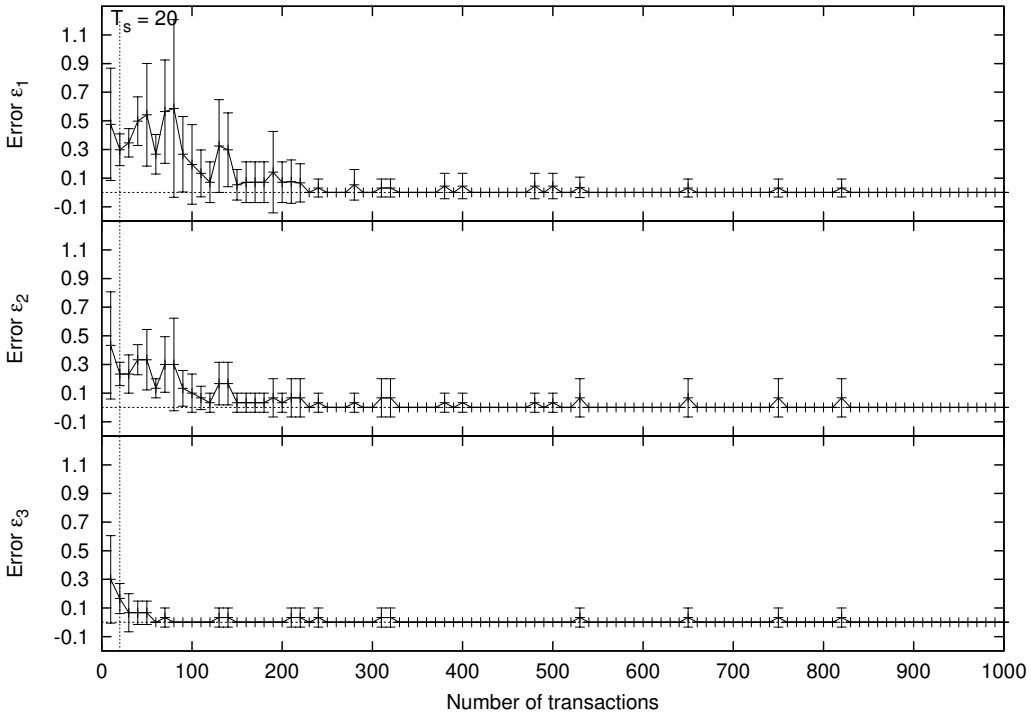


Figure B.23: Elevator: Non-uniform, number of operators n=4, $T_s = 70$.

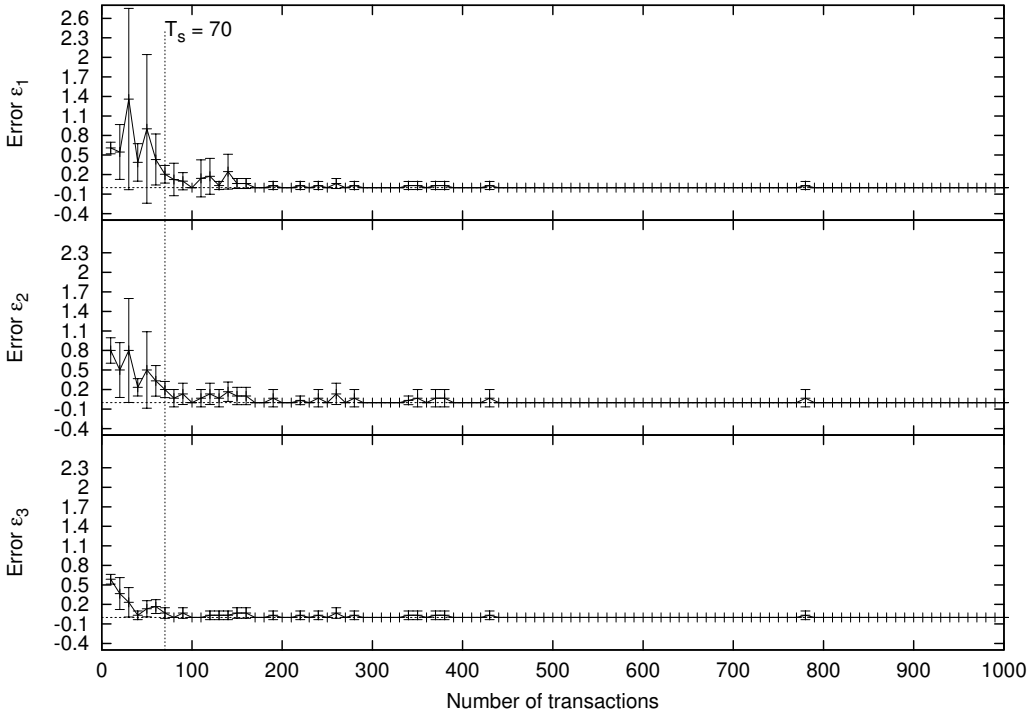
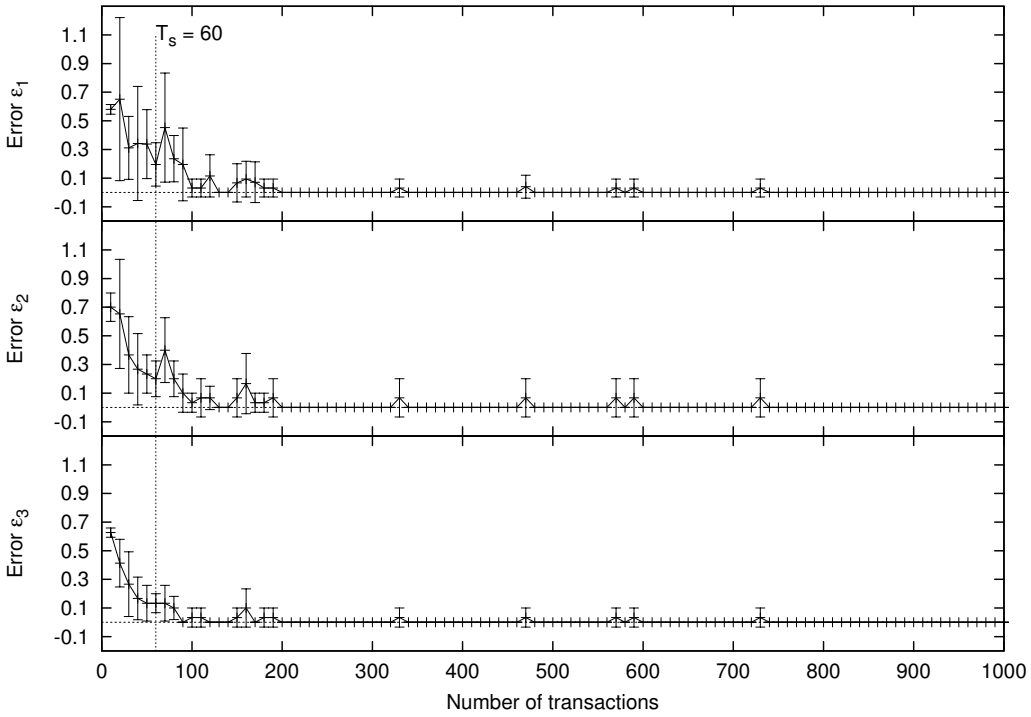


Figure B.24: Elevator: Non-uniform, number of operators n=1..5, $T_s = 60$.



Elevator with $5\% + 5\%$ Omission and Commission Noise

Figure B.25: Elevator: Uniform distribution, number of operators n=4, $T_s = 40$.

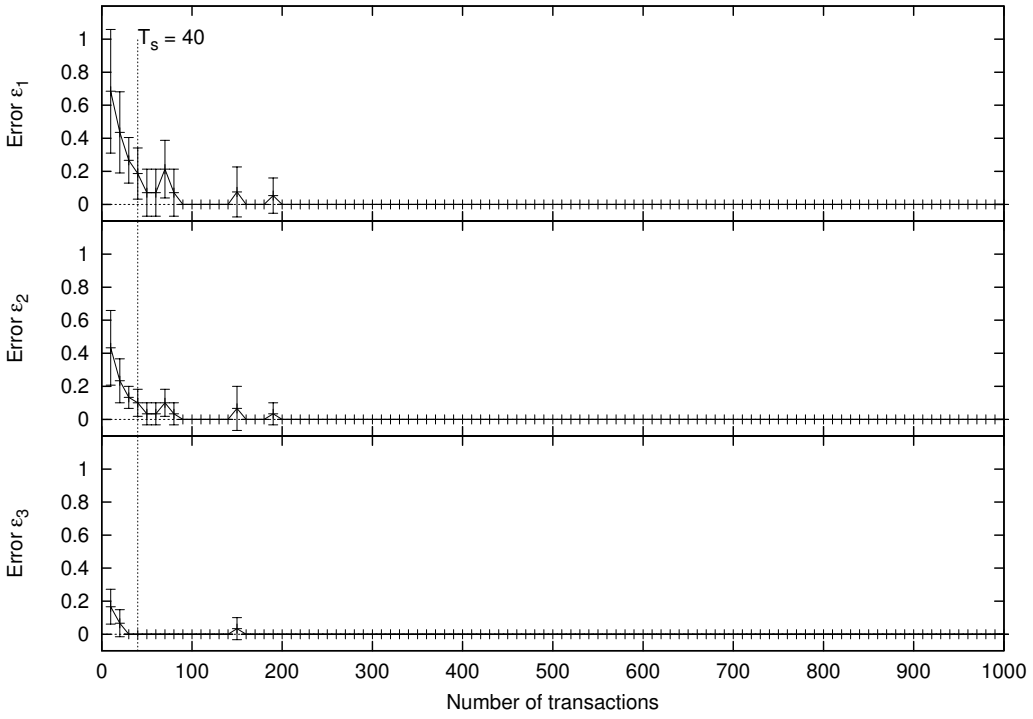


Figure B.26: Elevator: Uniform distribution, number of operators n=1..5, $T_s = 20$.

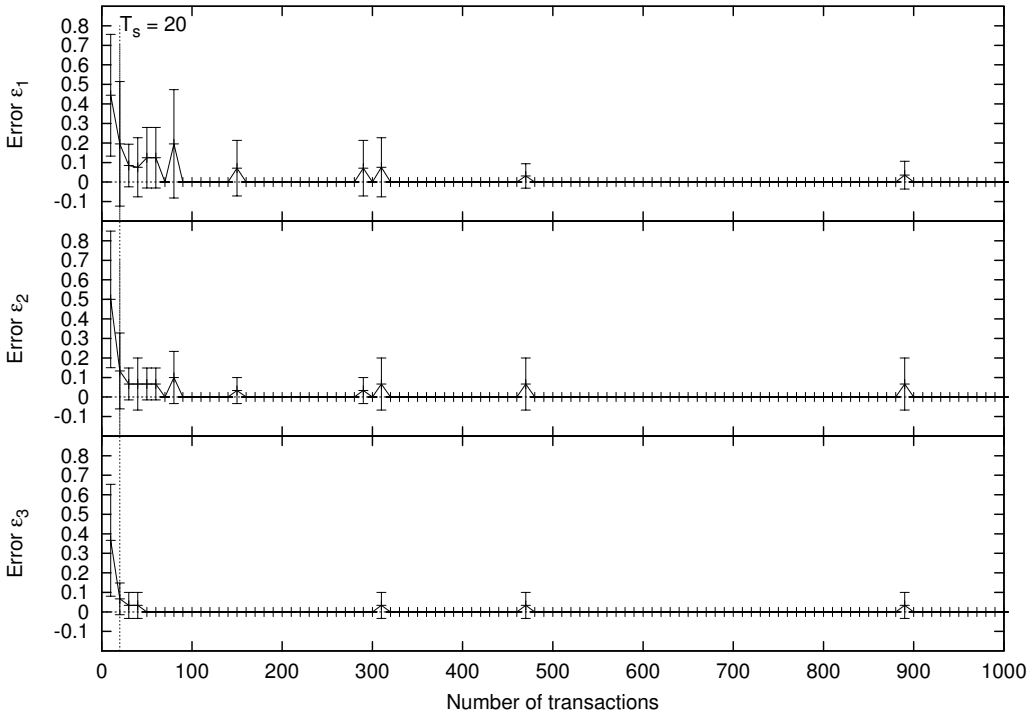


Figure B.27: Elevator: Non-uniform, number of operators n=4, $T_s = 70$.

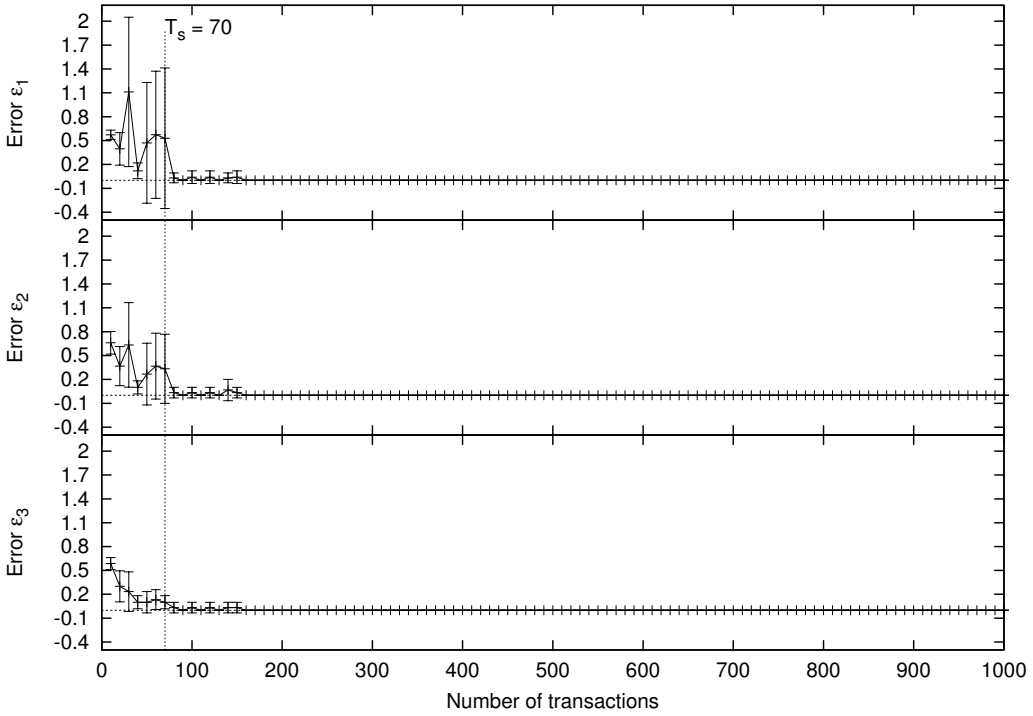
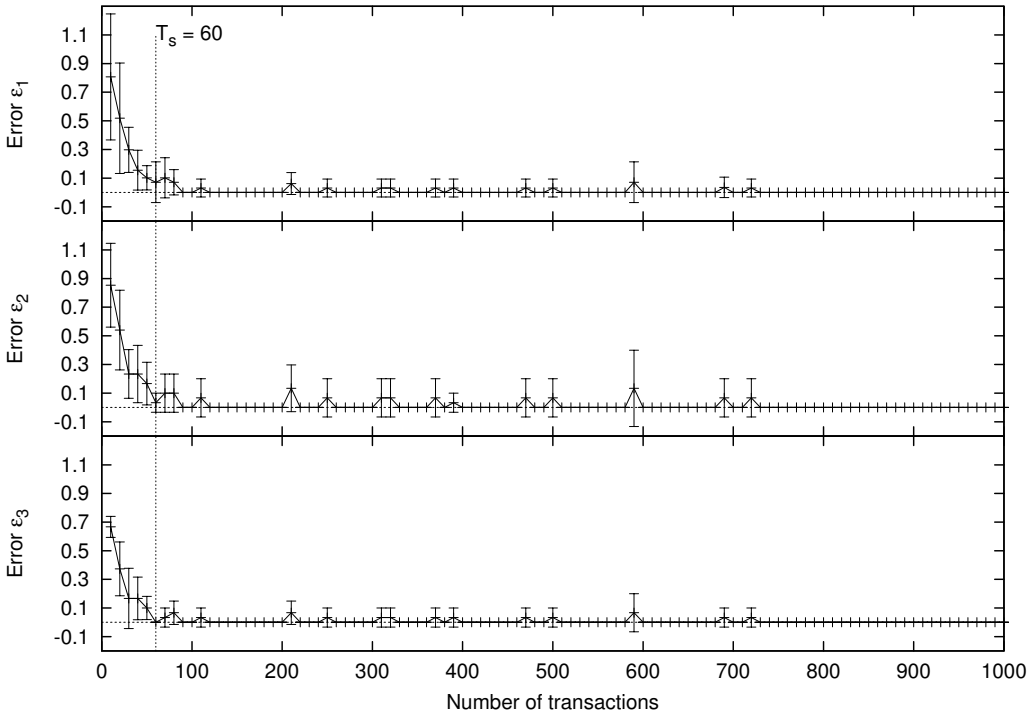


Figure B.28: Elevator: Non-uniform, number of operators n=1..5, $T_s = 60$.



B.2.2 Mars Rover

Mars Rover with 10% Commission Noise

Figure B.29: Mars Rover: Uniform distribution, number of operators n=4, $T_s = 50$.

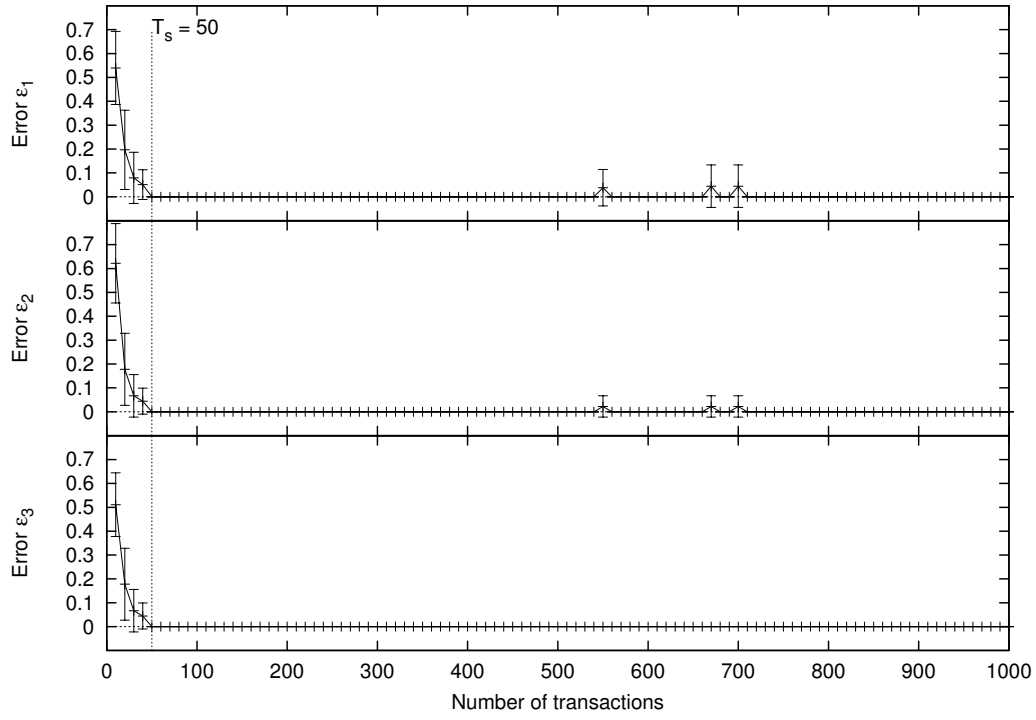


Figure B.30: Mars Rover: Uniform distribution, number of operators n=1..5, $T_s = 60$.

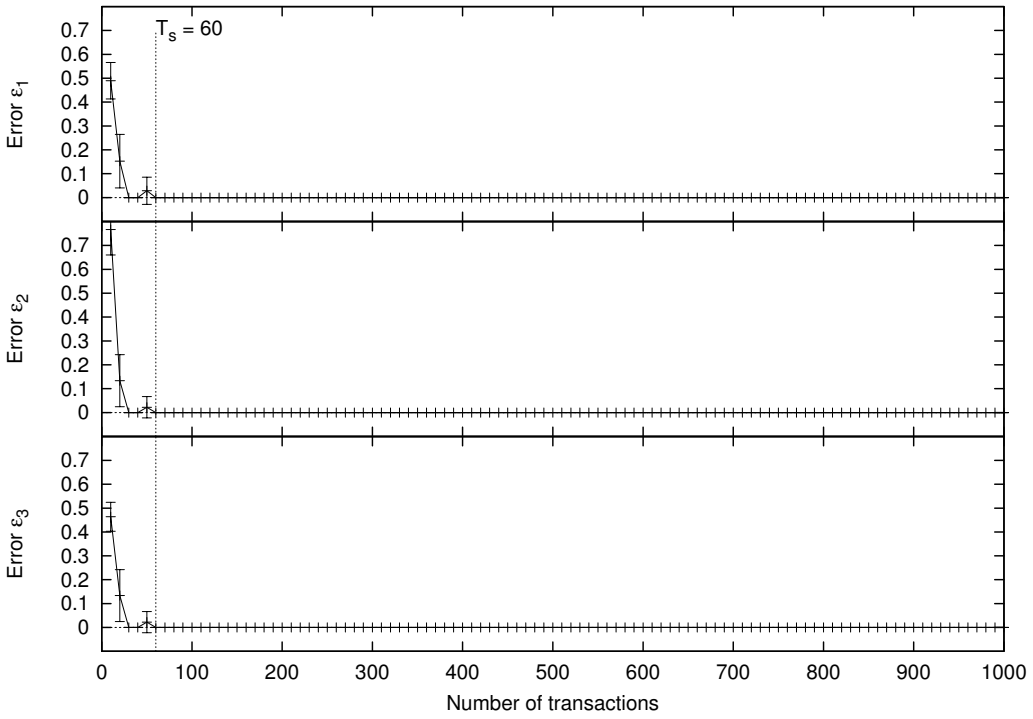


Figure B.31: Mars Rover: Non-uniform, number of operators n=4, $T_s = 340$.

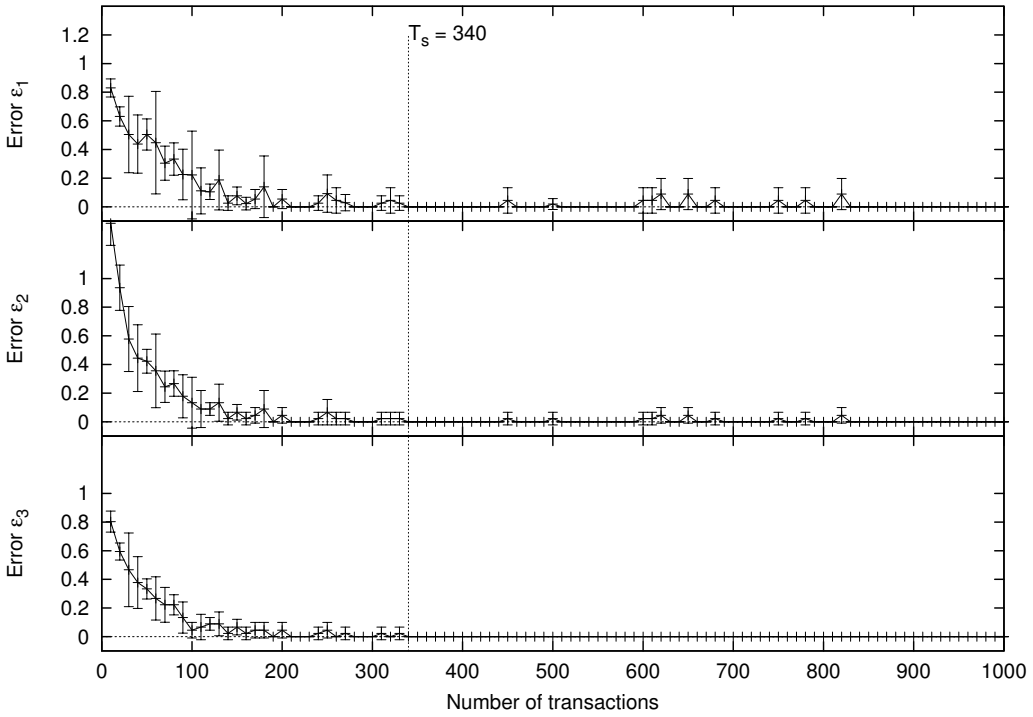
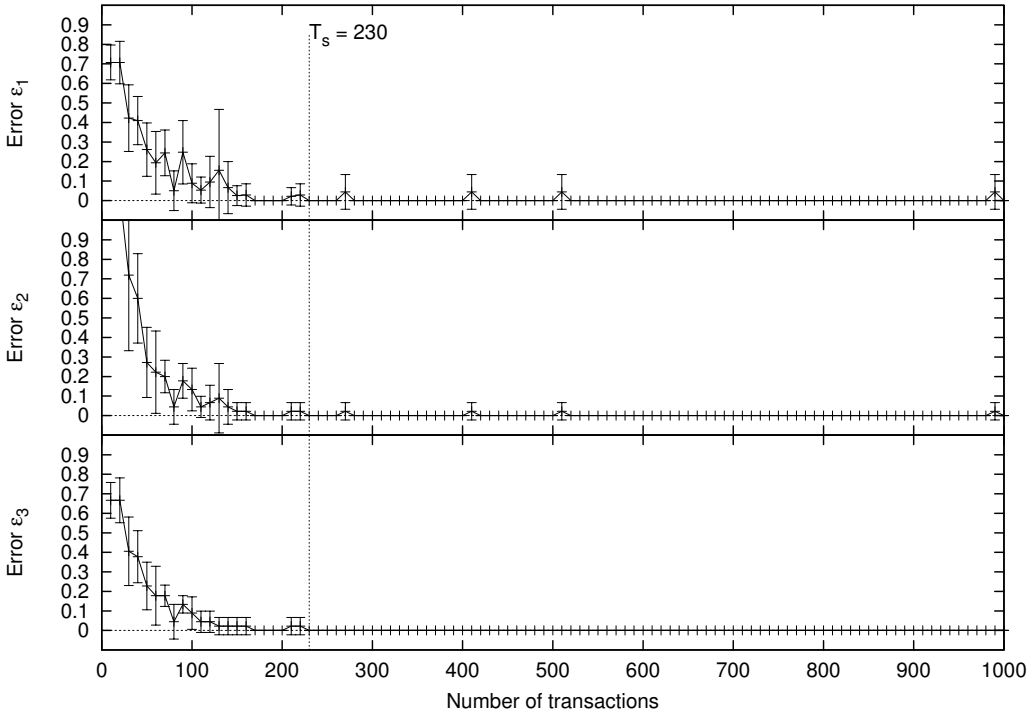


Figure B.32: Mars Rover: Non-uniform, number of operators n=1..5, $T_s = 230$.



Mars Rover with 10% Omission Noise

Figure B.33: Mars Rover: Uniform distribution, number of operators n=4, $T_s = 50$.

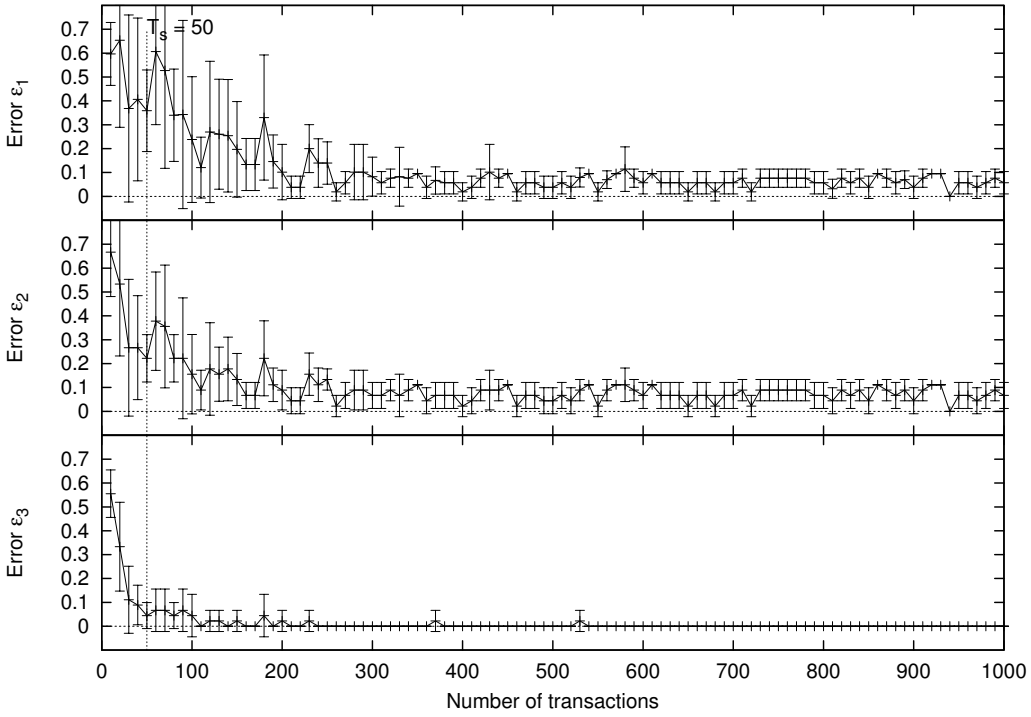


Figure B.34: Mars Rover: Uniform distribution, number of operators n=1..5, $T_s = 60$.

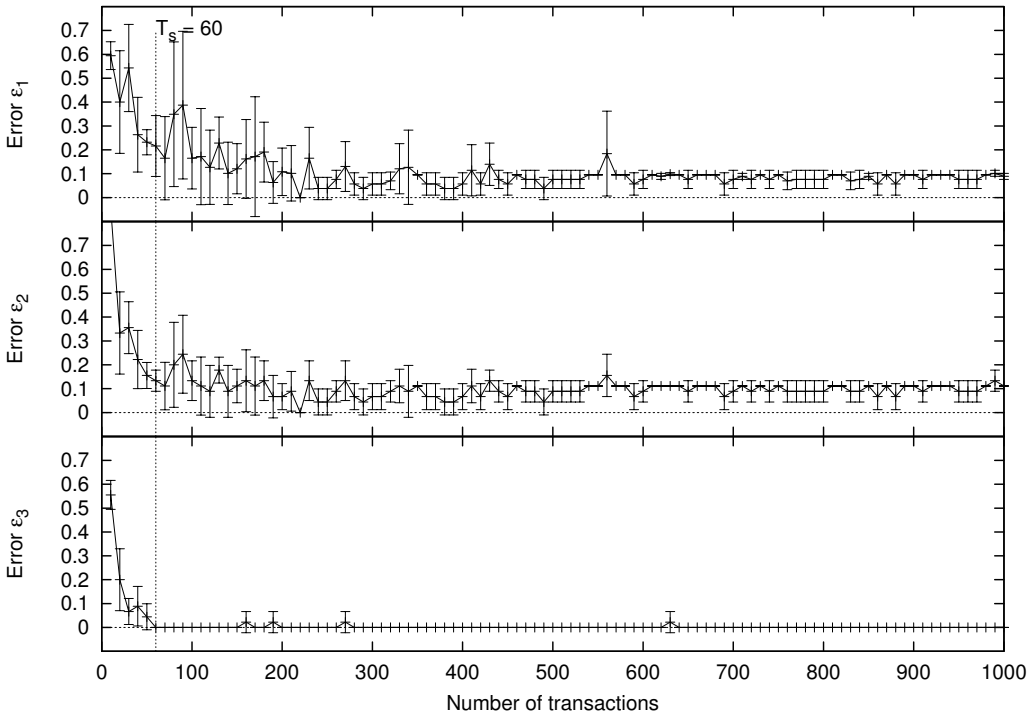


Figure B.35: Mars Rover: Non-uniform, number of operators n=4, $T_s = 340$.

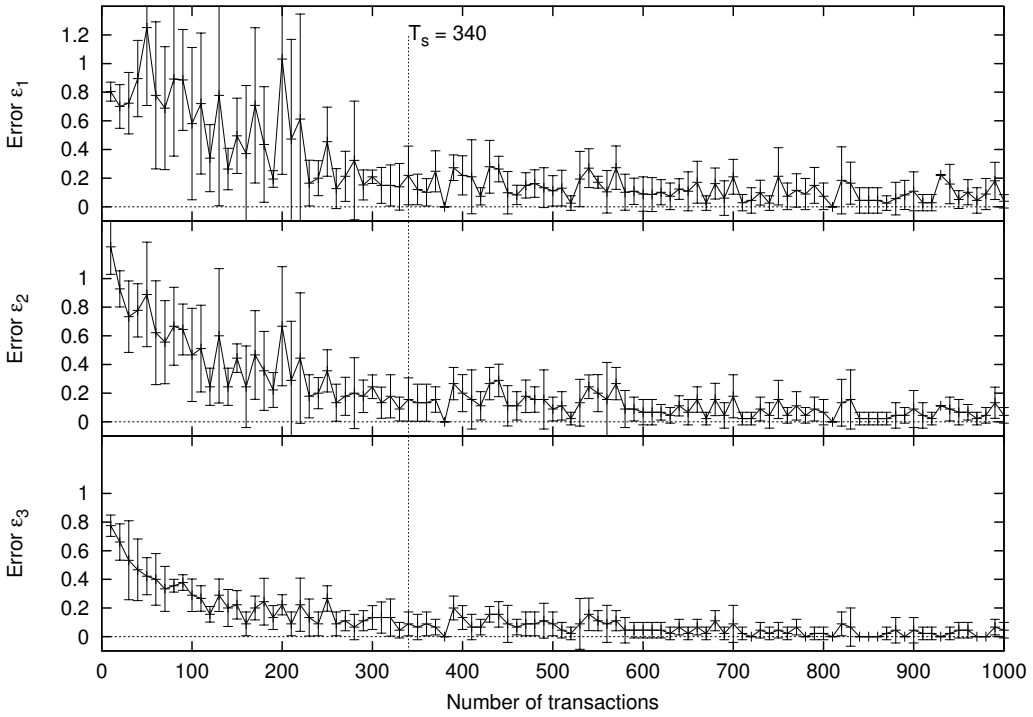
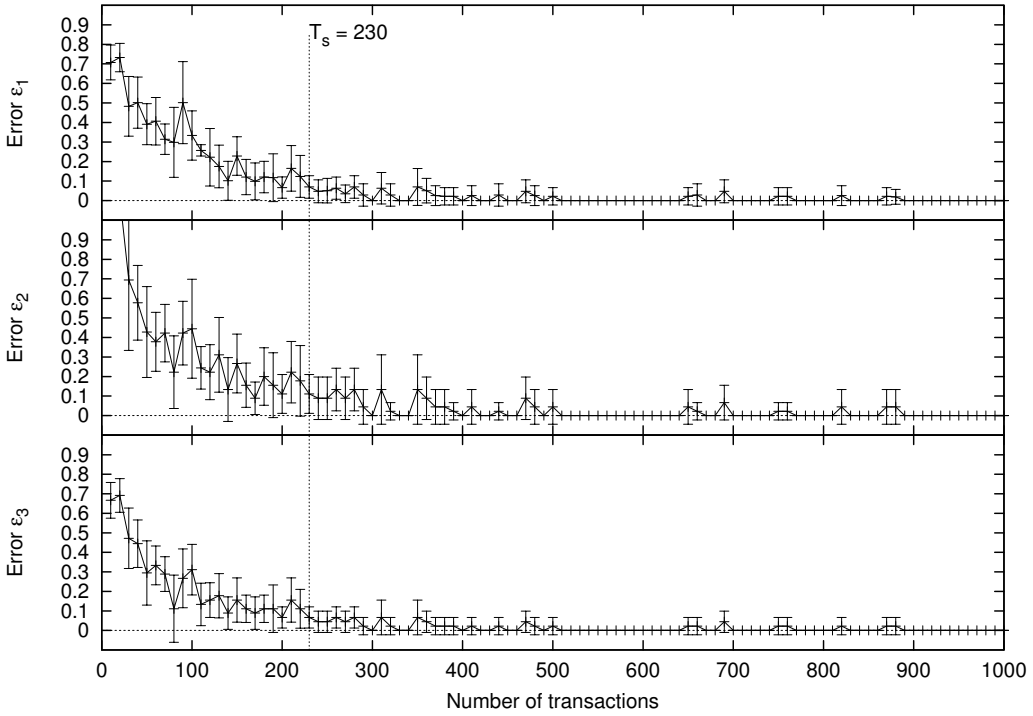


Figure B.36: Mars Rover: Non-uniform, number of operators n=1..5, $T_s = 230$.



Mars Rover with $5\% + 5\%$ Omission and Commission Noise

Figure B.37: Mars Rover: Uniform distribution, number of operators n=4, $T_s = 50$.

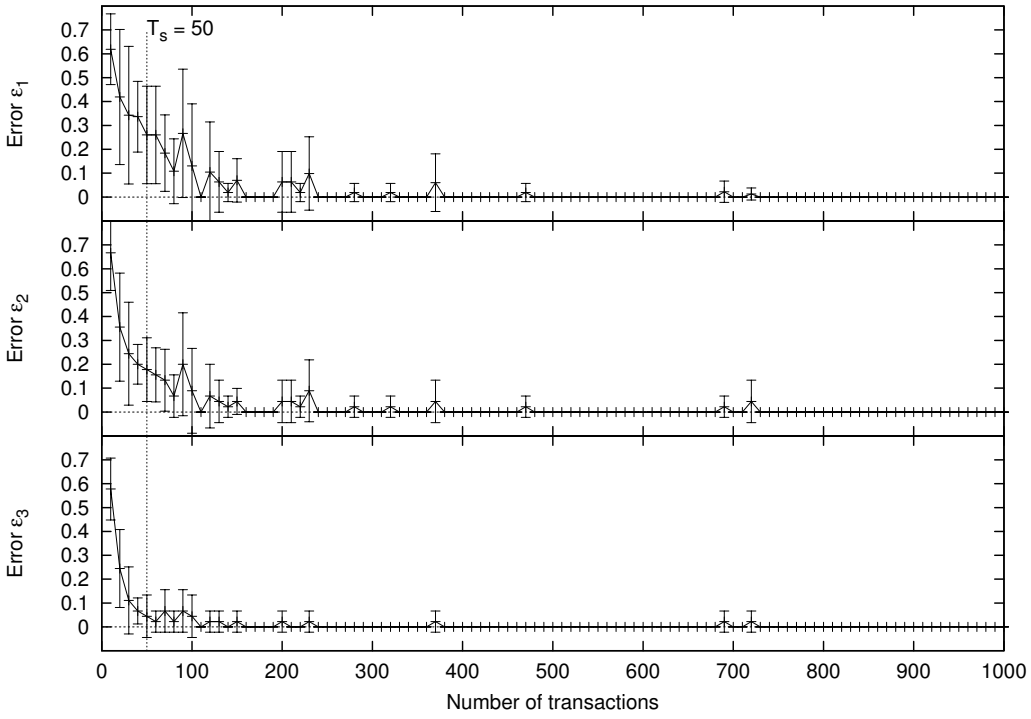


Figure B.38: Mars Rover: Uniform distribution, number of operators n=1..5, $T_s = 60$.

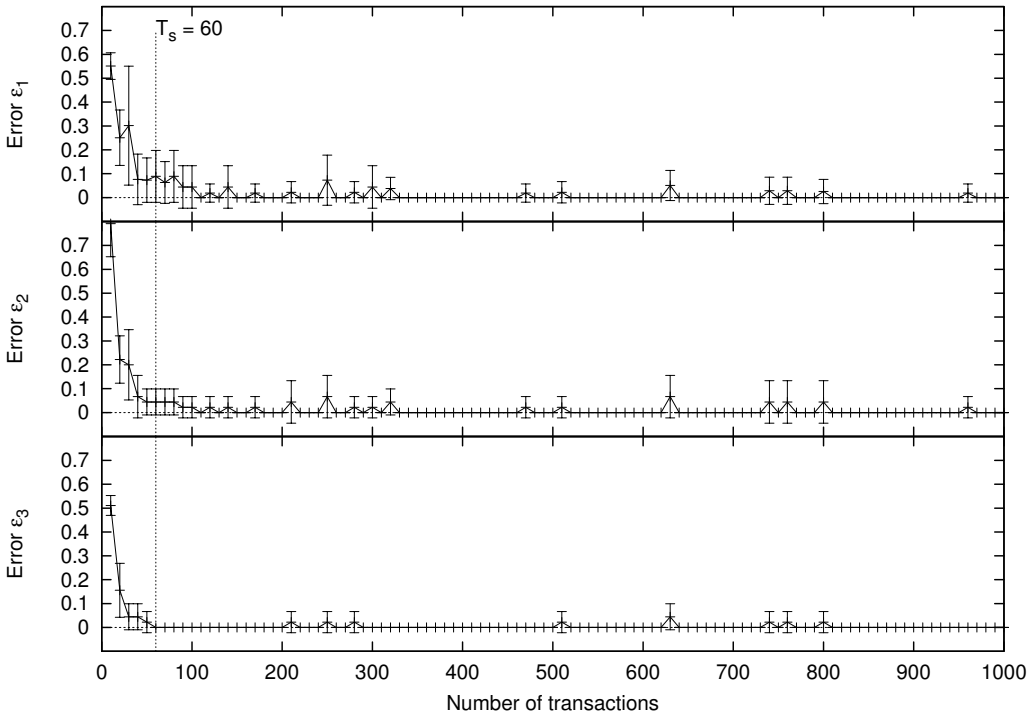


Figure B.39: Mars Rover: Non-uniform, number of operators n=4, $T_s = 340$.

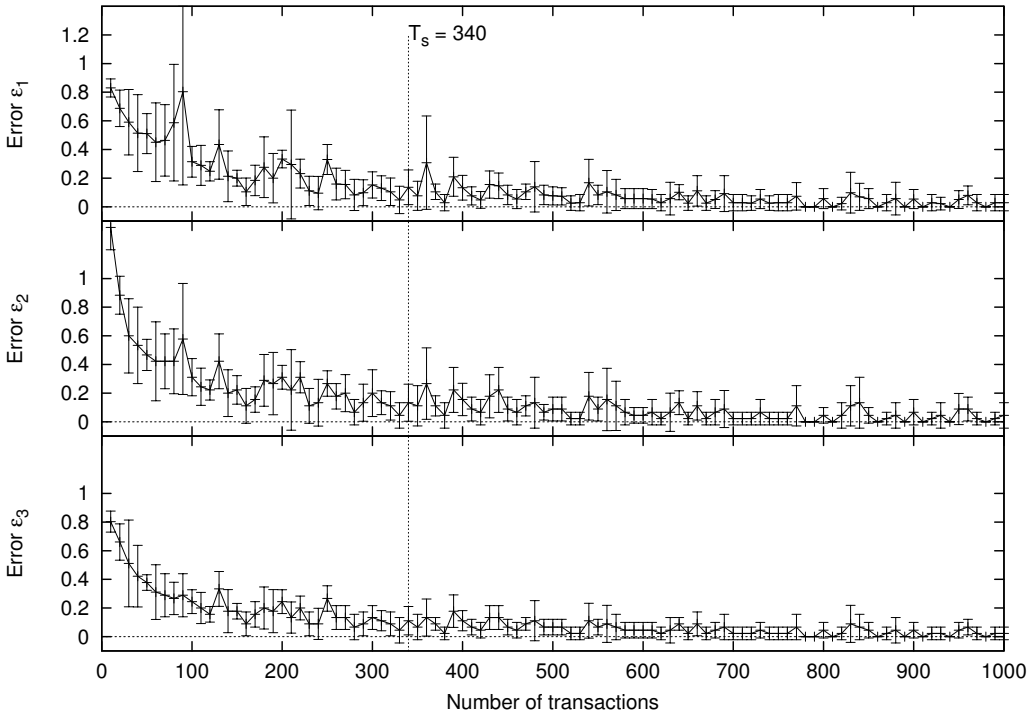
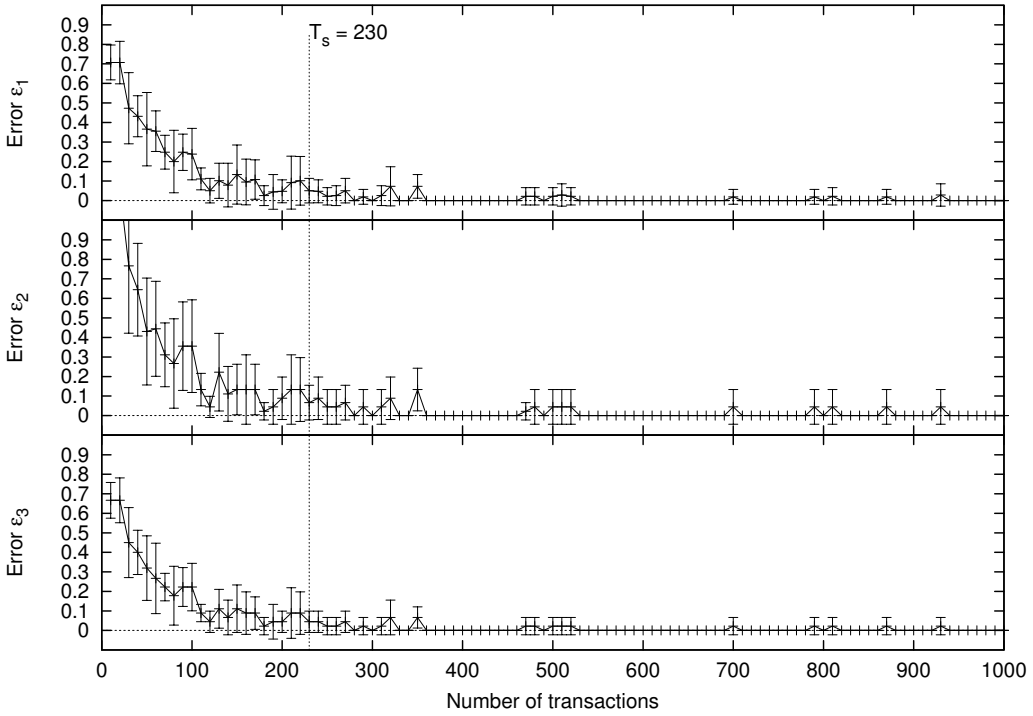


Figure B.40: Mars Rover: Non-uniform, number of operators n=1..5, $T_s = 230$.



B.2.3 Bridge inspection

Bridge Inspection with 10% Commission Noise

Figure B.41: Bridge Inspection: Uniform, number of operators n=4, $T_s = 220$.

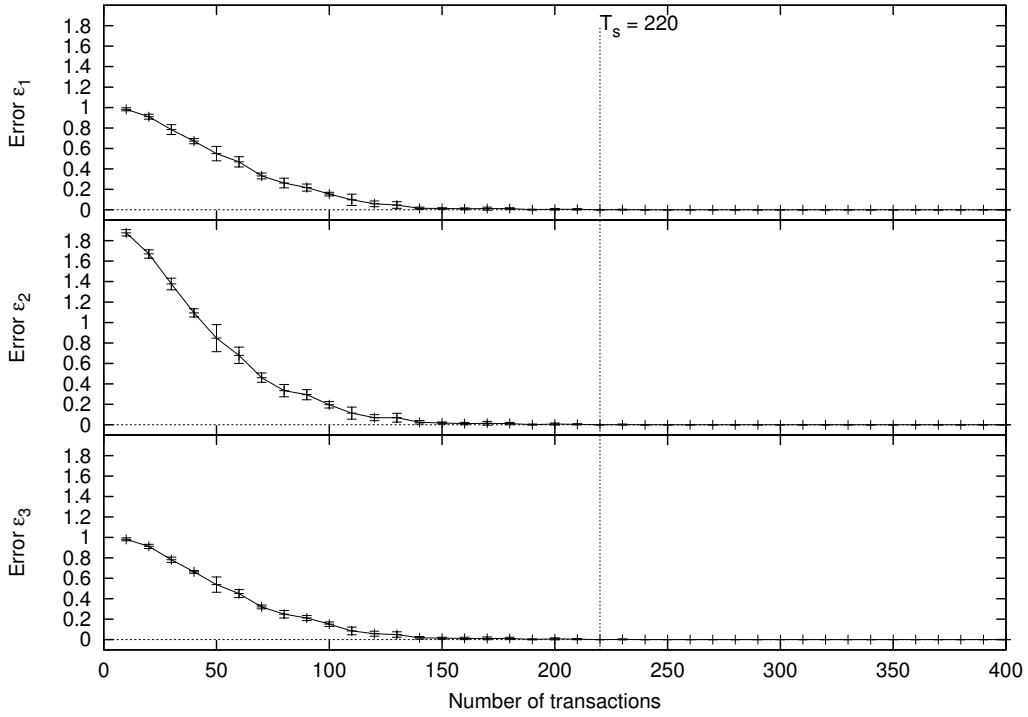


Figure B.42: Bridge Inspection: Uniform, number of operators n=1..5, $T_s = 290$.

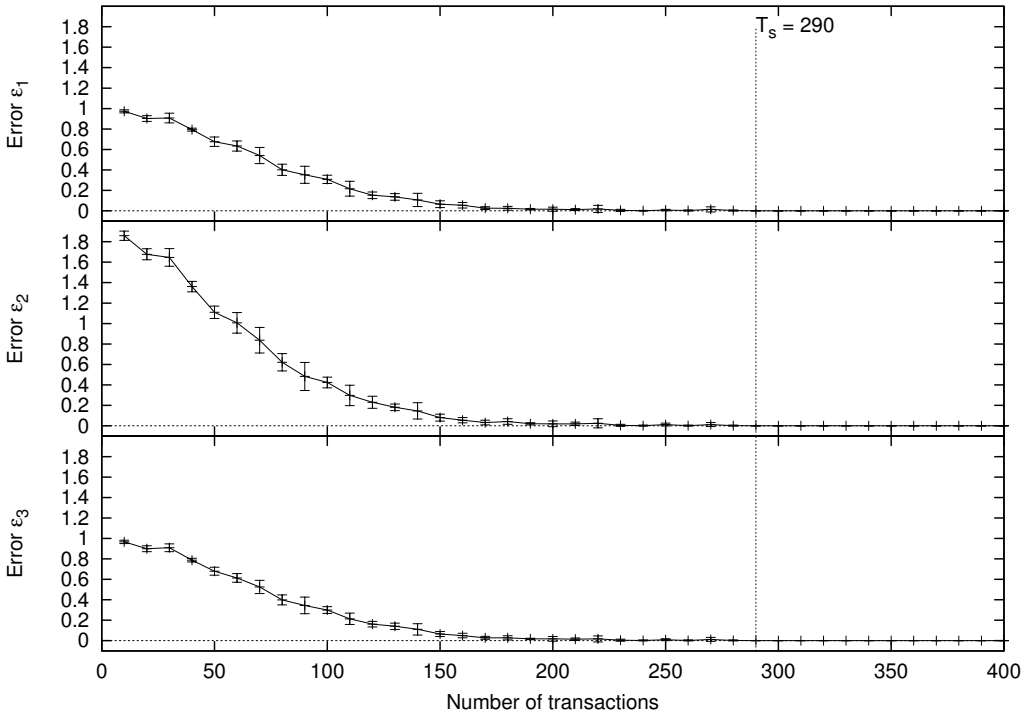


Figure B.43: Bridge Inspection: Non-uniform, number of operators n=4, $T_s = 650$.

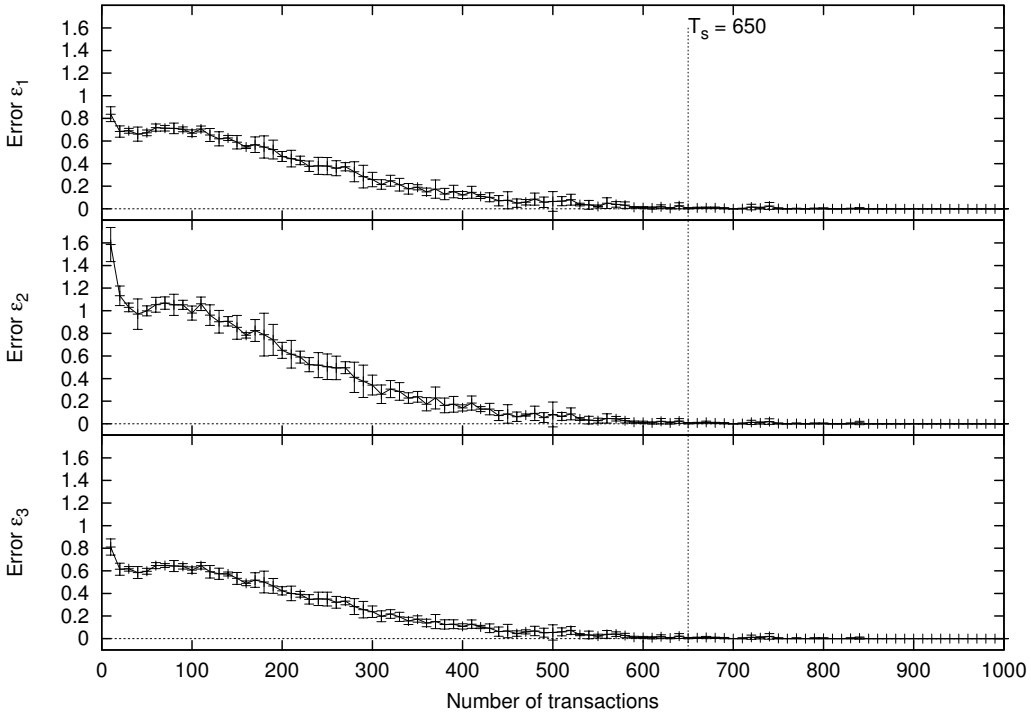
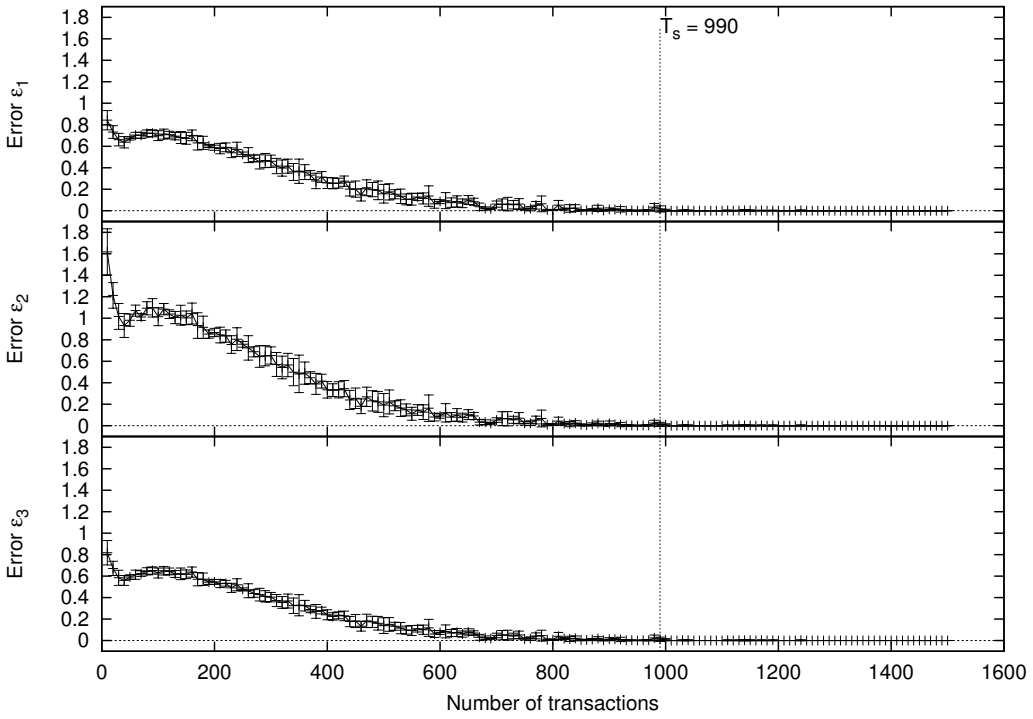


Figure B.44: Bridge Inspection: Non-uniform, number of operators n=1..5, T_s = 990.



Bridge Inspection with 10% Omission Noise

Figure B.45: Bridge Inspection: Uniform, number of operators n=4, $T_s = 220$.

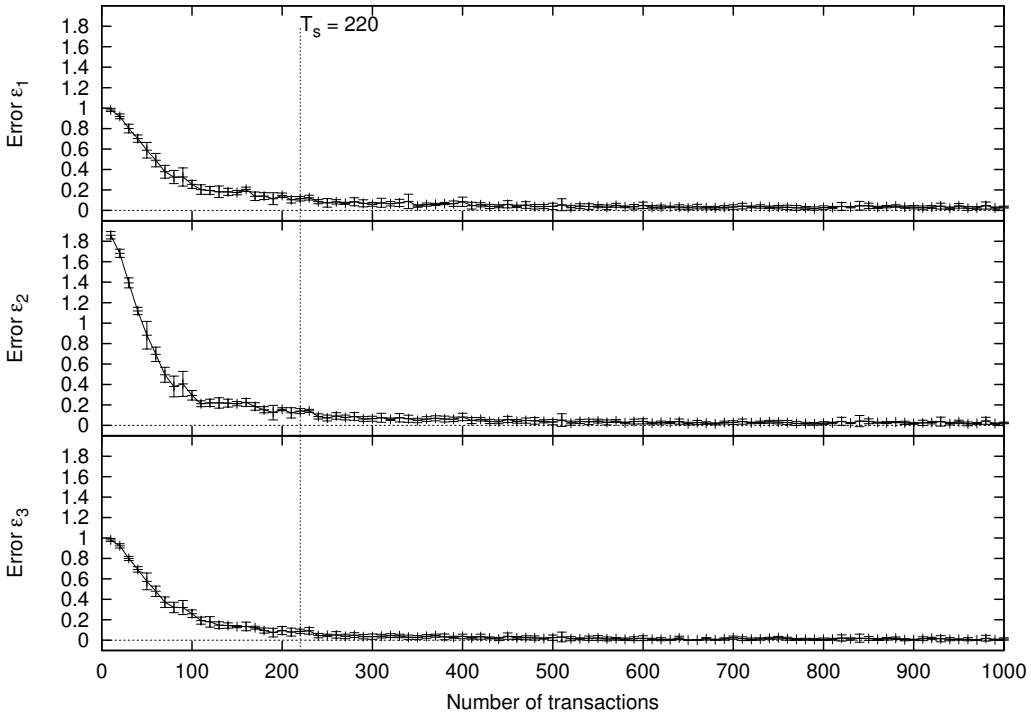


Figure B.46: Bridge Inspection: Uniform, number of operators n=1..5, $T_s = 290$.

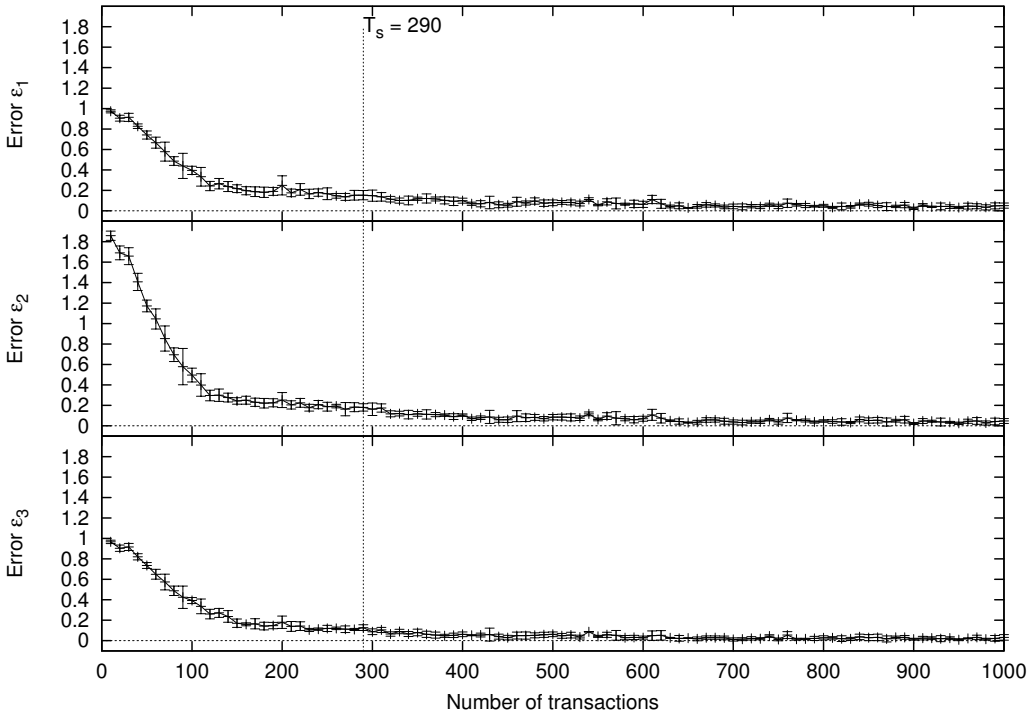


Figure B.47: Bridge Inspection: Non-uniform, number of operators n=4, $T_s = 650$.

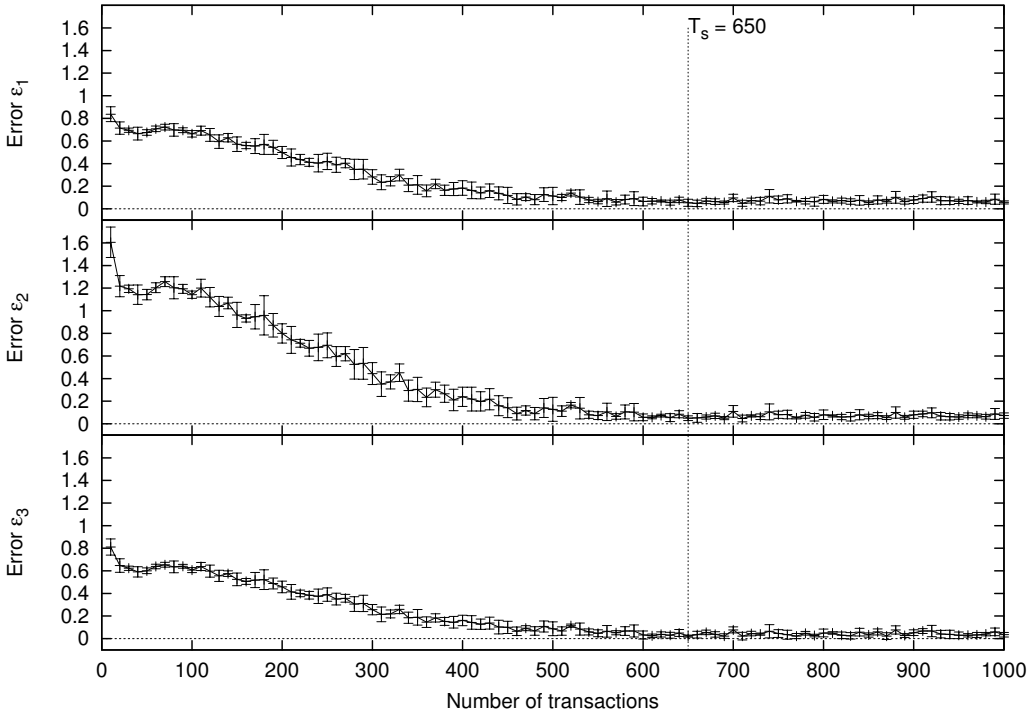
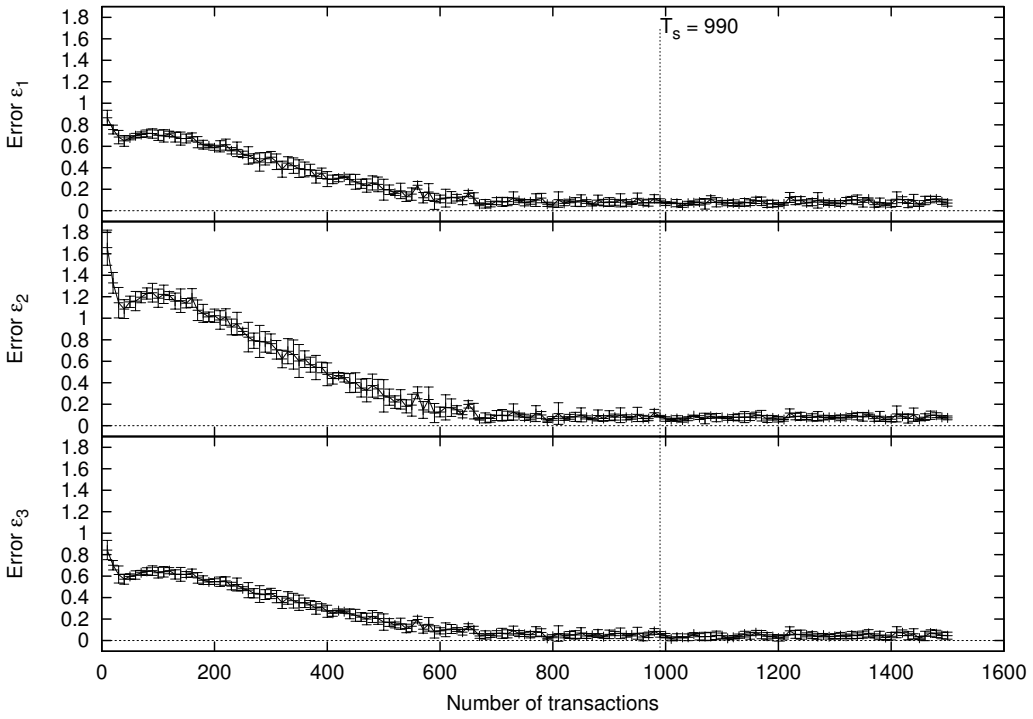


Figure B.48: Bridge Inspection: Non-uniform, number of operators n=1..5, T_s = 990.



Bridge Inspection with $5\% + 5\%$ Omission and Commission Noise

Figure B.49: Bridge Inspection: Uniform, number of operators n=4, $T_s = 220$.

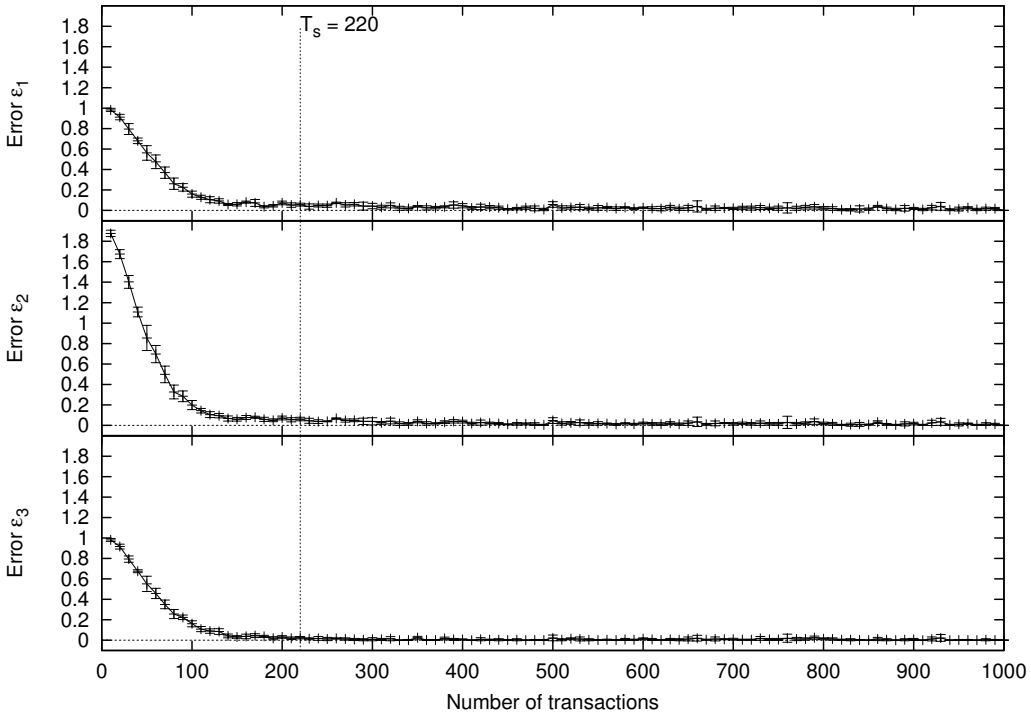


Figure B.50: Bridge Inspection: Uniform, number of operators n=1..5, $T_s = 290$.

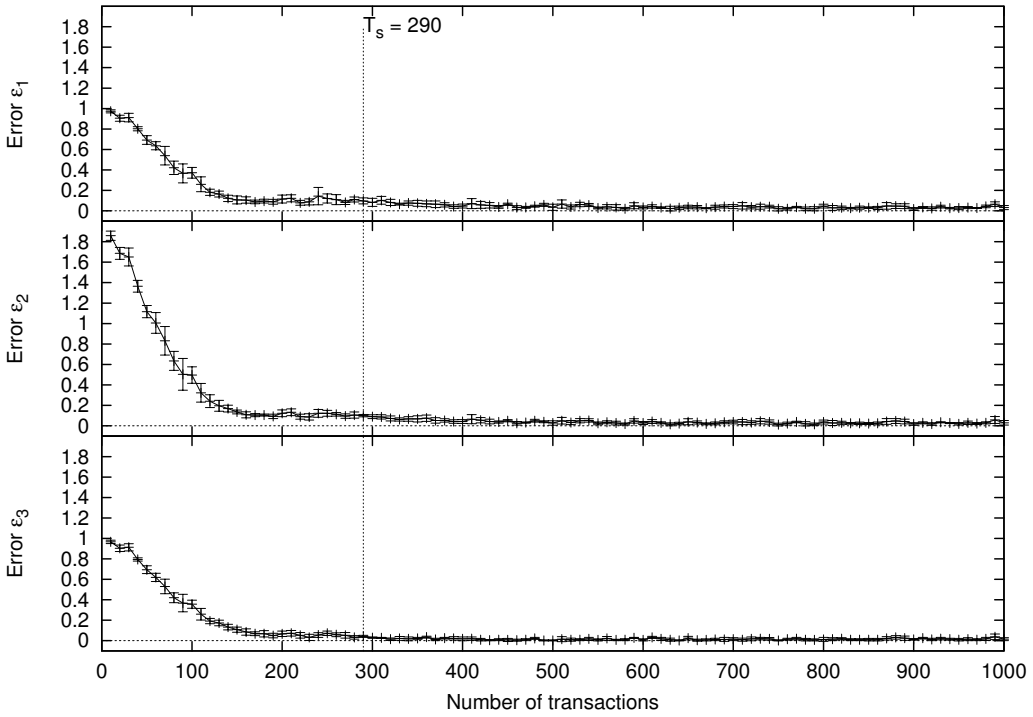


Figure B.51: Bridge Inspection: Non-uniform, number of operators n=4, $T_s = 650$.

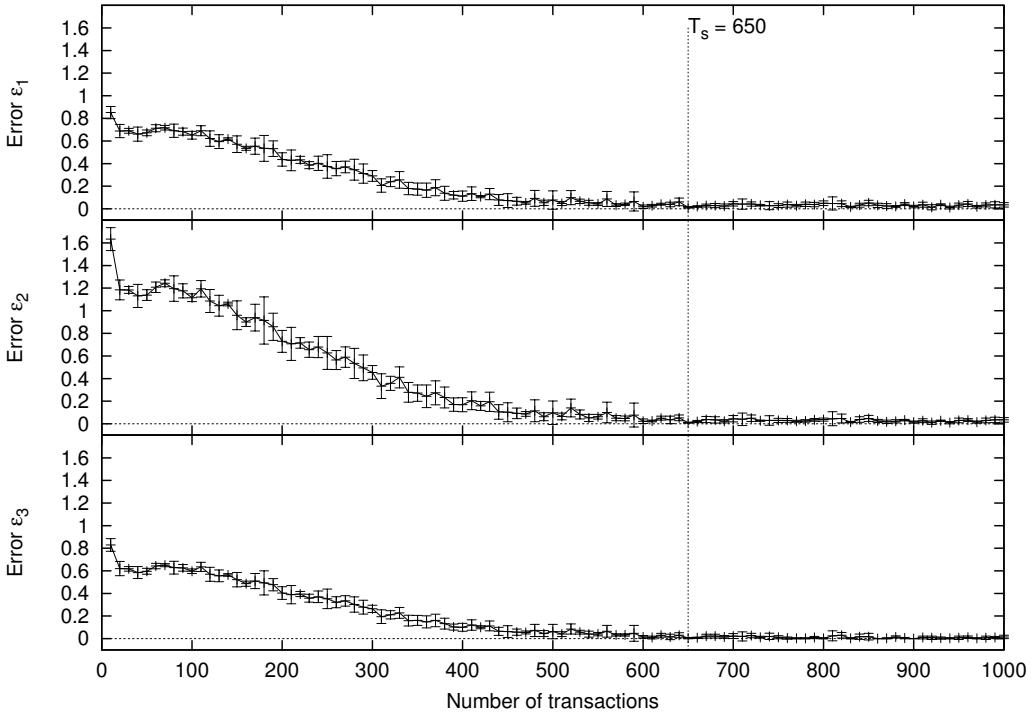


Figure B.52: Bridge Inspection: Non-uniform, number of operators n=1..5, T_s = 990.