Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Structural characterization of planning problems

por

José Anastacio Hernández Saldaña

como requisito parcial para obtener el grado de

maestría en ciencias en ingeniería de sistemas

Marzo 2019

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección de Estudios de Posgrado

Structural characterization of planning problems

por

José Anastacio Hernández Saldaña

como requisito parcial para obtener el grado de

maestría en ciencias en ingeniería de sistemas

Marzo 2019

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección de Estudios de Posgrado

Los miembros del Comité de Tesis recomendamos que la Tesis ≪Structural characterization of planning problems≫, realizada por el alumno José Anastacio Hernández Saldaña, con número de matrícula 1186622, sea aceptada para su defensa como requisito parcial para obtener el grado de Maestría en Ciencias en Ingeniería de Sistemas.

El Comité de Tesis

Dr. Romeo Sánchez Nigenda
Asesor

Dra. Satu Elisa Schaeffer
Asesora

Dr. Hugo Jair Escalante Balderas
Revisor

Vo. Bo.

Dr. Simón Martínez Martínez
Subdirector de Estudios de Posgrado

San Nicolás de los Garza, Nuevo León, marzo 2019

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección de Estudios de Posgrado

Los miembros del Comité de Tesis recomendamos que la Tesis «Structural characterization of planning problems», realizada por el alumno José Anastacio Hernández Saldaña, con número de matrícula 1186622, sea aceptada para su defensa como requisito parcial para obtener el grado de Maestría en Ciencias en Ingeniería de Sistemas.

El Comité de Tesis

[illegible]

Dr. Romeo Sánchez Nigenda
Asesor

[illegible]

Dra. Satu Elisa Schaeffer
Asesora

[illegible]

Dr. Hugo Jair Escalante Balderas
Revisor

Vo. Bo.

Dr. Simón Martínez Martínez
Subdirector de Estudios de Posgrado

San Nicolás de los Garza, Nuevo León, enero 2019

Para todos aquellos que se aventuran a saber. Sapere Aude.

Contents

List of Figures

List of Tables

Acknowledgments

Deseo agradecer a la Universidad Autónoma de Nuevo León (UANL) la oportunidad que me brindó para realizar mis estudios de posgrado.

A la Facultad de Ingeniería Mecánica y Eléctrica (FIME) por el apoyo brindado durante mis estudios de maestría.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico otorgado mediante una beca de estudios de tiempo completo.

Quedo agradecido al Posgrado en Ingeniería de Sistemas (PISIS) por darme la oportunidad de realizar mis estudios de maestría.

Al Dr. Romeo, por dirigir esta tesis, por guiarme en esta área de investigación totalmente nueva para mi y por apoyarme en la elaboración de este trabajo.

A la Dra. Elisa por ayudarme a dar lo mejor de mi, por su excelente labor pedagógica, por su paciencia y su calidez humana al guiarme, por transmitirme su dedicación por la labor científica como nunca la habia imaginado.

Al Dr. Jair por sus sugerencias, comentarios y apoyo que permitieron mejorar este trabajo.

A los profesores del Posgrado de Ingenieria de Sistemas, por compartir su conocimiento y por su labor educativa. A mis compañeros de generación y del laboratorio de Yalma por su amistad en estos años de aprendizaje, por sus consejos y los momentos felices que pasamos juntos.

A mis familia y amigos que siempre me apoyaron y estuvieron pendientes de mi. A mis amigos de FCFM por alentarme a seguir preparandome.

A Cathy por acompañarme en esta etapa de mi vida, tu paciencia y apoyo fueron indispensables para mi.

Resumen

José Anastacio Hernández Saldaña.

Candidato para obtener el grado de Maestría en Ciencias en Ingeniería de Sistemas.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio: Structural characterization of planning problems.

Número de páginas: 87.

Objetivos y método de estudio: El objetivo de esta tesis es identificar propiedades estructurales en instancias de problemas clásicos de planificación que permitan caracterizar su dificultad. Los problemas de planificación son difíciles de resolver ya que están dentro de la clase complejidad PSpace. Al identificar propiedades en las instancias de planificación analizando el grafo que representa a cada instancia, se facilita la solución de instancias con características similares. El presente trabajo sigue el método de selección de algoritmo para analizar las características de las instancias y descubrir cuáles afectan el desempeño de los planificadores.

Contribuciones y conclusiones: Este trabajo propone una metodología para clasificar las instancias basándose en el desempeño mostrado por los planificadores al resolverlas, esta metodología necesitó del desarrollo de herramientas para extraer, almacenar y analizar los grafos usados para estudiar las instancias. Este estudio aprovecha el uso de base de datos y análisis estadístico para identificar propiedades en las instancias estudiados. Además se identificaron dos propiedades que están presentes con diferentes valores en instancias con y sin solución.

Firmas de los asesores:

Dr. Romeo Sánchez Nigenda

Dra. Satu Elisa Schaeffer

Summary

José Anastacio Hernández Saldaña.

Candidate to the degree of Master of Science on Systems Engineering.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Thesis: Structural characterization of planning problems.

Number of pages: 87.

Objectives and study method: The objective of this thesis is identify structural properties in the instances of classical planning problems to characterize their difficulty. The planning problems are difficult to solve because they are in the complexity class PSpace. Identifying the properties in the planning instances by an analysis of its planning graph facilitates the solution of instances with similar characteristics. This present work follows the algorithm selection methodology to analyze which features impact planner performance.

Contribution and conclusions: The present work propose a methodology to characterize the instances based on the performance shown by planners. The thesis documents the development of tools to extract, store, and analyze the graphs used to study the instances. Finally, we take advantage of data base and statistical analysis to identify structural properties in the studied instances. Two properties were identified that are present with different values in solved and unsolved instances.

Adviser signatures:

Romeo Sánchez Nigenda, PhD

Satu Elisa Schaeffer, D. Sc.

Chapter 1

Introduction

Artificial Intelligence (AI) has gained major interest in the past decade, largely due to the advances in machine learning [37] with algorithms that learn from data to make predictions or take decisions to improve system performance, response time, recognize patterns, or solution quality. The advances on AI create implementation opportunities in mathematics, computer science, robotics, decision theory, among others, where the labor market, economy, and other aspects of life are affected [37]. The use of machine learning techniques on the data generated in the past decades made AI attractive, because it is possible to make decisions with a limited supervision or gain new insight of the data.

Artificial intelligence focuses on studying how to emulate rational thinking, whilst *automated planning*, as a branch of AI, studies how to emulate the ration thinking that evaluates the possible actions in a environment and decide which decisions of that actions achieves a defined goal. A main objective for AI is to create autonomous systems [32] and to achieve it, automated planning is a keystone in the AI future research.

Automated planning uses a description of the environment where it is going to plan. It is based on a model where the *initial conditions* are the description of that model including the objects in it, the set of available *actions* where each one requires a set of conditions to be fulfilled before it can be carried out, and a *goal*, which captures the conditions to be fulfilled by the sequence of actions. A *planner* is an algorithm that decides which series of actions accomplish the goal; the latter is referred to as the *plan* which is executed from the initial state in a specific order to achieve the goal.

To illustrate the automated planning problem a well-known problem called *blocks world* is discussed. In this problem there is a set of labeled blocks on a table where the goal is to build vertical stacks of blocks. The initial conditions are the definitions of the blocks: they could be above the table or above another block and each block could have its top free or with another block on it. Another initial condition is a hand that could be free or holding a block. The set of available actions are: pick-up of a block if the hand is free and the block has its top free, put-down of a block that is holding in the hand and put it over the table or a block. The goal is the position of the blocks on the stacks over the table. The plan is the followed sequence of pick-ups and put-downs from the initial conditions until the goal is reached.

One of the advantages of automated planning is its autonomy: when supervision is difficult or not intended, when the previous information is not available, or when the environment conditions, sets of actions, and goals change each time a plan is requested [6]. Despite of automated planning is a difficult problem to solve, in the past decades the efficiency of finding plans has increased and also the study of the difficulty of the planning problems.

There are different ways to model an automated planning problem based on certain assumptions about the environment. The one studied in this work is the *classical planning model* which was one of the first to be studied by its simplicity, and it was used to define some important problems such as: finding the existence of a plan in a given model it is known as the *Plan Existence* problem. Other problems studied by automated planning are the *Plan Length* problem, where a plan of at most steps than a limit k is determined or the *Minimal Plan Length* problem where a plan with the minimal amount of steps is determined [10]. The algorithms used to solve these problems are called *planning solvers* or *planners*.

One approach to solve a planning problem is to use the *state-space graph*. In this graph a node represents a *state* and each node state is related with other node states by the *actions* that can be performed in that state. With a set of node states and a set of action edges, a state graph is defined. A state is formed by a set of conditions in the environment and the initial conditions form a state known as the initial state. This graph is built from the initial state ensuring that the preconditions are fulfilled for each action; the result conditions from each possible action lead to a new state with different conditions. This process is repeated for each generated state creating a new state for each possible action in a node. Finally, when a state that matches the conditions of the goal is created with a path from the initial state that reaches it, the plan is the path of action edges from the initial state that reaches the goal state.

Modeling a planning problem using a state-space graph is computational difficult, because the graph grows exponentially [30, 31], which complicates the search for the goal. In computer science, the study of the computational resources needed to execute an algorithm is known as *algorithm analysis* and the study of how to describe a problem in terms of computational resources used to solve it is known as *computational complexity* [28]. The computational complexity defines the *complexity classes* to categorize a set of problems. It is based on the analysis of the minimum resources required by an algorithm to solve an instance in the worst-case scenario. One of the classes in which the problems require a huge amount of space and time for an algorithm to solve them is the complexity class PSpace. For examples the Plan Existence and the Plan Length problems of the classical planning are in this class [8, 30, 31].

In the past decades the *International Planning Competition* (IPC) has been organized by the *International Conference on Automated Planning and Scheduling*

(ICAPS), where a diversity of planning solver approaches participate testing its performances by solving sets of instances from different domains. The competition facilitates sharing the advances in the state of the art techniques, leads to study the domains used in the competition and incentivate the optimization of the solving techniques [5].

One of the most-used planners to solve automated classical planning problems in the competitions is *GraphPlan* [7]. It is an algorithm that constructs a multi-level graph, known as the *planning graph*, to represent the conditions and actions available from an initial state to search the goal. This algorithm, elaborated by Blum and Furst [5] is a rich source of information because it reveals the plan length and the reachability of the goal. Other planners use information retrieved from this graph to improve the search process or to bound the state space such as Blackbox, LPG, IPP, among others [7].

The planning graph is different from the state-space graph. In a state-space graph there are state nodes and each edge is an action, the levels are the number of edges from the initial state node to the goal state node and the conditions are repeated in each state. In a planning graph the conditions are nodes and the action joins its conditions with its effects, the conditions and effects are in different levels of the graph, each level represents an action in the plan and the states are sets of available conditions. This allows to build a model of the problem with less resources than the state-space graph [5].

Even though the planning graph uses less resources to create a model of the problem, the size of the model is not a definitive measure for the complexity of a planning problem, there are other factors that intervene, such as the quantity of delete edges allowed per action [8]. This leads to the study of domains to find other features that intervenes in the performance of the planners.

1.1 Hypothesis

Structural properties of the planning graph impact the performance of planner solvers and can be used to characterize the difficulty of the corresponding instance.

1.2 Objective

The objective of this thesis is to identify properties from planning graphs of the instances of classical planning problems to characterize its difficulty, and establish a methodology to categorize the instances difficulty based in performance of planners. This is performed by following the algorithm selection methodology to analyze which features impact the planning solvers performance.

1.3 Contribution

The present work establishes properties that identify instances which can be solved by the studied planners, a methodology is proposed to classify the instances based on the performance of the planners that tried to solve them. The development of tools to extract, store and analyze the used graphs to study the instances form part of hte contribution, taking advantage of the use of data bases to store the huge quantity of data, statistical analysis is employed to identify the properties in the studied instances.

1.4 Thesis Structure

This work is structured as follows. The background is developed in Chapter 2, consisting of three sections: automated planning, graph theory, and computational complexity; in Chapter 3, the previous works are reviewed and the differences among them are discussed; in Chapter 4, the methodology to obtain data is explained and the bases of the analysis are detailed. In Chapter 5, the proposed solution is described by explaining which properties are used, how the relations are established, and what are the expected results. In Chapter 6, the experiments and analysis of the results are presented. Finally, the conclusions and future work are discussed in Chapter 7.

Chapter 2

Background

This chapter explains all the background fundamentals of the present work, with the purpose of exposing the main concepts utilized. The first topic is planning consisting: the planning problem definition, existing approaches to solve it, and the *GraphPlan* algorithm. This is utilized in the present work to generate graphs and investigate what kind of relations exist in its structure. Afterwards, the graph-theoretical terms used to analyze the planning graphs are defined. Finally, the basic concepts to understand the complexity of planning problem are introduced.

2.1 Automated Planning

Automated planning is a branch of artificial intelligence that studies the process to devise a sequence of actions for an agent to achieve goals, based on a model of the environment where the actions will be carried out [27]. An *agent* is defined as any device that perceives its environment and takes actions that maximize its chance of success at some goal [31]. This model has four major components:

- State-transition system Σ.
- A state-transition function γ that defines the state conditions in the environment according to the actions modeled.

- Controller to perceive the environment state s and execute an action a according to a plan.
- Planner to devise a plan for the controller in order to achieve the goal, given as an input a system Σ, an initial state S_0 and some goal S_G.

Some automated planning problems forms are based on the assumption of restrictions which are considered when an environment is modeled [27]. The *restricted model* of automated planning consists in the following assumption restrictions.

A0 Finite Σ: the system Σ has a finite set of states.

A1 Fully observable Σ: it is completely known the state of Σ.

A2 Deterministic Σ: every action takes Σ to a one state only.

A3 Static Σ: the system Σ stays in the same state until an action is taken, there are no outside actions or further events.

A4 Restricted goals: attempt to reach the final state S_G, but it is not allowed to avoid certain states or use utility functions.

A5 Sequential plans: the solution plan is an ordered finite sequence of actions.

A6 Implicit time: actions does not have duration, the state transition is immediate.

A7 Offline planning: the agent is not concerned about system Σ while the planning process is being performed, there are no system supervision, new and restricted actions or variables after the initial state.

Modifying any of these assumptions completely changes the form of the automated planning problem; for example, if the assumption A6 does not hold it is known as *temporal planning*, if the assumption A2 does not hold it is *probabilistic planning*, if the assumption A7 does not hold it is *online planning*, and so on. When the eight assumptions are valid, it is known as *classical planning*.

The classical planning problem is defined as a system Σ, an initial state S_0, and a goal state S_G. The system Σ contains a γ transition-state function, a set A of finite actions, and a set S of finite states. The classical planning problem is to find a sequence of actions $(a_1, a_2, \ldots, a_k)$ that applied over the initial state S_0 it reaches the goal state. Resulting in a sequence of transition states $(S_0, S_1, \ldots, S_G)$ where each state is the result of the γ function applied over the previous state with the corresponding action $S_1 = \gamma(S_0, a_1), \ldots, S_G = \gamma(S_{G-1}, a_k)$.

A common way to manage a planning problem is to address each problem with specific representations and techniques adapted to the problem definition [27]. This is known as the *domain-specific approach*, where the definition of a specific model of actions and transitions is used to take advantage of some particular property. This improvement is dependent of the domain and is justified for having adequate solutions, but this do not contribute to the study or design of an autonomous planner. Because of this the automated planning focuses in an independent-domain approach to study the general and abstract sides of devising how to choose an action. The independent-domain approach is not opposed to the dependent-domain; it is important to have adequate and improved solutions in time and space, as it is also important to understand and study in general how any plan can be devised.

To describe the environmental conditions, the state conditions and the actions which are valid in an independent domain definition, there are different models to represent it, such as the set-theory or the state variable representations. These representations are restricted to the model; for this reason a more extended but equal expressive representation was developed, the Planning Domain Definition Language.

2.1.1 Planning Domain Definition Language

The Planning Domain Definition Language (PDDL) describes the environmental conditions where the agent starts planning. Four criteria are described to define a planning problem: an initial state, the available actions, the transiting state which contains the results of applying an action, and the goal state.

PDDL is an action centered language, based on *STRIPS* formulations (Stanford Research Institute Problem Solver) of planning problems, it is a simple standardization of the syntax for expressing a familiar semantic of actions, using the preconditions and effects to describe the applicability of the actions.

For PDDL a state S is represented as a conjunction of ground conditions. The initial state is conformed by the initial conditions of the world representation, where a closed-world representation is used, which means conditions not mentioned are considered false. A set A of action schemas defines the conditions needed to perform each action with its effects. This action schemas consists of an action name, the list of all variables used in the schema also called parameters, the preconditions, which consists of unions of positive literals without functional dependency that need to be true before the action, and the effects that are unions of positive literals without functional dependency which are true after the action.

Algorithm 1 PDDL action schema example

```
Action( Fly($plane$, $from$, $to$)                    ▷ Action name and parameters
    PRECOND: At($p$,$from$) $\wedge$ Plane($p$) $\wedge$ Airport($from$) $\wedge$ Airport($to$)
    EFFECT: $\neg$ At($p$,$from$) $\wedge$ At($p$,$to$)
)
```

The goal is a conjunction of literals that are true after executing the plan, and it is the result of an action, so the plan is the sequence of actions which are executed in the initial state resulting in a final state that satisfies the goal.

For PDDL, the domain definition refers to the action schemes defined with a set of variables that can be used in several problems, while the *problem definition* is defined in a specific domain, and it refers to the assignation of the variables, the initial state and the goal state.

2.1.2 Planning Models

There is a variety of approaches used to solve a planning problem according to the way the preconditions and effects are modeled.

When a planning problem is mapped into a satisfiability problem is called SAT PLAN, by replacing each action schema with a set of ground actions, every action variable is replaced for all possible values. This defines a new ground action for each configuration, and finally it is used a propositional solver to get the variables values that satisfies the goal [31].

A state-space search is the approach that tries to reach a goal state from the initial state by constructing a transition graph with all states, then searches inside the graph to find a plan that reaches the goal by a shortest path algorithm, here the plan is the sequence of operators corresponding to the edges. But this approach also grows exponentially in size [30, 31], for example, in a simple planning problem with thirty binary state variables there are $2^{30} = 1,073;741,824$ possible plans. With the help of heuristic search algorithms, a state-space search deals with this exponential growth.

In 1997, Blum and Furst [5] propose an algorithm known as *GraphPlan*. The algorithm creates a structure known as *planning graph*, in polynomial time, the algorithm also captures reachability information while creates the graph. The planning graph is directed and leveled with two kinds of nodes and three kinds of edges as seen in Figure 2.1.

In Figure 2.1 the levels alternate between two levels, in level 0 is a *proposition level* that contains of proposition nodes or fact nodes, which are represented with blue circled nodes, the level 1 is a *action level* containing action nodes which are represented with green squared nodes. Edges in a planning graph represent relations between actions and prepositions. The *action nodes* in action level i are connected by four kind of edges: *precondition-edges* which join the action node with the *proposition*

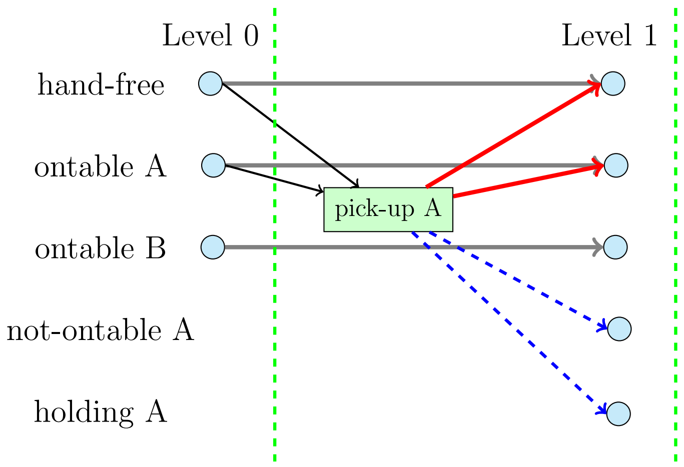


Figure 2.1: *GraphPlan* example. The blue nodes represent the facts, the green squared nodes represent actions, the black edges represent the action preconditions, the red edges represent the delete edges and the blue dotted edges represent the add edges; the green dotted lines divide the levels in the graph.

nodes in proposition level $i-1$ represented with black edges, *add-edges* which join the *add-effects* in proposition level $i+1$ represented with blue dashed edges and by *delete-edges* to their *delete-effects* in proposition level $i+1$ represented with red edges. The first level of a planning graph is a proposition level and consists of one node for each precondition in the Initial State [5].

At first, the planning graph has just a single proposition level containing the initial conditions as nodes which are also known as fact nodes, The *GraphPlan* algorithm runs in stages, in each stage the *GraphPlan* takes the planning graph from the previous stage and extends it one step, creating the next action level and the following proposition level by adding the add-effects, the delete-effects and also adding no-op actions which is a do-nothing action, leaving the same conditions for the next stage. Then the algorithm searches in the extended planning graph for a valid plan of length i, in each stage the *GraphPlan* algorithm either discovers a plan or proves that there is no plan of with i steps or fewer [5].

GraphPlan analyses the mutual exclusion edges among the nodes. For action nodes, two actions in the same level are mutually exclusive under three scenarios: if

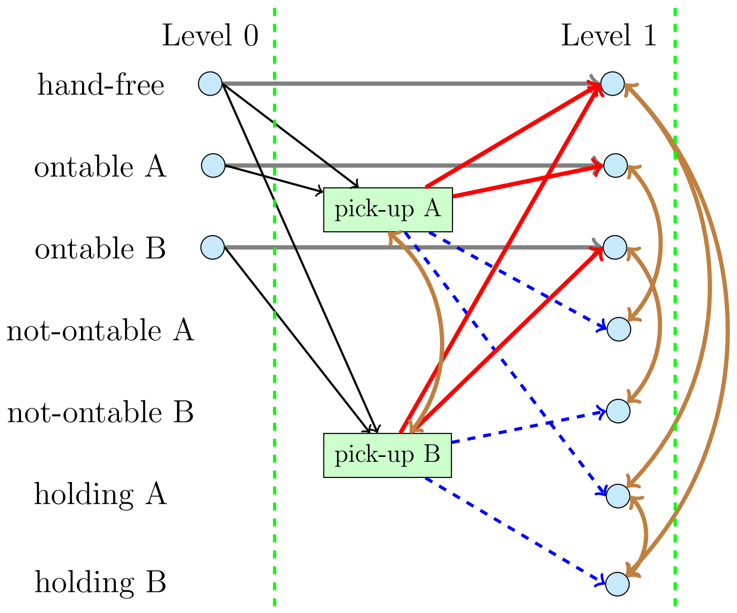


Figure 2.2: *GraphPlan* example with mutually exclusive edges represented with brown edges

the actions have inconsistent effects, this means that no valid plan can contain both actions, if the actions interfere with each other, and if the actions have mutually exclusive preconditions. Similarly if two preconditions cannot be true in a valid plan at the same time, then they are mutually exclusive, also known as mutex. Exclusive edges identify actions that interfere with each other as dependent actions, and competed preconditions, if two preconditions for different actions are marked as exclusive [5]. The analysis of the exclusive relations in the planning graph allows to identify properties of the instance.

The Figure 2.2 shows some mutex relations with brown edges. This graph is part of the planning graph generated from a blocks world problem, in this problem, there are certain number of blocks in a table. The actions of pick-up or put-down a block from the table, and the actions stack or unstack a block over other the goal is to form piles of blocks over the table in a certain order. The pick-up actions are mutually exclusive because each of them interferes with the other when hand-free is deactivated in every pick-up action, intuitively, it is not allowed to simultaneously pick up two blocks with the same hand. The ontable and not-ontable conditions are mutex because one is the negation of the other, just as the facts holding and hand-free.

The *GraphPlan* algorithm is sound and complete [5]: any plan the algorithm finds is a legal plan, and if there exists a legal plan then the algorithm will find one. Also *GraphPlan* creates a planning graph in polynomial time.

2.1.3 Planners

Planners are developed based on the approach used to model the problem. Planners that use a state-space graph and heuristics to guide the search with in the graph are LAMA, HSP, and FF. Blackbox, IPP, STAN, and LPG are based on *GraphPlan* and SATPLAN, MAXPlan are based on planning as satisfiability.

State-space graph planners have been improved to deal with temporal planning problems, while the planning as satisfiability approach was improved to avoid exploring all the possible variable configurations, both with the help of heuristic methods. There are planners which combine some or all of the approaches to get information to guide heuristics and optimize the planner performance such as AltAlt, Blackbox, and HSP.

2.1.4 International Conference on Automated Planning and Scheduling

The International Conference on Automated Planning and Scheduling (ICAPS) is a forum for researchers in the Artificial Intelligence fields of planning and scheduling. Since 1998 it is has been carried out by the International Planning Competition (IPC) to create planning domain problems, make comparisons among existing planners and measure the progress in the field [26]. In the 1998 competition, almost all participants were based on the *GraphPlan* algorithm [26]. After this, partial-order and satisfiability techniques were used with the *GraphPlan* algorithm to improve the state-of-the-art solvers.

Nowadays, IPC includes different competition tracks, besides the classical planning, there are temporal and probabilistic planning tracks due to the advances achieved since the beginning of the competition. Also the classical competition track has been improved by supporting new requirements and creating new domains to research.

Each competition proposes a set of domains where a set of problems are defined, also new planners are introduced using new techniques to be tested. In the remain of this section, the available information of each competition is described.

2.1.4.1 IPC 1998

The competition domains, problems used, and the participant planners were reported by McDermott [26]. The information in this section is mainly based on that work, and on the historic web page of the competition[1], and later works [16, 22, 36].

This competition was featured by the Artificial Intelligence Planning and Scheduling Conference (AIPS), there were five participants planners in the STRIPS track with five domains in total. Table 2.1 describes the domains used and Table 2.2 describes the participanting planners.

Besides the planers exposed in Table 2.2, there was other domain defined in the competition, the assembly domain that was not used in the present work because it is not in a STRIPS definition and was not possible to get the planning graphs from it. Each domain has 30 instances.

[1]http://icaps-conference.org/index.php/Main/Competitions

Table 2.1: IPC 1998 planners.

Planner name	Description
Blackbox	A *GraphPlan* based planner that converts the planning graph into a Satisfiability problem.
IPP	An optimized implementation of *GraphPlan*, extended to handle expressive actions.
HSP	A planner based on heuristic search guided by means-ends analysis.
STAN	An optimized implementation of *GraphPlan* that uses an in-place graph representation and performs sophisticated type analysis to compute invariants.

2.1.4.2 IPC 2000

This competition was also featured in the AI Planning and Scheduling Conference, the competition domains used and the participant planners were reported by Lin et al. [25]. The information of this section is mainly based on that work and of the competition web page.

In this competition fifteen planners contest, with five domains in total. The description of the domains used is in Table 2.4 and the description of the participant planners is in Table 2.3.

Table 2.3: IPC 2000 planners

Planner name	Description
Blackbox	A *GraphPlan* based planner that converts the planning graph into a Satisfiability problem.
IPP	An optimized implementation of *GraphPlan*, extended to handle expressive actions

Planner name	Description
HSP2	A descent of the HSP planner of the 1998 IPC. The idea is similar planning instances are mapped into state-space search problems that are solved with heuristics extracted from the representations.
STAN	An optimized implementation of *GraphPlan* that uses an in-place graph representation and performs sophisticated type analysis to compute invariants.
MIPS	Uses binary decision diagrams to compactly store and maintain sets of propositionally represented states. The concise state representation is inferred in an analysis prior to the search and, by utilizing this representation, accurate reachability analysis and backward chaining are carried out without necessarily encountering exponential representation explosion
System R	A planner based on regression, and solves a goal one at a time.
FF	Fast-Forward is a variation of the HSP. Using *GraphPlan* information, FF employs a novel local search strategy that combines Hill-climbing with systematic search.
PbR	Planning by Rewriting takes the output of a simple, low-quality planner and rewrites the plan to produce a better plan.
Propplan	Based on naive breadth-first state-space search, uses ordered binary decision diagrams to implement exhaustive state space exploration.
SHOP	Simple Hierarchical Ordered Planner, is a domain-independent HTN planning system that uses a novel left-to-right plan-elaboration scheme
TokenPlan	Combines classical AI planning and game theory. It handles sets of states relevant to the criteria which are optimized by the game-theory to control the level of splitting of the search space.
BDDPlan	Uses binary decision diagrams to support reasoning in the fluent calculus.

Planner name	Description
TALPlanner	Domain-dependent search control knowledge expressed as temporal logic formulas is used to effectively control forward chaining.
AltAlt	A planner that uses effective and admissible heuristics extracted from the planning graph to drive the backward state space search.
GRT	Greedy Regression Table planner is a domain independent heuristic state space planner.

2.2 Graph Theory

This thesis is principally influenced from graph theory, so a basic introduction is presented to set a background, primarily based on the book of Diestel [9].

A *çgraph* is a pair of sets $G = (N, E)$ where E is contained in the Cartesian product of N satisfying $E \subseteq N \times N$. The elements of N are called *nodes*, vertices or points; the nodes are denoted by lowercase letters as a or b assuming $a, b \in N$ and if the nodes are enumerated or labeled then are represent as a_0 or a_{label}. The elements of E are called *edges* or lines. The node set of a graph is referred as $N(G)$ and the edge set is referred as $E(G)$; an edge involving two nodes is usually written as ab, E_{ab} or $E(a, b)$ where $a, b \in V$.

The number of nodes in a graph is known as its *order*, written as $|G|$, $|N|$, or n, the number of edges is denoted as $||G||$, $|E|$, or m. A node a is *incident* with an edge E_{ab} if $a \in E_{ab}$, then E_{ab} is an edge at a. The set of edges incident to node a is denoted as $E(a)$; the two nodes incident with an edge are end nodes or ends, and the edge joins its ends. Two nodes a, b are adjacent or neighbors if an edge exists in G with a and b as end nodes, and if two edges share one end then are adjacent edges.

If all nodes in G are pairwise adjacent, then G is a *complete graph* K_n, otherwise

Table 2.2: IPC 1998 domains.

Domain	Description
gripper	A robot must move a set of balls from one room to another, being able to grip two balls at a time, one in each gripper.
logistics	There are several cities, each containing several locations, some of which are airports. There are also trucks, which can drive within a single city, and airplanes, which can travel between airports. The goal is to get some packages from various locations to various new locations.
movie	The goal is to have lots of snacks in order to watch a movie. There are seven actions, including rewind-movie and get-chips, but the number of constants increases with the problem number.
mystery	There is a transportation network through which vehicles could move carrying cargoes. A vehicle could move from one node to a neighbor node if there were fuel at the originating node.
mprime	This is the mystery domain with one extra action, the ability to squirt a unit of fuel from any node to any other node, provided by the originating node has at least two units.

Table 2.4: IPC 2000 domains

Domain	Description
blocks world	A finite number of blocks stacked into towers on a table large enough to hold them all.
logistics	Same domain used in the 1998 IPC, where a collection of packages at various locations and cities and are transported between locations by trucks or airplanes.
schedule	This domain consists of nine machining operators that do things such as paint, grind, and drill an object. The goal is to transform some collection of objects by performing various machining operations on them.
freecell	The solitarie card game included in the WINDOWS operating system, with four squares known as the home cells, one for each suit and four free cells that act as temporary space. The goal is to move all the cards home in order.
elevator	Based on a sophisticated elevator that given a floor request to a controller, it indicates which elevator use in an attempt to minimize the delay time.

it is a *incomplete graph.* A set of nodes or edges is *independent* or stable if any pair of its elements are not adjacent.

If G is a non-empty graph, the set of neighbors of a a node in G is denoted as $N_G(a)$ or if the graph reference is clear just by $N(a)$. The *degree* of a node is the number of edges at a, $d(a) = |E(a)|$, in a simple graph the degree of a node is equal to the number of neighbors of that node. The number $\delta(G) := \min\{d(a) \mid a \subseteq N(G)\}$ is the *minimum degree* of G, and the *maximum degree* is $\Delta(G) := \max\{d(a) \mid a \subseteq N(G)\}$. If $\delta(G) = \Delta(G) = k$ then G is *k-regular*, or simply regular. The *average degree* of G is given by $d(G) := \frac{1}{|N|}\sum_{a\subseteq N(G)} d(a)$. The *density* of a graph is given by $\epsilon(G) := \frac{|E|}{|N\times N|}$.

2.3 Computational Complexity

Computational complexity is an area of computer science that classifies the problems according to its inherent difficulty. An *algorithm* is a sequence of actions to be performed from an given input to reach an output, so a solve procedure for a problem is an algorithm. The *analysis of algorithms* is the area where is the studied the resources needed by an algorithm to solve a computing problem. For example, a sorting problem, having all possible permutations of the elements can be solved by reviewing each one, but the number of permutations is given by $n!$ where n is the number of elements to sort.

Considering the resources, the model of computation and the mode of computation, a problem can be classified into a complexity class. The *model of computation* refers to the definition of allowed operations and its resource cost, for example, in a *Turing machine* (TM) defined by four major components: a tape, that is divided in cells and in each cell a character from a finite alphabet is placed, a header that writes and reads the characters over the cells of the tape, a set of states which represents the possible states of the TM, and a transition table, with the instructions to perform based on the TM state and the characters on the tape. The number of cells used on the tape is the resource of space, while the number of instructions performed by the TM, is the resource of time. The *mode of computation* refers to a probability to change from one state to another, a TM can be defined in a deterministic or in a non-deterministic mode.

In a TM, the list of characters on the tape are called words, and the set of words computed by a TM that reach a final state is known as a language. A TM can simulate any algorithm with out significant loss of efficiency [28]. All computing problems or languages handled by a TM in a specified mode can be classified according to a hierarchy class based on the size of the problem instance and the main resource to be bound, principally space and time. So, a complexity class is defined as the set of all languages decided by a TM operating in the appropriate mode, such that for any input x, the TM expends at most $f(|x|)$ units of the specified resource [28].

Some complexity classes [28] used in this thesis are:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PS{\scriptsize PACE}} \subseteq \mathrm{E{\scriptsize XP}T{\scriptsize IME}}; \tag{2.1}$$

$$\mathrm{PS{\scriptsize PACE}} = \mathrm{NPS{\scriptsize PACE}}. \tag{2.2}$$

The class P represents all computing problems solvable in deterministic polynomial time $\mathrm{P} = \mathrm{T{\scriptsize IME}}(\mathit{poly}(n))$ for an input size $f(n)$, the class NP represents all computing problem solvable in a non-deterministic TM in a polynomial time $\mathrm{NP} = \mathrm{NT{\scriptsize IME}}(\mathit{poly}(n))$ for an input size $f(n)$, the class PSpace represents all computing problems solvable with a polynomial size of space $\mathrm{PS{\scriptsize PACE}} = \mathrm{S{\scriptsize PACE}}(\mathit{poly}(n))$ for an input size $f(n)$, and ExpTime represents all computing problems solvable in a exponential amount of time $\mathrm{E{\scriptsize XP}T{\scriptsize IME}} = \mathrm{T{\scriptsize IME}}(2^{\mathit{poly}(n)})$ for an input size $f(n)$.

While the classes P, NP, and ExpTime are by definition bounded by time, the class PSpace is bounded by space: there is a polynomial number $\mathit{poly}(n)$ of space units so there are $2^{\mathit{poly}(n)}$ possible values, and if there are k possible states, at worst case, a solve procedure which uses a polynomial amount of space can take

$k \cdot poly(n) 2^{poly(n)}$ steps to end, which is a exponential amount of time ExpTime.

Equation (2.2) refers to an important complexity result, the Savitch theorem. It establishes that an algorithm in a non-deterministic mode which uses a polynomial amount of space can be transformed into an algorithm in a deterministic mode using an square amount of space.

Three important definitions in the complexity classes field are *reductions*, *completeness*, and *hardness*. A reduction between two problems A, B refers to a transformation T which for any input $x \in A$ that answers the problem A produces an input $y = T(x)$ for the problem B, where this transformation take at most polynomial time. Completeness in a complexity class refers to all set of problems x in a complexity class for which is possible to reduce any other problem in that class y to the problem x. Finally, hardness in a complexity class refers to a problem y that can be reduced to a complete problem in that class in polynomial time.

The theoretical complexity of planning is in the complexity PSpace. Even if some severe restrictions are made, the problem often remains quite difficult; for example, if negative effects are disallowed, is reduced to an NP-hard problem, but if negative preconditions are disallowed is reduced to the class P. Therefore, by its complexity, optimal planning is usually hard, but sub-optimal planning is sometimes easy with the help of good search heuristics [31].

Chapter 3

Related Work

The existing approaches for studying the planning problems are presented in this chapterin the form of literature review of automated planning, the computational complexity works in automated planning and similar works in other fields studying the structural properties on problems to improve solving procedures.

3.1 Literature Review

Automated planning research begins in 1969 with the development of theorem solvers [13]. The translation from a theorem solver to an automated planning is defining a domain where the axioms are the initial conditions, the actions are the operations allowed between the axioms, and the goal is locate a specific theorem given a set of axioms. The theorem solver problem was quite difficult given the hardware and techniques available, it was even considered intractable. In 1971, with the STRIPS definition [11] the basis for classical planning was established and the automated planning research started to grow.

In 1991 Bylander, Bäckström and Nebel [3, 8] published a series of works that established the classical planning problem as the complexity class PSpace-complete considering the models of the STRIPS and for the SAS+ definitions.

These definitions are equivalent differing how to definite the actions and conditions, but for the same domain a SAS+ definition can be transformed in a STRIPS definition and vice versa, they are equally expressive [1].

The results of Bylander take away the intractable status of the automated planning problems and proves that under certain restrictions in the domain definition there are planning problems in other complexity classes. For example, a problem definition that allows no negative postconditions is in NP-complete complexity class or a problem definition that allows no negative preconditions and just one post condition is in P complexity class.

Alongside the computational complexity study of the planning problems, in the automated planning field new techniques were developed in order to solve the planning problems with planners such as POP, *GraphPlan*, SATPLAN among others. These planners solved planning problems with reasonable resources using heuristics, memoization, relaxation techniques, etc. These results establish a series of works where other planning problem types were studied by computational complexity [2, 30] and also a study for the computational complexity of the domains used in IPC and the benchmarks for research [12, 14, 16].

Another important result was the development of the *GraphPlan* algorithm. It implements a structure that uses a polynomial amount of space besides others structures which uses a exponential amount of space [5]. *GraphPlan* uses a limited structure to solve the planning problems where others solvers, such as SATPLAN, used to create a exponential structure to simplify it. These differences result in the development of two kind of planners that perform differently on same domains: the planners that use a propositional satisfiability solver, called SATPLAN, and planners that use *GraphPlan*.

The sequential domains form a hierarchy that have mutual exclusive relations that restrict parallelism and forces a sequence of actions in it [24], the non-sequential hierarchy of domains does not have this restriction, and with in it was established the transportation domain hierarchy [16, 18], where is possible to take advantage of parallelism according of the locations, mobiles, and portables defined.

An important result is that the SATPLAN-based planners had fast results on the sequential domains, while *GraphPlan* based planners are faster in non-sequential domains with parallel action such as the transportation domains [16, 24].

Kautz and Selman [24] develop *Blackbox*, a planner that creates a plan graph based on *GraphPlan* algorithm to try to solve it, and if no solution is found, parses the graph to a *conjunctive normal form well-formed formula* (CNF wff) and uses satisfiability heuristics to search for a plan. The integration of these approaches was motivated by the discovery of the sequential hierarchy in planning problems and resulted in a planner that had the best results in IPC of 1998 [26].

Furthermore Hoffmann and Nebel [21] use *GraphPlan* to take advantage of a result from Bylander [8, 21] to create the *Fast-Forward* planner. Bylander established the difficulty of plan existence in planning problems with non-negative postconditions as NP-hard [8], the Fast-Forward planner uses the relaxed problem with not negative postconditions, represented by delete edges in the planning graphs and solve it, the plan obtained by the relaxed solution is used as heuristic evaluation to estimating the goal distance [19–21]. The planners based on this approach are known as heuristic-based planners.

In the lastest competitions, the approach on which most of the planners are based is the *Fast-Downward* planner. The Fast-Downward planner was developed by Helmert and is a heuristic based planner but instead of using relaxed problem, the planner take advantage of causal relations in actions to constructs a structure known as *causal graph* to use it to guide the plan search.

The previous planners were develop by studying the planner performance, the domains characteristics, and its computational complexity. This is quite familiar in optimization, where given a optimization problem is important to understand the relationship between the algorithm performance and the optimization problem on which it runs. It is known that there is no algorithm performs well on all classes of problems regardless the structure and characteristics of the instance [38]. The algorithm portfolio approach uses this to use the appropriate algorithm given the structure of the instance.

An example of an algorithm portfolio was exposed in the 2007 SAT competition [39] where the situation is quite similar to the situation presented in IPC. There is a variety of solver approaches available and rather than choosing the best solver for a given class, an online decision maker chooses the solver that offers the best performance for the given instance based on machine-learning models of instance features and the algorithms past performance. Algorithm portfolio is also used in other problems, such as scheduling, knapsack, and job shop [15, 33, 34, 39].

In early competitions, some planners have the opportunity to choose the algorithm to use and the heuristic based planners. However in recent competitions, the portfolio approach is widely used in the state-of-the-art planners, optimizing running time and solution quality of the planners and heuristics performance [35].

The algorithm portfolio is mainly based on the algorithm-selection problem [29] to decide which algorithm from the portfolio is likely to perform best based on measurable features of a collections of instances. This algorithm uses four components:

- **Problem space:** the set of instances.
- **Feature space:** the set of measurable characteristics of instances.
- **Algorithm space:** the set of algorithms, also known as the portfolio.
- **Performance space:** the set of mapping function for each algorithm to a set of performance metrics.

Based on the components of the algorithm, the portfolios used in the planner solvers mainly generated the performance space with performance of the algorithms

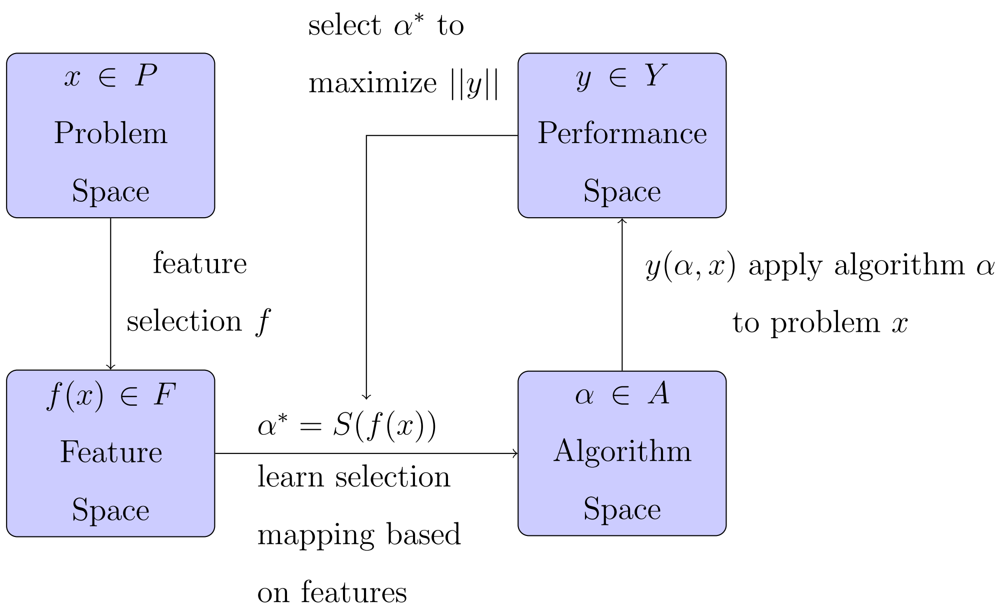


Figure 3.1: Components of algorithm selection

on the set of problems from the past competences, while the feature space varies in each planner according with the approaches used by the heuristics, the selections mapping $S(f(x))$ is inferred with machine-learning techniques [35].

Analyzing the features used by the *Fast-Forward* planner [21], where a relaxed problem without negative postconditions is used to guide the plan search. Bylander [8], demonstrate that the relaxed problem without negative postconditions is NP-complete, becuase it removes all mutex relations from the graph [21].

Bäckström et al. [4] analyze the unsolvable planning problems instances through mutex groups: sets of propositions or facts that any state reached from the initial state have one mutex proposition. They suggest it as an important feature to analyze.

3.2 Comparative Study

The computational complexity works on automated planning are mainly focused on the analysis of the problem definitions [3, 8], although there are studies of the computational complexity of the domains [12, 14, 16, 22, 26] no study has been published that tries to identify a structure in the problem based on the analysis of the problem structure of the planning graphs.

Algorithm portfolios are designed to offer a optimized performance of the planning solver based on previous information; there is no study of features along the domains and planners in order to obtain a metrics about the difficulty or to identify features related to the performance of the planners.

In Table 3.1 a comparison with other works is presented considering the following features: if there is a difficulty classification of the instances in the work, if there is a feature analysis of the instances, if there is a domain feature analysis, algorithm features analysis, problem structure analysis, complexity analysis, or whether it is planning oriented. The work considered is discussed next.

Kautz and Selman [24] present the Blackbox planner based on an analysis of the algorithms and problem structure that identifies a benefit of the *GraphPlan* over the domains with parallel actions is one of the result that motivates the fusion of the *GraphPlan* and SAT planner techniques; the present work follows this perspective and extends it with a analysis of the graph structure of the instances to identify features by domain and problem.

Hoffmann and Nebel [21] present the Fast-Forward planner based on the analysis of the domains, algorithms and instance definition of planning problems, in this works was followed its perspective to identify instance structure features that could impact the performance of a planner.

Helmert [16] presents complexity results based on the domain definitions. Also the transportation hierarchy is defined in the present work. The present work follows this approach to identify some domain hierarchy based on the graph structures of the instances.

Smith-Miles and van Hemert [33] present a portfolio algorithm for the Travel Salesman Problem. The structure of a algorithm portfolio is taken in account to use it as a base for the feature discovery analysis.

Vallati et al. [35] present the catalog of the participant of IPC of 2014 where some algorithm portfolio are presented, this portfolios use machine learning techniques to categorize some problem features over the best planner for it; the present work is different because is focused in the instance structure.

Bäckström et al. [4] study the unsolvable instances of planning problems based on features of the instance structure. The present work also tries to contribute in also identify features for unsolvable problems.

3.3 Contribution

The present work offers a different approach to identify features on instances that impact the planner performance. Taking advance of planning-graph structure, is possible to study near than four hundred graphs structures and near than six million nodes. With the quantity of information generated, tools were developed to analyze the graphs and databases were implemented to facilitate the study of the graphs. Finally, the present work offers a classification method for the domains based on the instance performance.

Table 3.1: Comparison of cited works.

Feature	Kautz and Selman [24]	Hoffmann and Nebel [21]	Helmert [16]	Smith-Miles and van Hemert [33]	Vallati et al. [35]	Bäckström et al. [4]	Present work
Difficulty classification	✗	✗	✗	✓	✓	✗	✓
Problem features analysis	✗	✗	✓	✓	✓	✓	✓
Domain features analysis	✗	✓	✓	✗	✗	✓	✓
Algorithm features analysis	✓	✓	✓	✓	✓	✓	✓
Problem structure analysis	✓	✓	✓	✓	✓	✓	✓
Complexity analysis	✗	✓	✓	✗	✗	✓	✓
Planning oriented	✓	✓	✓	✗	✓	✓	✓

Chapter 4

Methodology

The methodology is discussed in this chapter, starting with the description of the planning-instance analysis; later, the tools used to obtain the graphs and the data extracted are explained. Finally the domains, algorithms, and the available performance results are discussed.

4.1 Description of the Analysis

In order to study the planning instances and identify the structural properties to categorize them, this analysis is focused in the structure of the instances and the planning solver performance. By following the portfolio algorithm [29, 33], the algorithm-selection components are used as guide as the base of this analysis, defining the problem set, the algorithm set, the performance set and the feature set.

To obtain the problem set, the Blackbox planner is used to extract the planning graphs from each instance. Blackbox was developed by Kautz and Selman [24] and presented in the 1998 International Planning Competition (IPC) [26]. This planner is based on the original *GraphPlan* planner. It use GraphPlan to find a plan and when no plan is found, the planning graph is parsed into a conjunctive normal form (CNF) well-formed formula (wff) to use solving methods based on the planning as propositional satisfiability (SATPLAN) [23]. Blackbox works with the PDDL definition domain meanwhile *GraphPlan* planner uses the STRIPS definition domain. The diversity and quantity of the available instances in PDDL compared to the instances in STRIPS is the main reason why Blackbox was used to obtain the planning graphs. This allows us to study most of problem definitions used in IPC, which are in PDDL definition.

The Blackbox implementation is written in C++ programming language and it use a parsed code from the *GraphPlan* implementation from Blum and Furst [5] which is written in C programming language. Both implementations use a list of node structures to represent the graph, where each node has a list of its edges. To extract the graphs, a deprecated function that draws the planning graph is modified to parse its graph structure into a file, this avoid the modification of the original algorithm.

The graph structure was parsed to a (JSON), because it allows to represent the nodes and edges of the graph with all the properties defined in the Blackbox implementation. Extracting the graphs as JSON also allows to analyze the graph outside the planner solution process. Finally, the JSON files are stored in a database. The list of properties defined in the node class of the *GraphPlan* and Blackbox is given in Table 4.1, the graph class contains the instance name and the list of nodes. The graph information is the basis for obtaining the feature set for the algorithm selection.

As mentioned, the instances used in IPC are available in PDDL on the ICAPS page[1]. This archive contains all competitions, since the first one in 1998 until present. The competitions of the years 1998 and 2000 were selected for executing the plan graphs and performing the analysis.

This archive also contains the results of the competition for each planner that executed an instance; if a plan was obtained, the plan length and execution time

[1]http://icaps-conference.org/index.php/Main/Competitions

Table 4.1: Description of node properties used in *GraphPlan*.

Property	Description
Name	Node name.
Hashval	Hash identifier.
Type	Node type.
In edges	List of in edges.
Out edges	List of out edges.
Del edges	List of delete edges.
Exclusive	List of exclusive edges.
Excl in this step	List of exclusive nodes in this level.
Is true	Indicates if the node is included in plan.
Cant do	Indicates if the node is exclusive.
Uid block	Unique identifier for the block.
Uid mask	Unique identifier for the time.
Is noop	Indicates if the node is a No-Op action.

Table 4.2: Performance values of the planners.

Property	Description
Name	The instance name.
Comp	The competition name.
Planner	The planner name.
Time	Reported execution time.
Steps	Reported plan length.

are reported also the comparatives among the planners results are included. Table 4.2 presents the available information from the selected IPC competitions about the planners and its performance for each instance. The results comprise the performance set and the participant planners are the algorithm set.

In order to obtain the graphs, the Blackbox planner needs to be execute until a solution is found or it halts. The parameters to set are the quantity of nodes allowed, the time to execute the plan search by each method, and number of levels or steps for the plan length search. Such information is not available in the competition result reports, so the parameters were set as follows, up 32,000 nodes per level, which is the maximum value allowed, up to 10 seconds of execution for each solver available in Blackbox in each level and, 200 levels before halting.

The planning graphs are multi-leveled with two types of nodes and four types of edges. There are fact and action nodes, and for the edges there are in, out, delete, and exclusive edges. On each level, the fact nodes are connected by the in edges to the action nodes of its level, are connected with the out and delete edges of the previous level, and with the exclusive edges between those facts that have a mutex relation; the action nodes also have exclusive edges with those actions that are in a mutex relation.

With the extracted planning graphs of the Blackbox planner, an analysis is done to identify which characteristics of the graph are used as the features of the feature set. First a python script was written to perform the analysis in each JSON file, but due to the quantity of graphs and its file size the time to read each file was huge, there were more than 600 graphs with sizes with some MB in small instances to sizes above of 10 GB in the big instances.

The graphs are structures with collections of nodes and collection of edges, therefore a database was used to store the graphs and perform the analysis through queries. This reduces the analysis time and makes it easy to obtain the summarized data of graphs. The selected database was MongoDB[2], an open-source database that instead of storing the information in table structures, manages documents with schemas. The JSON representation of the graph includes in each node a list of each edge by type; the MongoDB documents facilitate storing this as a whole document instead than creating tables with the node and edge information. With the JSON representation of the planning-graph instances, importing them to the database was done with a python script.

Based on the planning-graph structure, the characteristics for each node of the graph were analyzed to studying the distribution of the characteristics in all the graph and on each level. These characteristics are described in Table 4.3.

With the information in Table 4.2, a node metric can be used among instances, and level information can be grouped; the list of characteristics used by level and by graph are in Tables 4.4 and 4.5.

With the summarized information by level and graph, metrics like ratios of nodes and edges by type, growth by level can be obtained. Now, that is the defined the methodology to obtain the feature metrics, a review of the planner is exposed describing its performance on the competition results to identify common features among them.

[2]https://www.mongodb.com/

Table 4.3: Description of the node attributes used in the analysis.

Property	Description
Hash	Hash value of node name, used to identify the node in the graph.
Name	Node name.
Type	Type of the node a fact or an action.
Level	Level number where the node belongs.
Total in degree	Total amount of edges of all types were the node is the tail.
Total out degree	Total amount of edges of all types were the node is the head.
Total in edges	Total amount of in edges of the node.
Total out edges	Total amount of out edges of the node.
Total delete edges	Total amount of delete edges of the node.
Total exclusive edges	Total amount of delete edges of the node.

Table 4.4: Description of level-based metrics used in the analysis.

Property	Description
Gid	Identification key of the graph.
Name	Instance name from which the graph was extracted.
Level	Level number.
Total	Total nodes in the level.
Total facts	Total fact nodes in the level.
Total actions	Total actions nodes in the level.
Total mutex facts	Total mutex fact nodes in the level.
Total mutex actions	Total mutex actions nodes in the level.

Table 4.5: Description of graph-based metrics used in the analysis.

Property	Description
Gid	Identification key of the graph.
Name	Instance name from which the graph was extracted.
Competition	Competition name.
Total levels	Maximum number level in this graph.
Total	Total nodes in the graph.
Total edges	Total edges in the graph.
Total facts	Total fact nodes in the graph.
Total possible edges	Maximum possible edges in the graph.
Total actions	Total actions nodes in the graph.
Total mutex facts	Total mutex fact nodes in the graph.
Total mutex actions	Total mutex actions nodes in the graph.
Density	The density of the graph edges.

4.2 Input Data

Each competition has a different set of domains and planners, so for each one is analyzed which features can be used. To identify the factors that influence the planner performance, the information already available in the competence results was used, the main results reported are the plan length and the execution as shown in Table 4.6.

Table 4.6: Description of performance measures of planners.

Metric	Description
Competition	Name of the competition where the result is reported.
Planner	Name of the planner used to solve the planning problem instance.
Domain	Name of the domain where the problem instance is contained.
Problem	Name of the problem instance, each one is different of each other.
Plan length	The plan length reported by the planner: -1 if no plan found.
Time	Time reported in milliseconds of the planner execution.

4.2.1 Results of IPC 1998

The domains, instances, and participants of IPC 1998 were described in Section 2.1.4.1. The full dataset is available in the git repository[3] of the present work. In Figures 4.1, 4.2, 4.3 the variation of time, length and solved instances in each planner by domain are presented.

As shown in Figure 4.1, the domains for *movie*, *mprime*, and *mystery* are the ones that require fewer actions and *gripper* and *logistics* have instances with more than fifty steps; *mystery* and *mprime* domain are quite similar as *mprime* has one more action than *mystery*, this is reflected with similar plan lengths. In Figure 4.2 is

[3]https://github.com/ppGodel/StructuralCharacterizationOfPlanningProblems

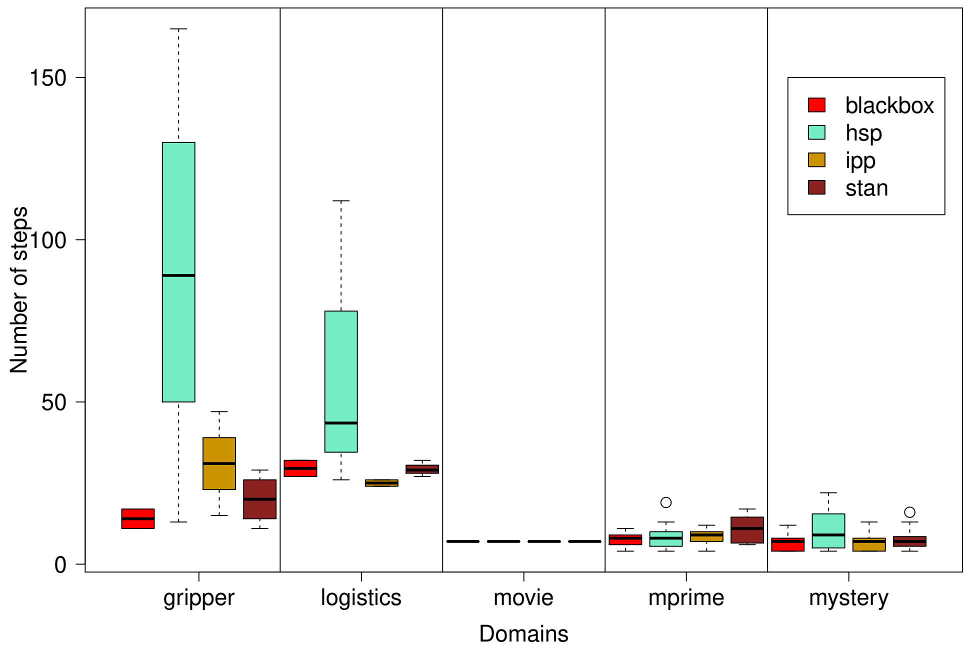


Figure 4.1: Comparison between plan length and domain in IPC 1998

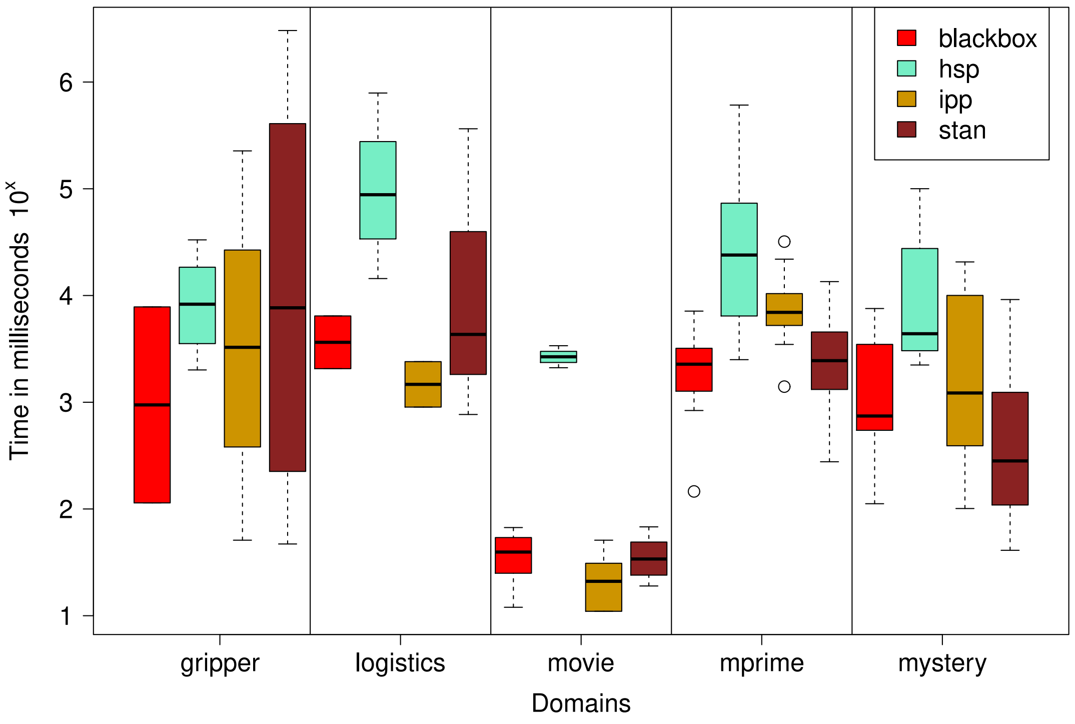


Figure 4.2: Comparison between solution time and domain in IPC 1998

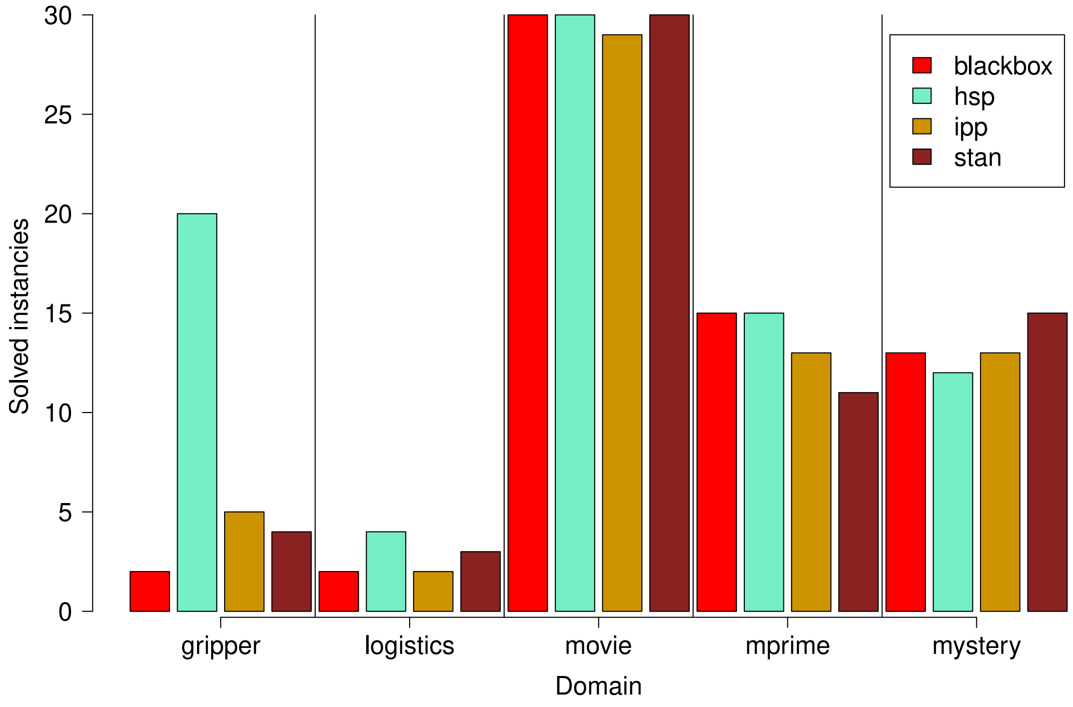


Figure 4.3: Number of solved instances by planner in each domain in IPC 1998

the time comparison, the *movie* domain requires less than a second to get a plan the other domain which require a greater amount of time, also the mystery and *mprime* have similar times. Finally the quantity of solved instances by planner and domain is shown in Figure 4.3, is important to mention that not all instances were resolved, specially for gripper, the HSP planner solved the most quantity of instances and solved the longest plans but use more time. The *GraphPlan* based planners solved almost the same quantity of instances.

A main result from 1996 by Kautz and Selman [23] is the definition of *sequencing domains*. In the sequential domains the interaction of exclusion edges between the actions and sub-goal sequences result in a sequential actions goal. For this competition, *assembly* is the only sequential domain but it was not included in this analysis. The domains *gripper*, *logistics*, *mystery*, and *mprime* are transportation domains which could be solved with parallel actions, this will be considered in the difficulty classification analysis.

The competition results suggest some facts to consider: the instances in the *movie* domain appear to be easy to solve, considering the small amount of steps and time required and that almost all planners solved them. The instances in domains *mprime* and *mystery* appear to be average in difficulty, considering that them use a grater amount of time and steps than the *movie* domain and not all instances were solved. Finally the instances in domains *gripper* and *logistics* take almost the same time to be solved, but have larger plan lengths and fewer instances solved, which suggest that they are the more difficult instances to solve in this competition.

4.2.2 Results of IPC 2000

For this competition the domains, instances, and participants of the IPC 2000 were described in Section 2.1.4.2. The full dataset is available in git repository[4] of the present work. In Figures 4.4, 4.5, and 4.6 the variation of time, length, and solved instances for each planner by domain are presented according with the registered results.

This competitions had a increased and diverse participation with fifteen planners and five domains. The plan length comparison is shown in Figure 4.4: almost all planners have similar performance and just one planner solved instances with plans of 150 or more steps. For domains, *freecell* had longest plans and *schedule* the shortest plans. The time comparison of planners is shown in Figure 4.5. In contrast with the plan length, the times reported had more variance for all planners; no planner is the quickest or the slowest for all domains. For domains, *schedule* had the shortest plans. Finally, the percentage of solved instances by planner and domain is shown in Figure 4.6 where just *system-r* planner solved more instances over all domains, and the *elevator* domain had the greatest number of planners that solved almost all of its instances.

[4]https://github.com/ppGodel/StructuralCharacterizationOfPlanningProblems

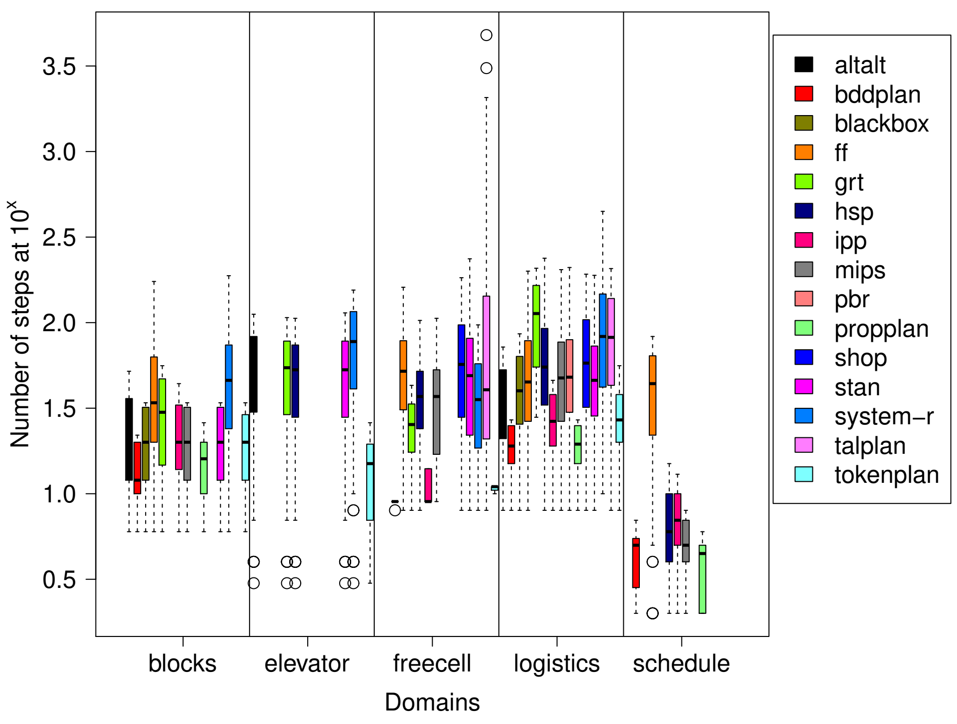


Figure 4.4: Comparison between plan length and domain in IPC 2000.

4.3 Preprocessing

In the studied competitions, blocks is the only sequential domain. The domains *elevator* and *logistics* are consider as transportation domains and *freecell* and *schedule* are not considered in a particular type of domains. For the IPC 2000, the domains were more challenging in terms of the percentage of solved instances and times reported. While in IPC 1998 there were no domains with long plans, the IPC 2000 had several instances with long plans. For the percentage of solved instances and time required to solve them, the *logistics* and *schedule* domains present more difficulties.

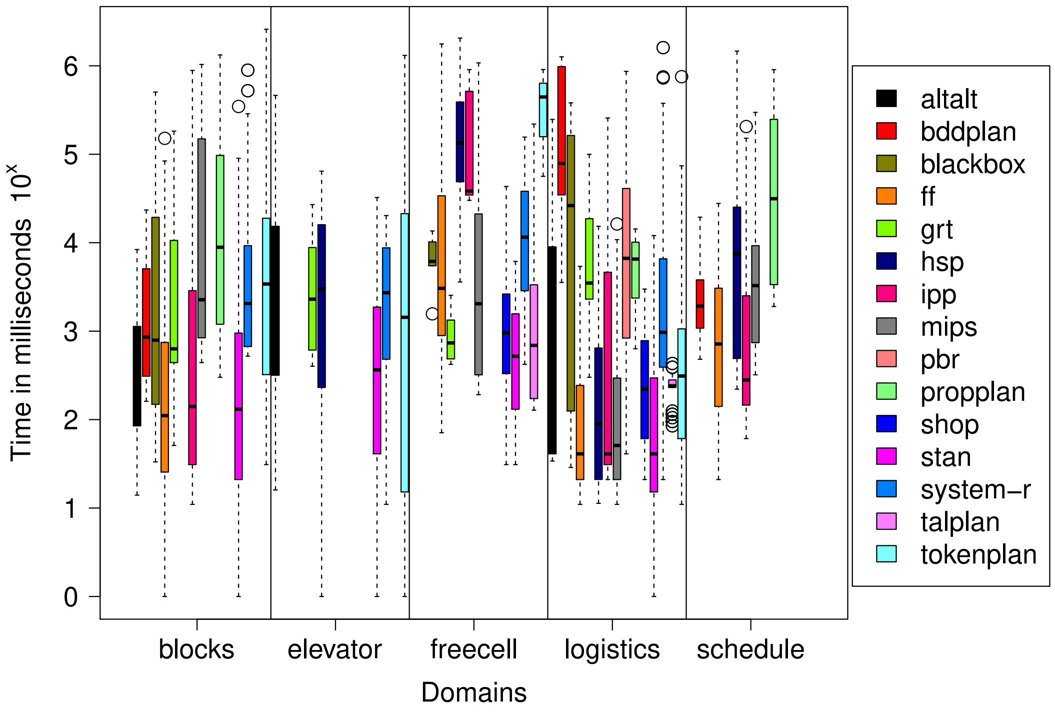


Figure 4.5: Comparison between solution time and domain in IPC 2000.

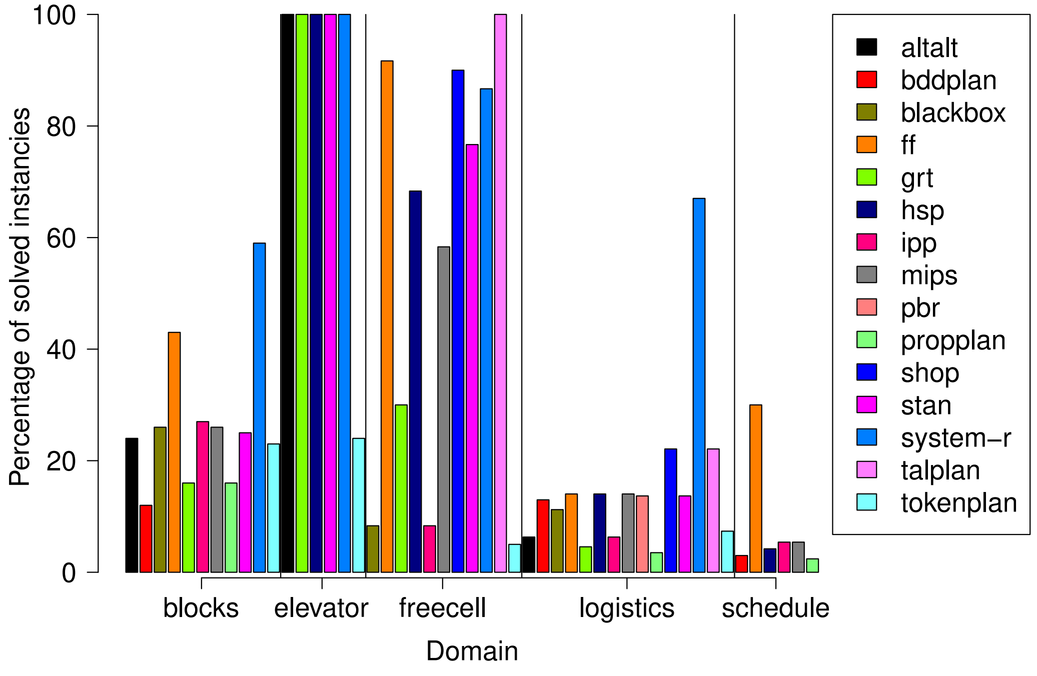


Figure 4.6: Number of solved instances by planner in each domain in IPC 2000.

Chapter 5

Proposed Solution

An instance-difficulty classification of planning problems and a methodology to identify properties that explains that difficulty based on the instance planning graph are proposed in this chapter. The instance-difficulty classification uses the results of the International Planning Competition (IPC) and some size metrics of the planning graphs to classify the instances; these are described first. Later the proposed analysis of the planning graphs to identify properties present that intervene in the classification is described.

5.1 Classification of Instances by Difficulty

According with computational complexity results, if the space required to solve an instance grows, then it is expected that the time required to solved the instance also grows. This result is used to guide the difficulty classification proposed in the present work. From the extracted planning graphs, size properties are used to compare them with the reported time of the planners and characterize the instance difficulty. The increase in the solution time t is compared with the space needed to solve it s, where the time t is expected to increase by some function $f(s) = t$, and that function can be deduced through a regression. Also, from the regression, a prediction band is calculated to include the possible values that fit within.

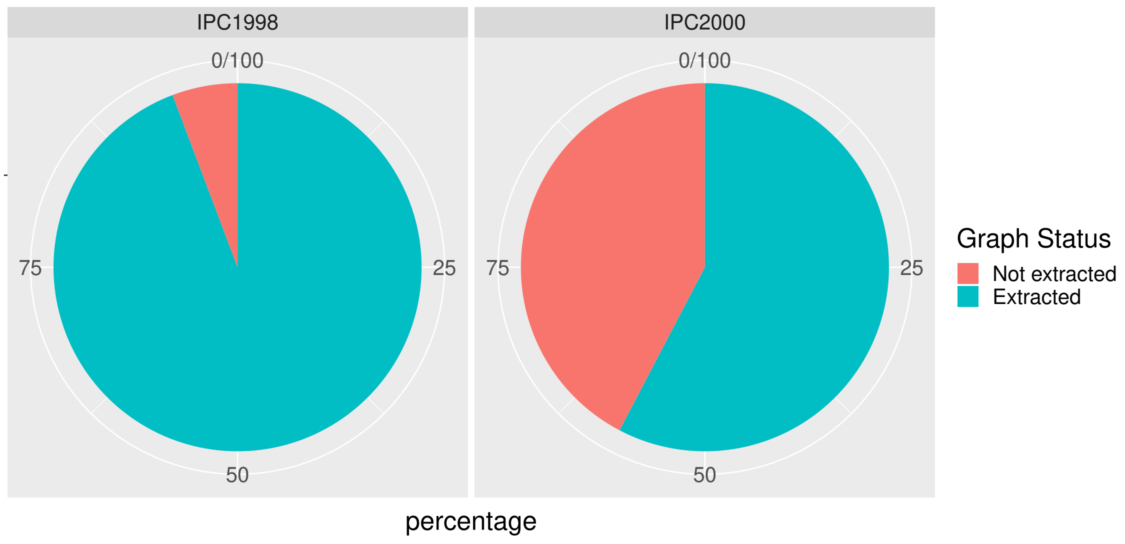


Figure 5.1: Percentage of the instances with a graph extracted by competition.

For this classification, the instances from IPC of 1998 and 2000 are used, and based on the competition results, all the instances are assigned to the following classes: solved, extracted, and parallel. The solved factor determine if the instance were solved by at least one planner. Then, such for all instances are given as input to the Blackbox planner in order to extract their planning graphs, there are three possible subsets: some instances are solved and their planning graph is extracted, while others are not solved but with an incomplete graph extracted, and there are some instances that could not be solved with blackbox and no planning graph were extracted. Finally, for the set of solved instances with a graph fully extracted, there are two kinds of graphs: whether the problem instance has parallel actions in its plan and when it does not has parallel actions. The quantity of instances in each set is shown in Figures 5.1, 5.2, and 5.3.

For the classification, are used the solved instances that have extracter its planning graph, the instances without a graph extracted are not considered in the present work. For instances without parallel actions, the number of levels is equal

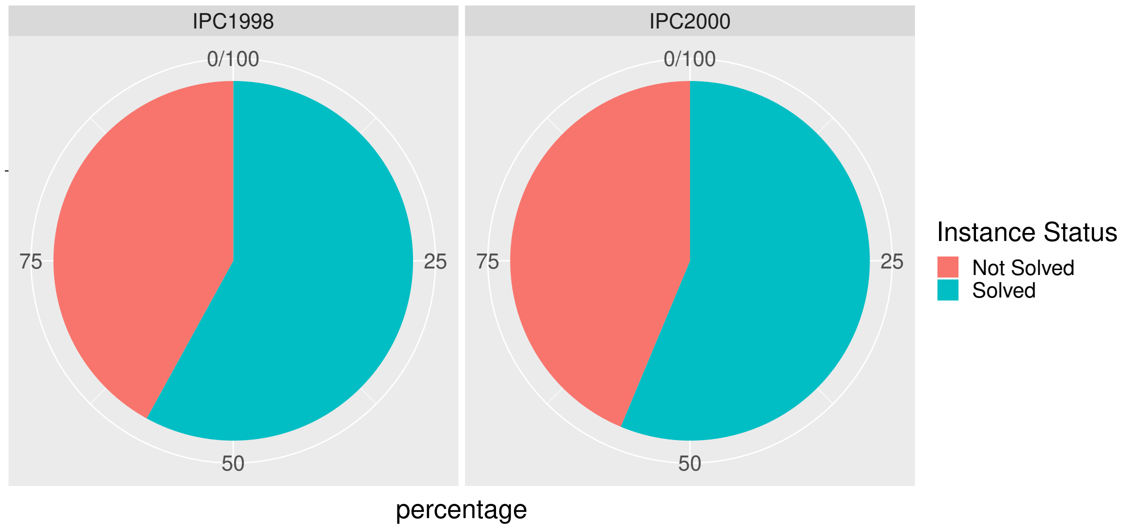


Figure 5.2: Percentage of solved instances by competition.

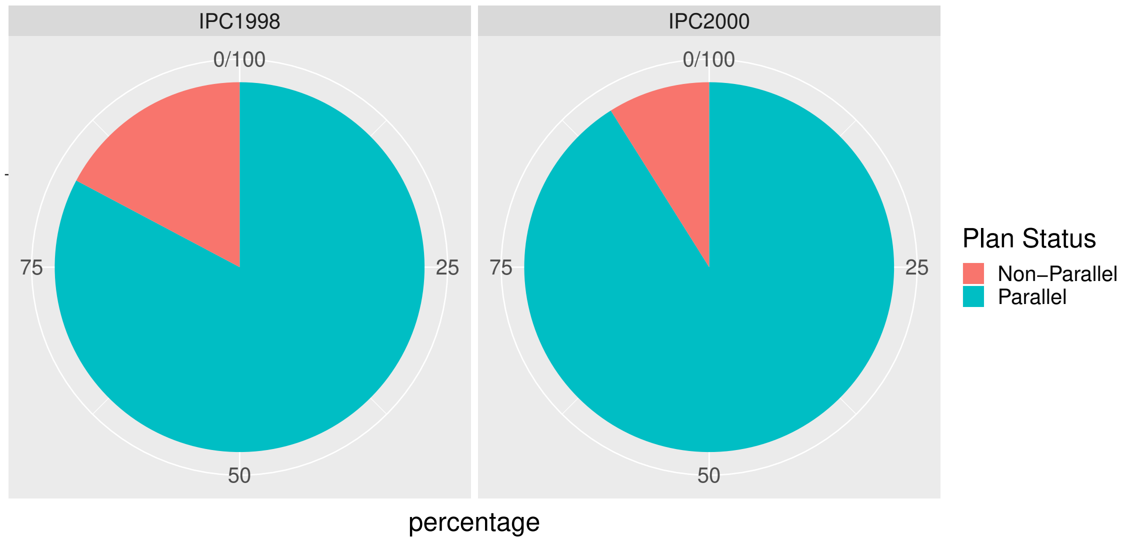
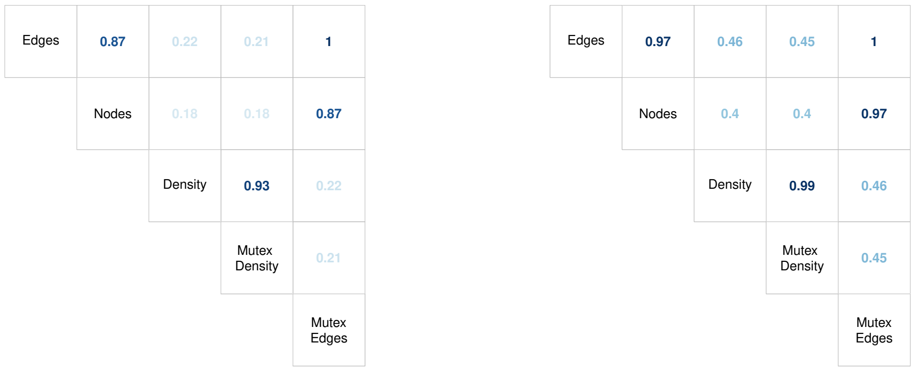


Figure 5.3: Percentage of plans with parallel actions by competition.

to the plan length, while the instances with parallel actions have less graph levels than the plan length. This is differentiated in the classification, because this affects the size of the graphs.

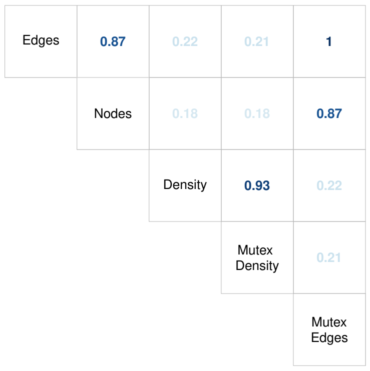
After the graph size measures are selected, the next step is the difficulty classification. The instances with values of time and size that are outside the regression or out its prediction band are the *outliers*. They show a unexpected solved time compared with its size. If the instance is above the prediction band, it is considered a *hard instance* and if the instance in under the prediction band, it is an *easy instance*. The size measure used for leveled graphs are quantity of nodes, edges, graph levels, and density. Also the size of the mutex edges such as total mutex edges and mutex density are included for their relation with the study of insolvable instances [4]. As shown in Figures 5.4a, 5.4b, 5.13, and 5.5b in IPC 1998 the time correlated with all measures for correlation test Kendall and Spearman, but for IPC 2000 there was low value of correlation between time and factors compared with IPC 1998, specially for density measures that do not correlate with a p-value greater than 0.05.



(a) Kendall Correlation test

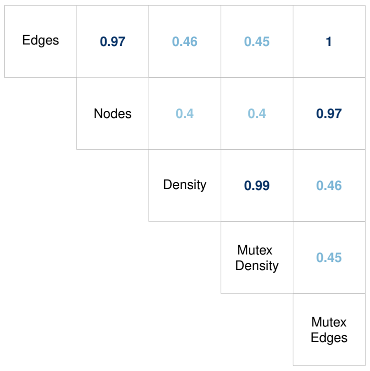
(b) Spearman Correlation test

Figure 5.4: Spearman and Kendall tests of the graph properties extracted from IPC 1998 instances, the numbers in the cells are the ρ values.



Edges	0.87	0.22	0.21	1
	Nodes	0.18	0.18	0.87
		Density	0.93	0.22
			Mutex Density	0.21
				Mutex Edges

(a) Kendall Correlation test



Edges	0.97	0.46	0.45	1
	Nodes	0.4	0.4	0.97
		Density	0.99	0.46
			Mutex Density	0.45
				Mutex Edges

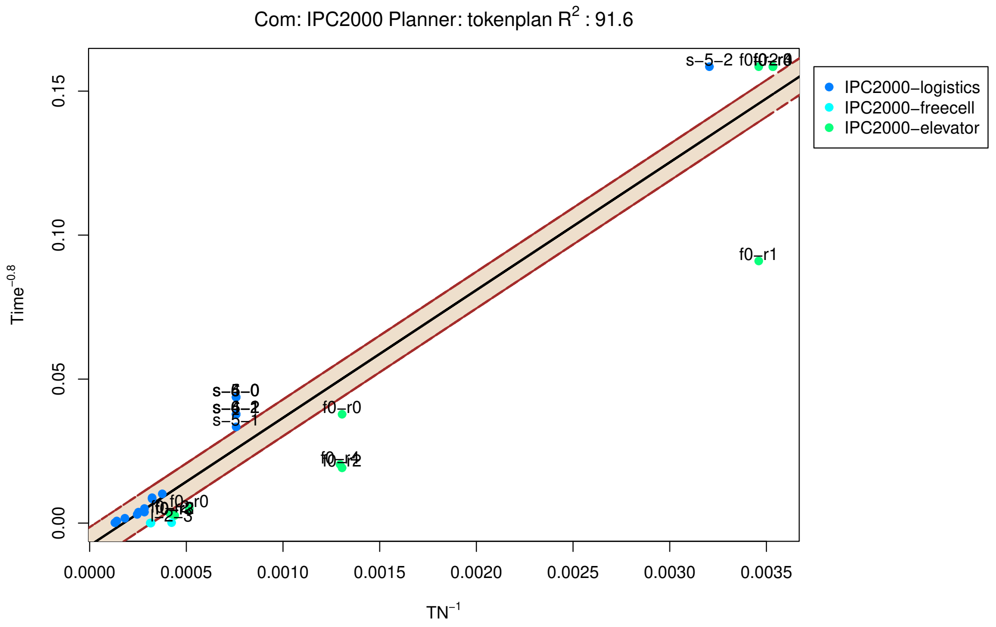
(b) Spearman Correlation test

Figure 5.5: Spearman and Kendall tests of the graph properties extracted from IPC 1998 instances, the numbers in the cells are the ρ values.

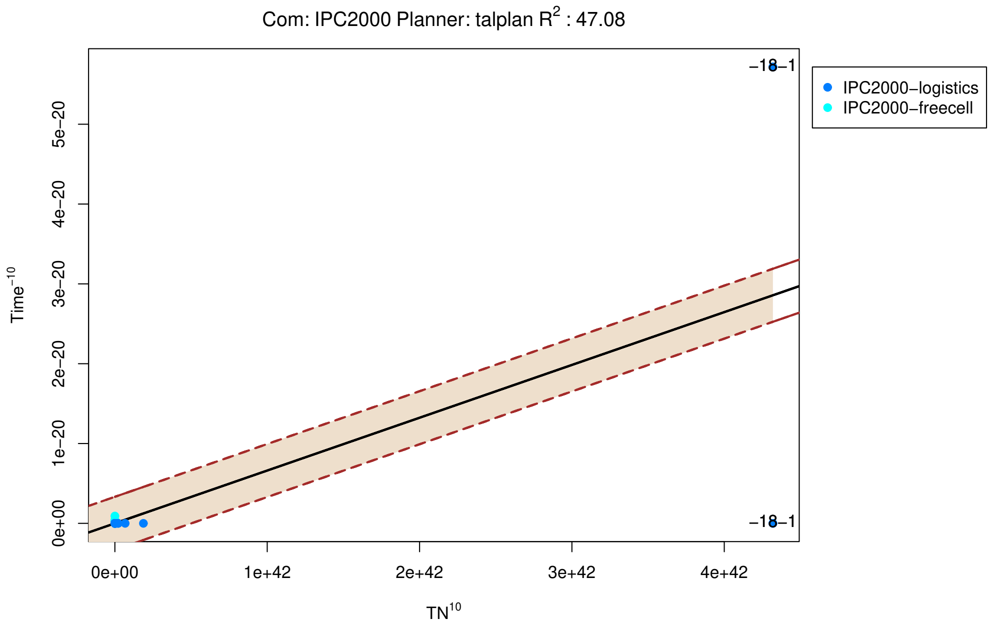
Each regression line is transformed using the Tukey ladder, to choose the transformation that maximize the R^2 value, and the regressions are filtered using the ones that have a R^2 value between 85% and 95%. Some examples are presented in Figures 5.6, 5.7, 5.8, and 5.9 where it is appreciated the regression bands and the outliers.

With the instance classifications there are 355 linear regression for each planner in each competition: 245 in the parallel domain as shown in Figure 5.10a, and 110 in non-parallel domains as shown in Figure 5.10b. Not all regressions are significant and some register a high quantity of outliers (low R^2 value), so the regressions with R^2 values between 85% and 98% are chosen. With these values, 45 regressions from the parallel domain and 20 regressions of the non parallel domain are used to perform the classification.

Using the filtered regressions, the difficult classification of each instance is defined through polls that use the classification of each model by domain and planner to assign a difficulty vote for each instance. There are two polls, easy and hard poll. The votes for the polls are calculated dividing the number of linear models ℓ where an instance i is classified as an easy or hard outlier $e_{i\ell}$ between the total of the n linear models considered in that domain. This score assignation have the issue that some problems could have a tie in the difficulty scores as seen in Figure 5.11. To

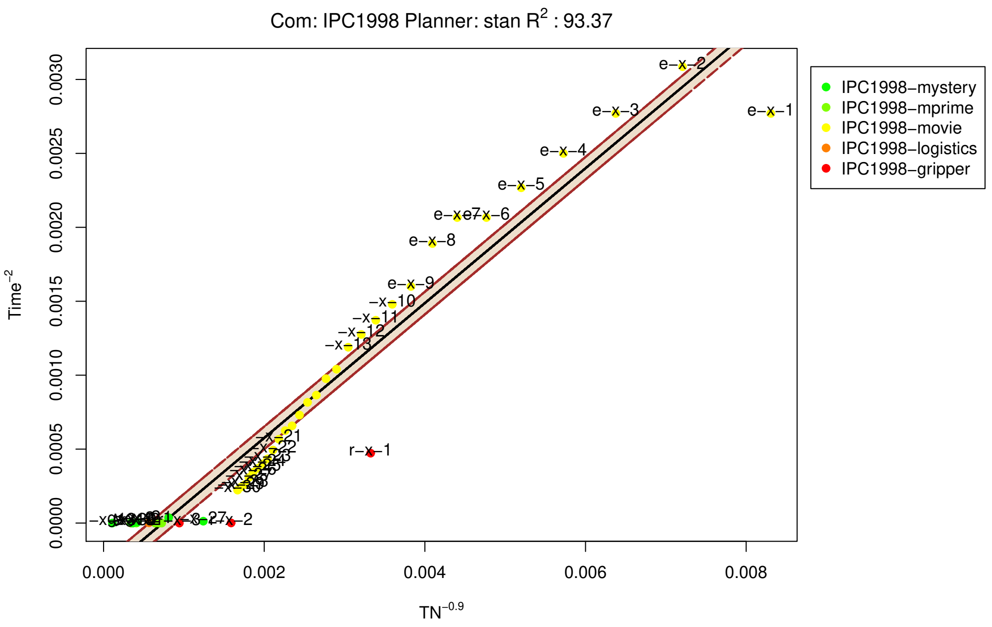


(a) parallel domain

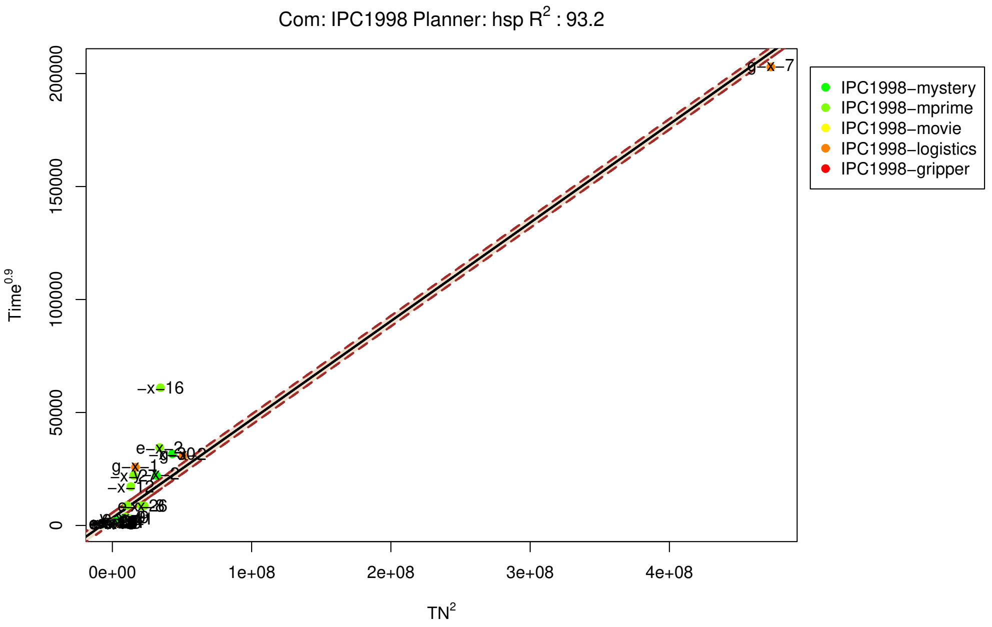


(b) Parallel domain

Figure 5.6: Regression for time and total nodes of parallel instances of IPC 2000

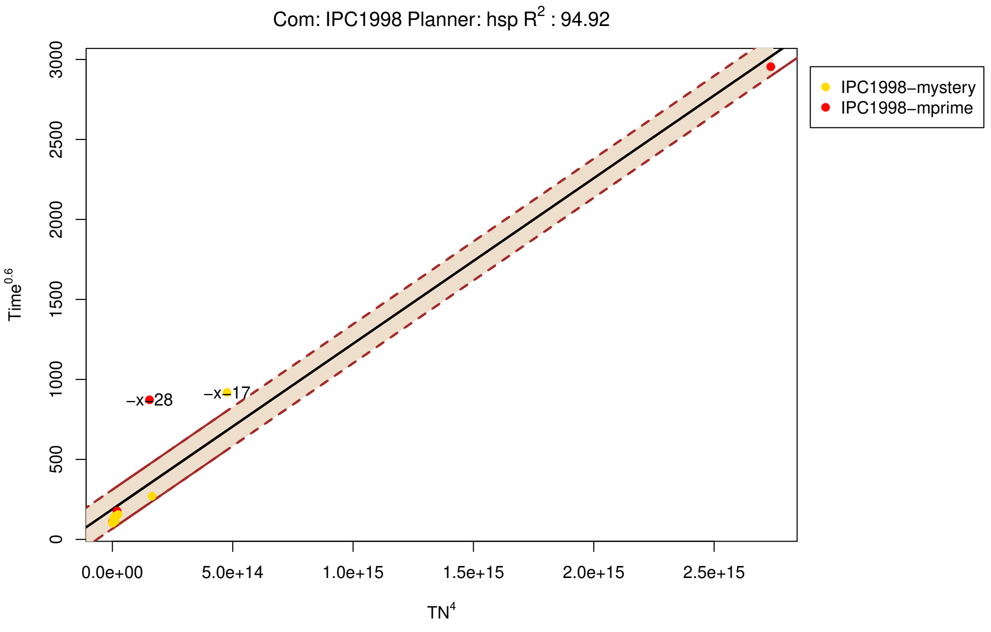


(a) Parallel domain

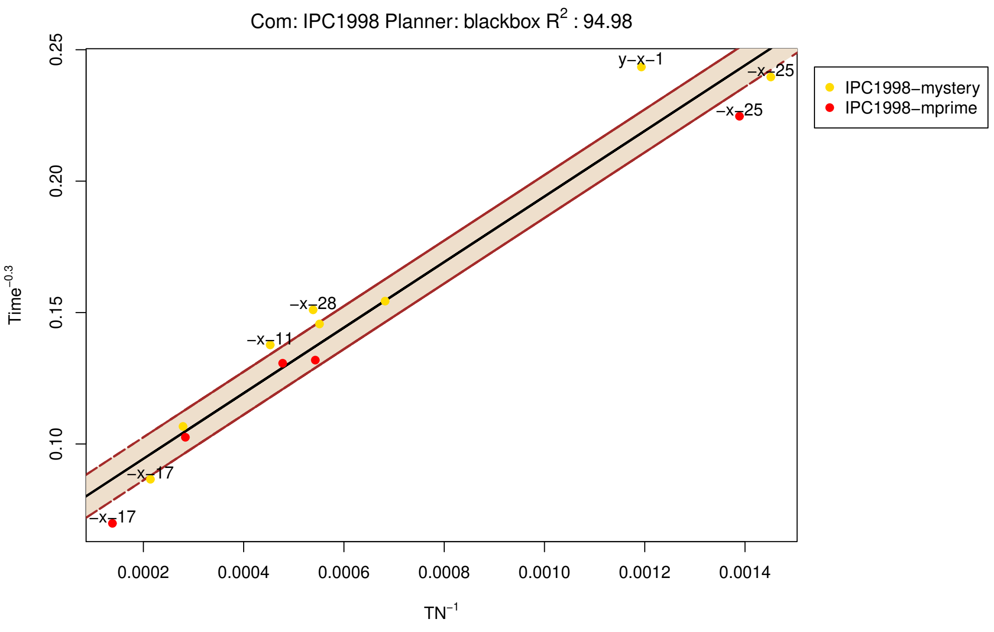


(b) Parallel domain

Figure 5.7: Regression for time and total nodes of parallel instances of IPC 1998

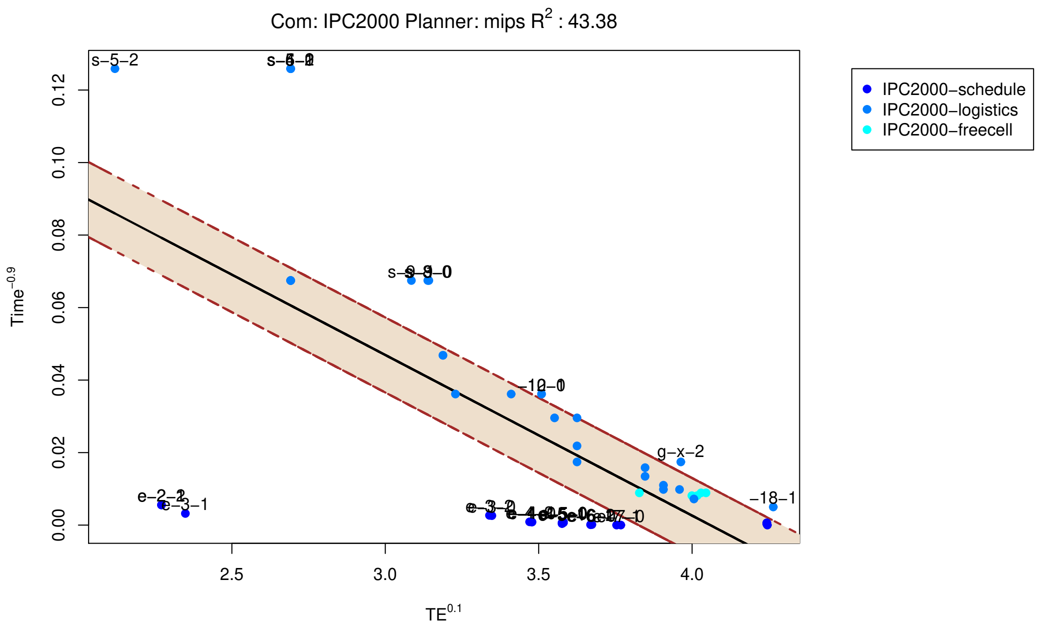


(a) Parallel domain

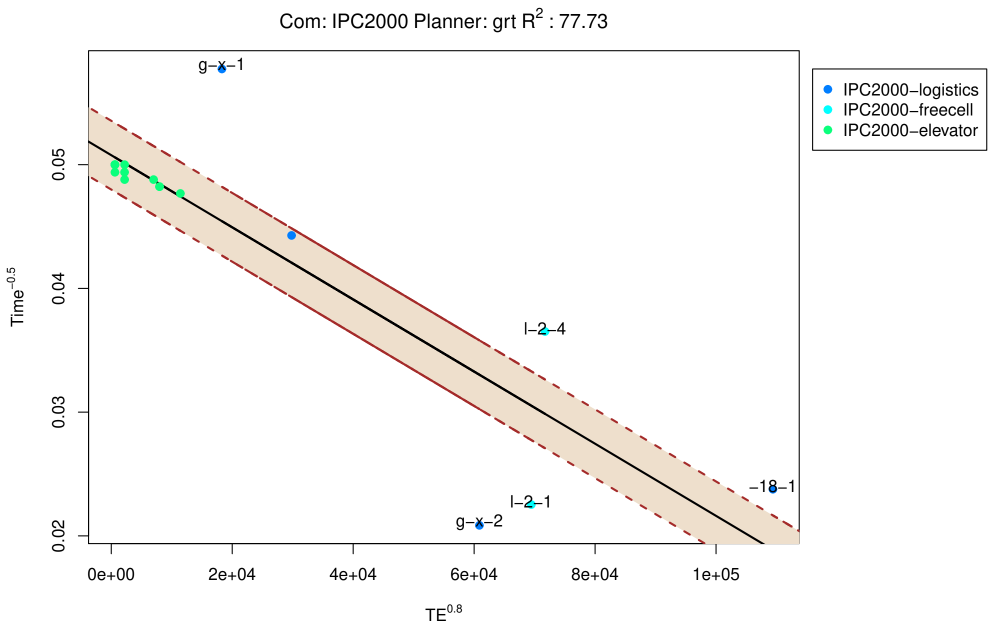


(b) Parallel domain

Figure 5.8: Regression for time and total nodes of non-parallel instances of IPC 1998

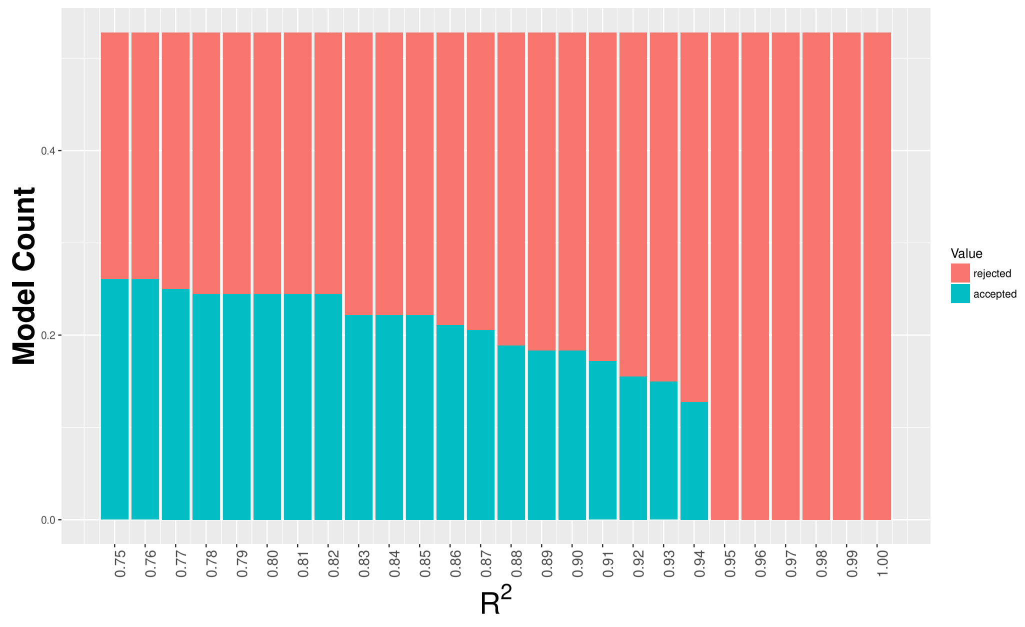


(a) parallel domain

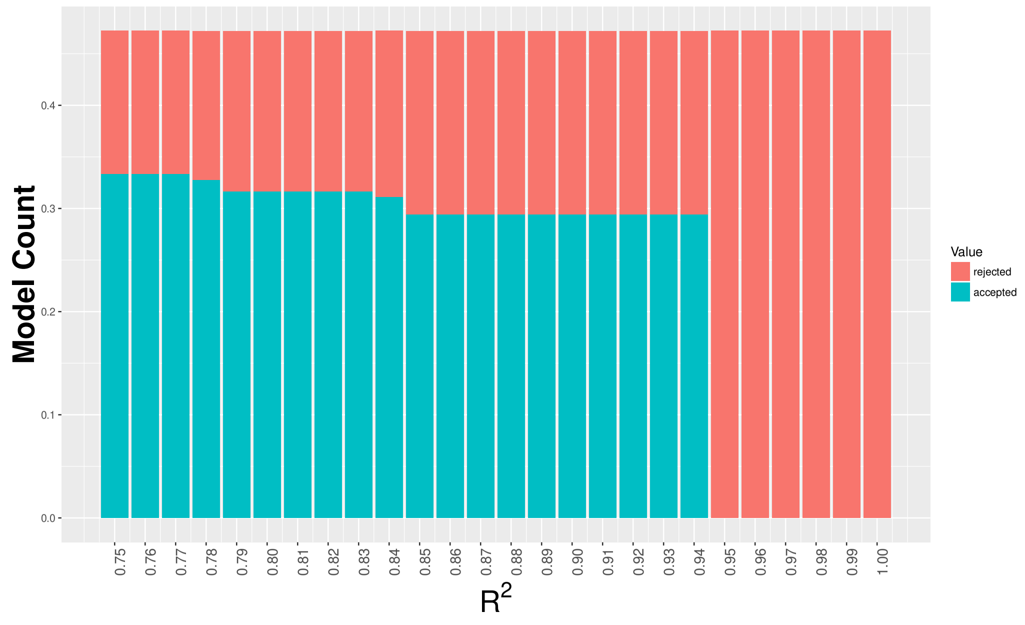


(b) Parallel domain

Figure 5.9: Regression for time and total nodes of parallel instances of IPC 2000



(a) Accepted linear models above the fixed R^2 value from the parallel domains



(b) Accepted linear models above the fixed R^2 value from the non parallel domains

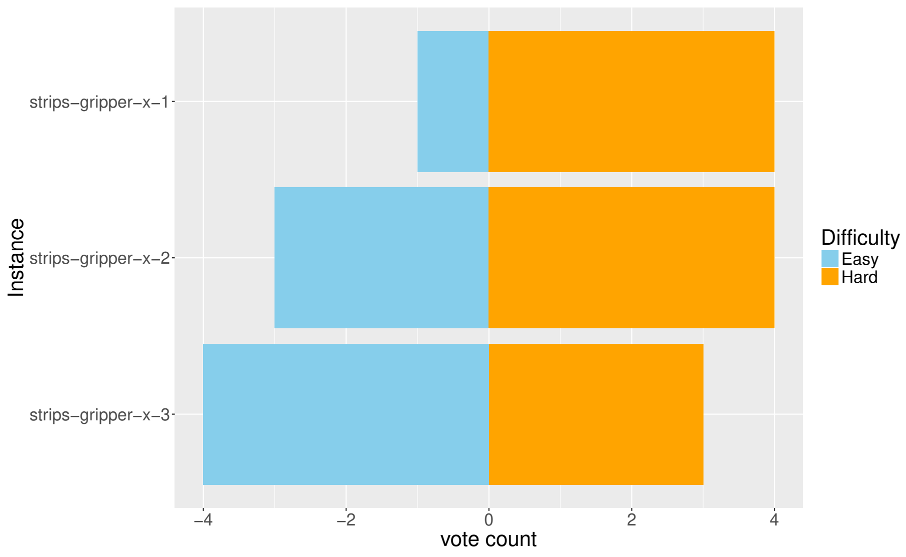


Figure 5.11: Score comparison of classified instances.

avoid this and to consider the kind of outlier, the distance between the instance result and the prediction band is used to give more importance to the far outliers. The *distance score* is calculated dividing the distance $D(e_{i\ell})$ of the instance to the prediction band of the linear model ℓ between the sum of all instance distances of the linear model $\sum_{j=0}^{m} D(e_{ij})$, in this way, far outliers have a high distance value and near outliers have a low distance value. So, the multiplication of the distance score with the difficulty score are used to calculate the classification as seen in Figure 5.12:

$$\begin{aligned} ES_i &= \frac{1}{n}\sum_{l=0}^{n} \frac{e_{i\ell}D(e_{i\ell})}{\sum_{j=0}^{m} D(e_{j\ell})} \\ HS_i &= \frac{1}{n}\sum_{l=0}^{n} \frac{h_{i\ell}D(h_{i\ell})}{\sum_{j=0}^{m} D(h_{j\ell})}. \end{aligned} \tag{5.1}$$

This methodology can classify an instance in four different classes: as *Easy* instance if its easy score is greater than zero while its hard score is zero or its easy score is greater than the 50% of its hard score, as a *Hard* instance if its hard score is greater than zero while its easy score is zero or its hard score is grater than the 50% of its easy score, as a *fit* instance if its hard and easy score are zero or both scores are less than the value of 0.001, and as *Not Classified* if its easy and hard score are less than the 50% of the other.

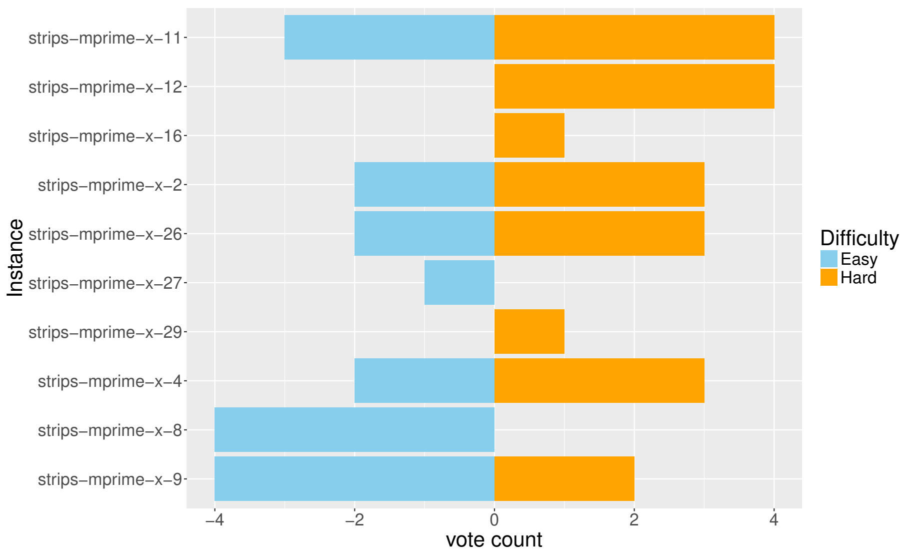


Figure 5.12: Score comparison of non-classified instances.

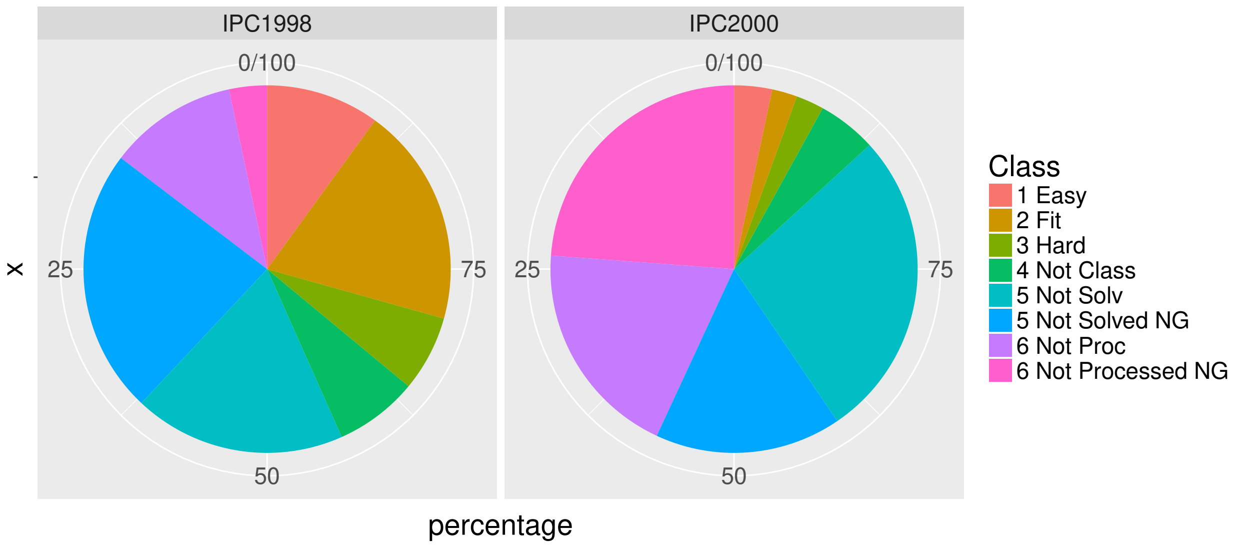


Figure 5.13: Percentages of classified instances by competition.

5.2 Selection of Metrics

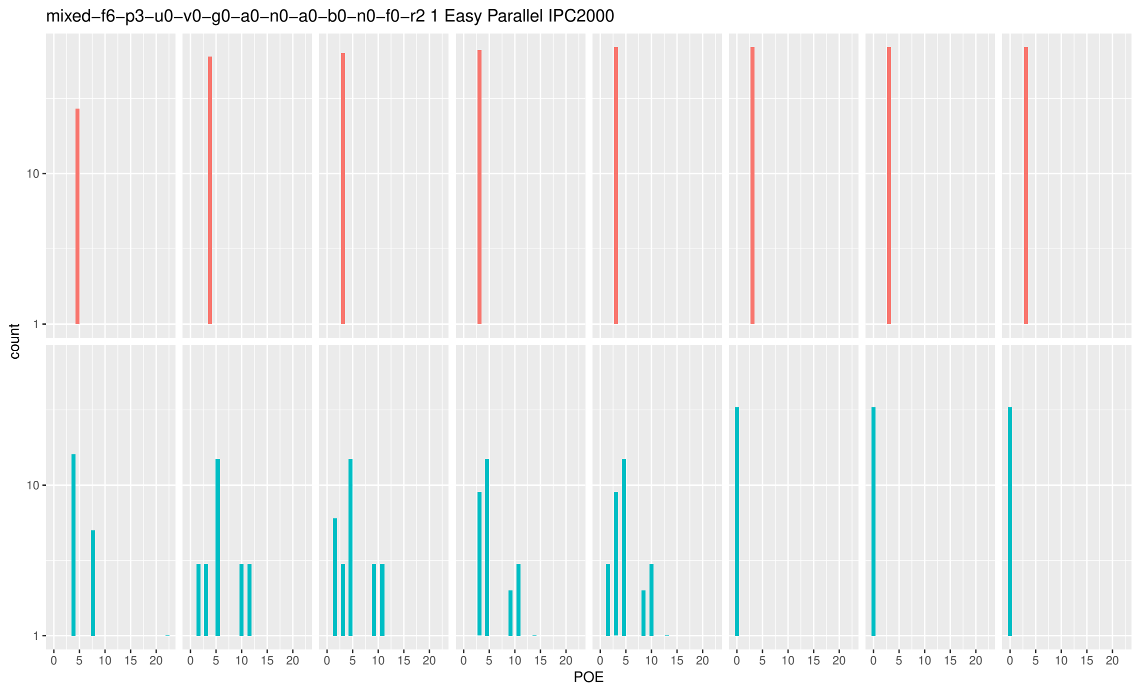
The metrics selection considers properties that measure the connectivity of the graph through its level. In the graph plan structure, actions and fact nodes have mutually exclusive edges (mutex edges) with other nodes of same type in its level. The fact nodes have delete and out edges with the action nodes of its level, and the action nodes have delete and out edges with the fact nodes of its next level, so the first property selected is the percentage of mutex nodes (PME). It measures for each node the quantity of mutex nodes over the total number of nodes in a level. The second one is the percentage of nodes that its out edges connects (POE): it measures for each node, the quantity of nodes that it connects over the total number of nodes of the level. The third one is similar than the previous but with the delete edges (PDE) of the node. This metrics are chosen because are node based and they allow to study its distribution over the levels of the graph. Level and graph-base metrics are not considered.

Now that the metrics are chosen, over 600 graphs are measured for 10 million nodes. Initially this was executed by a python script, but it used lot of resources to store graphs structures in memory and calculate the metrics, so, the JSON files with the graph structure were stored in MongoDB, a non-relational database. This allows the use of map-reduce functions that analyze the nodes and that reduce the time and memory needed to calculate the metrics, because the files are not fully loaded in memory, instead they are read from disk and parallel processing is used to calculate the measures.

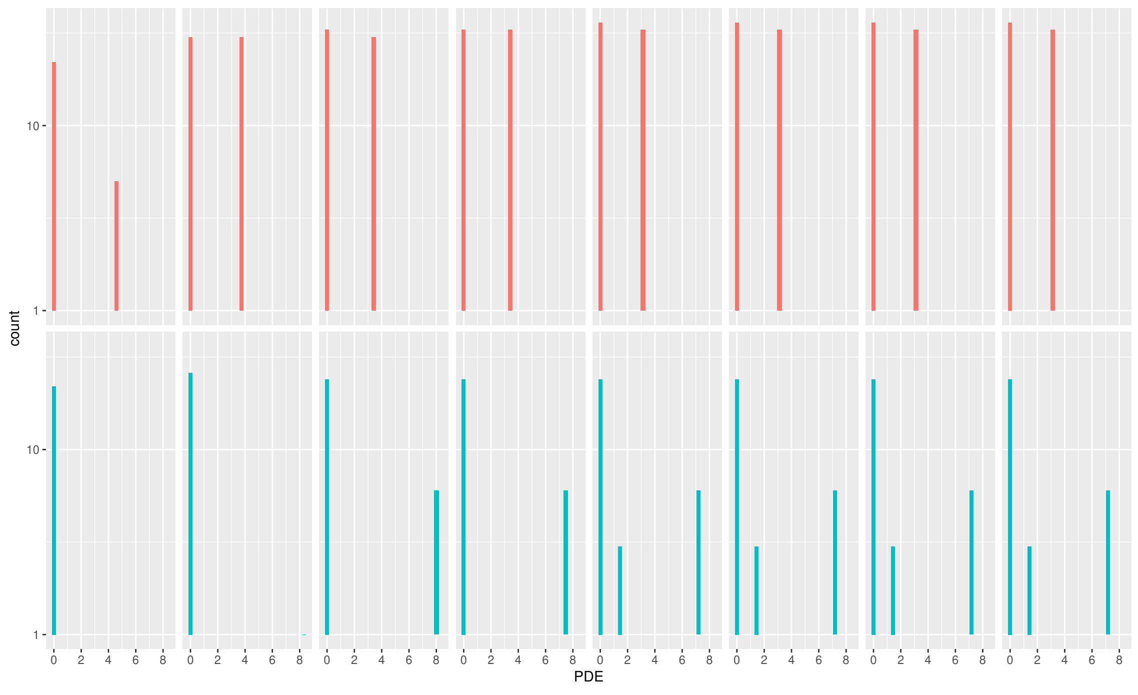
The proposed metrics are focused on the distribution of the values over the nodes. These distributions are analyzed to identify if some distribution parameter is present or absent in any of the classifications. First the distribution of node measurements is examinated by level and by node type: the red values represent the action nodes and the blue values are from fact nodes, the horizontal axis represents the value of percentage of edges of the type analyzed and the vertical axis represents the quantity of nodes in the level; the distributions on the left are from the first levels of the graphs and the distributions on the right are from the last levels of the graph; finally at bottom is the distribution of nodes by level.

Figures 5.14 and 5.15 show the measurements of a graph of the *elevator* domain form the 2000 IPC that is classified as an easy instance with parallel actions, this graph have 8 levels and its plan is of 16 steps. The POE measurement of the action nodes have a constant value in all levels of 5% affecting the same percentage of fact nodes. For the fact nodes, the values are spread between 3% and 13% for the first levels but after the 4 level the values are constant near 1%. The PDE measurements of the action nodes are near constant in 0% and 4%, meanwhile the values for the fact nodes are in 0, 1.5 and 7%. The PME measurements the action nodes have values at 0% and around 90% and the fact nodes have values of 0, 25, 60, and 80%. Finally the growth rate of the graph through the levels is show in the last image, the maximum number of nodes is 100 that is reached in level 4. This easy instance graph show low values from POE, PDE for action and fact nodes, and for PME the action and fact nodes have their values separated on two values; also this graph has a fast growth rate.

Figures 5.16 and 5.17 show the measurements of a graph of the *schedule* domain form the 2000 IPC that is classified as an regular instance with parallel actions, this graph have 3 levels and its plan is of steps. The POE measurement of the action nodes are near 0%. For the fact nodes the values are spread between 3% and 10%. The PDE measurements of the action nodes are between in 0% and 1% meanwhile the values for the fact nodes are spread between 0% and 3.1%. The PME measurements the action nodes have their values distributed, starting between 0% and 25% and finishing between 0% and 40%, and the fact nodes have their values at first level near 0% and then get distributed between 0% and 20%. Finally, the growth rate of the graph has a maximum number of nodes of 2,500 but this time its growth by level is faster than the easy case. The instance classified as regular show more nodes by

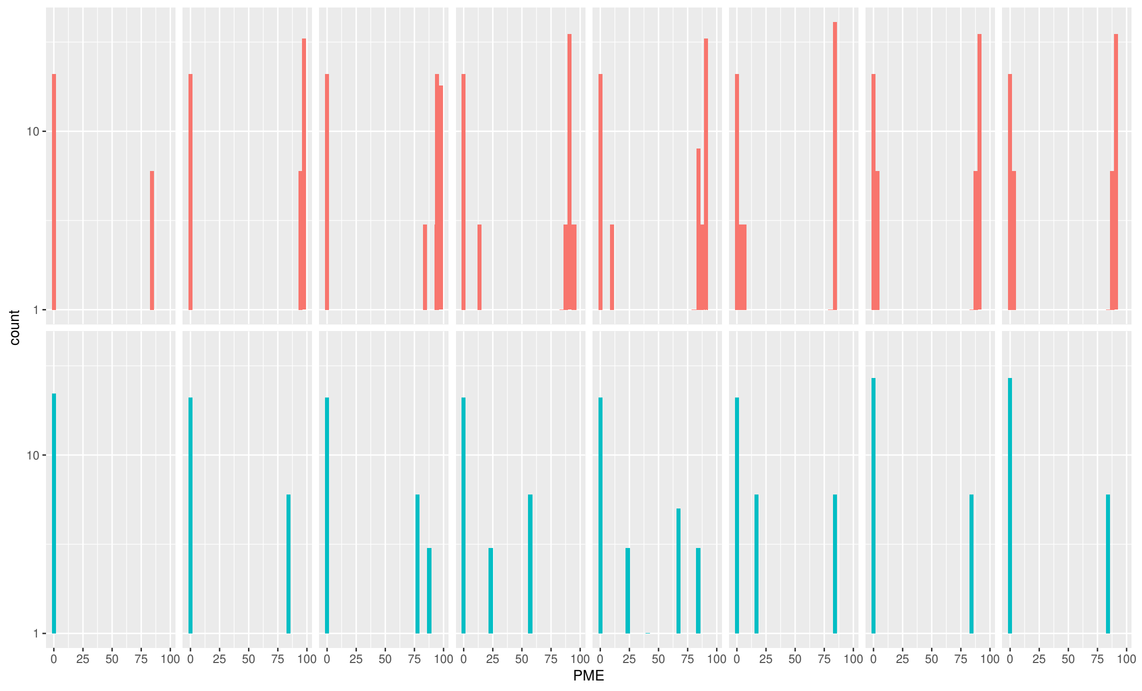


(a) Distribution of POE in instance *elevator 2*.

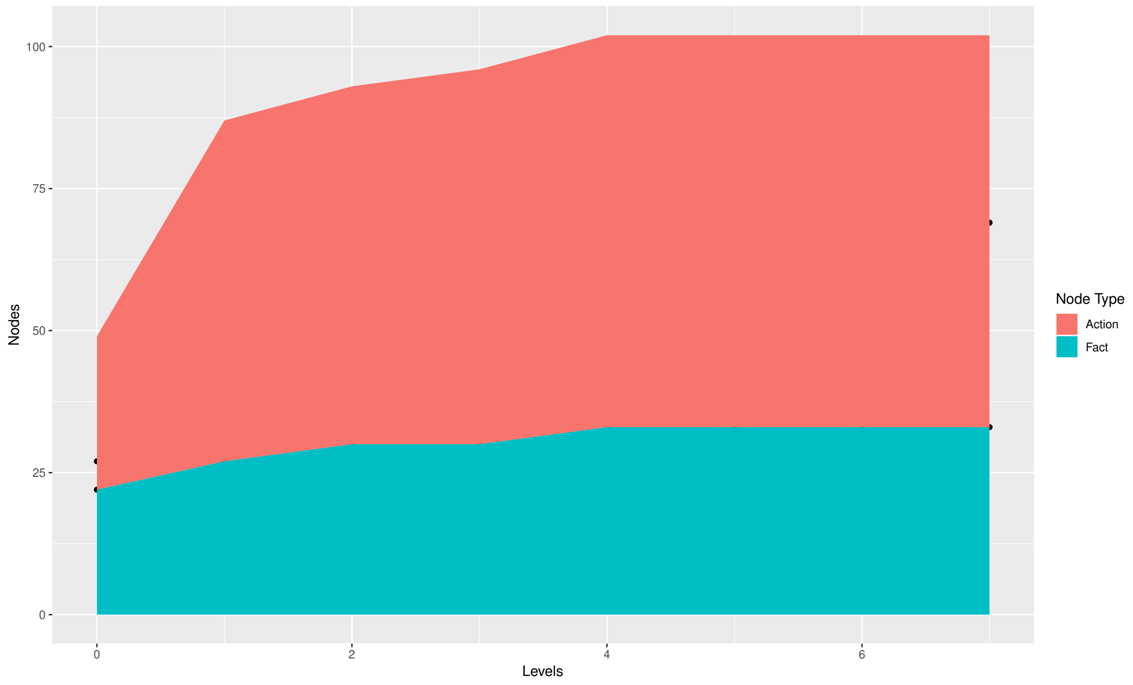


(b) Distribution of PDE in instance *elevator 2*.

Figure 5.14: Exploratory analysis of instance *elevator 2*.



(a) Distribution of PME in instance *elevator 2*.



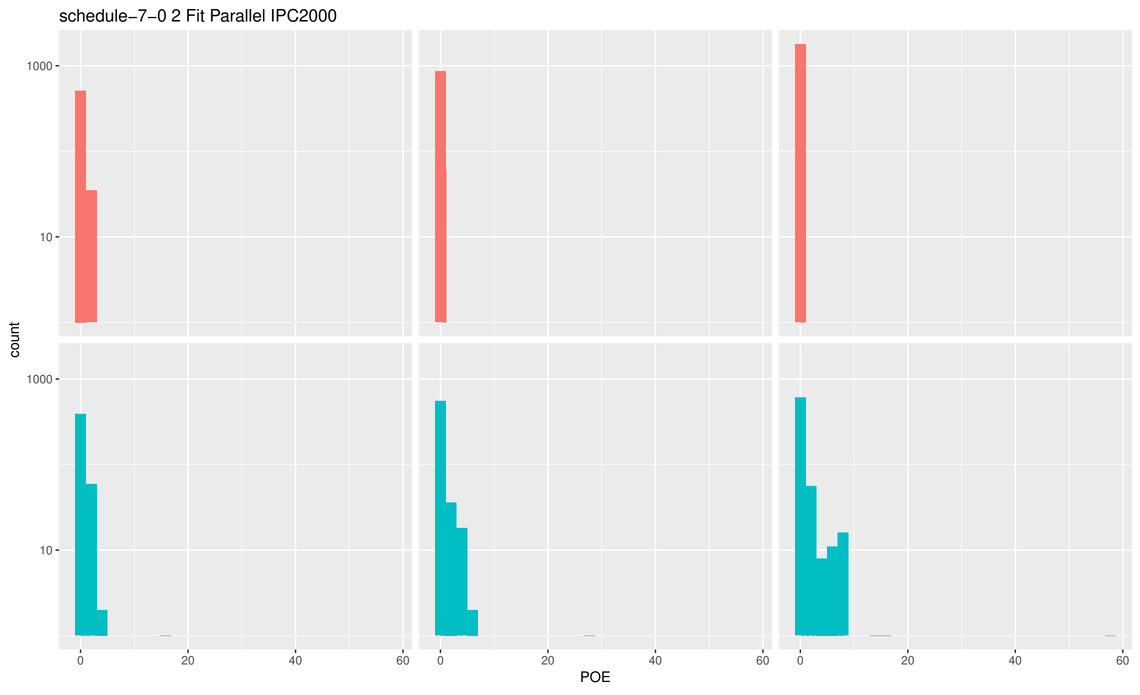
(b) Node growth by level in instance *elevator 2*.

Figure 5.15: Exploratory analysis of instance *elevator 2*.

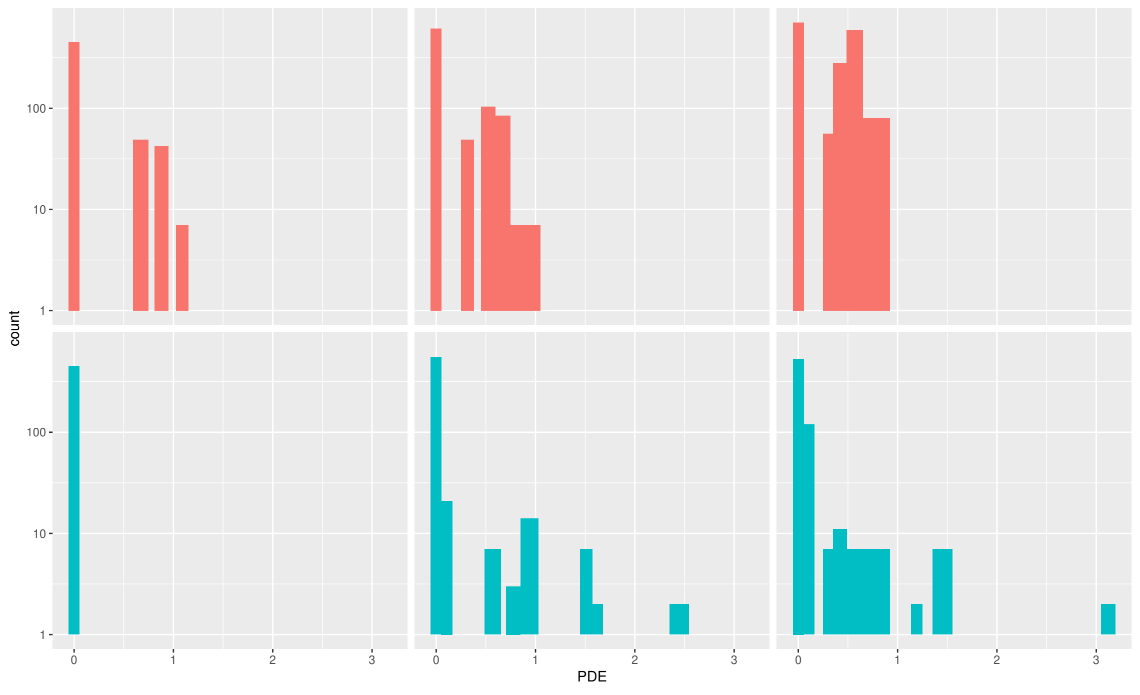
level in POE, PDE, PME for actions and fact nodes, and for PME the action and fact nodes have distributed their values also this graph has a fast growth rate.

Figures 5.18 and 5.19 show the measurements of a graph of the *mystery* domain form the 1998 IPC that is classified as an Hard instance with parallel actions, this graph have 5 levels and its plan is of 16 steps. The POE measurement of the action nodes are near 0%. For the fact nodes the values are spread between 0% and 20%. The PDE measurements of the action nodes are also in 0% and 1% meanwhile the values for the fact nodes are spread between 0% and 3%. The PME measurements for the action nodes have its values also distributed, starting between 0% and 25% and finishing between 0% and 60% and the fact nodes have its values at first level near 0% and then get distributed between 0% and 20%. Finally in the growth rate image of the graph the maximum number of nodes is 2600 but this time its growth by level is bigger than the easy case. The graph for hard instance show more nodes by level in POE, PDE, PME for actions and fact nodes, and for PME the action and fact nodes have distributed their values also this graph have a fast growth rate for the action nodes and the fact node remain almost the same.

Figures 5.20 and 5.21 show the measurements of a graph from the *schedule* domain form the 2000 IPC that was not solved. This graph has three levels and its plan is of 9 steps. The POE measurement of the action nodes are near 0%. For the fact nodes the values are spread between 0% and 15%. The PDE measurements of the action nodes are also in 0% and 1.5% meanwhile the values for the fact nodes are spread between 0% and 3%. The PME measurements for the actions nodes have their values also distributed, starting between 0% and 25% and finishing between 0% and 60%. The fact nodes have their values at first level near 0% and then get distributed between 0% and 30%. Finally in the growth rate of the graph the maximum number of nodes is 1,700, its growth by level is similar than the regular case. This graph show similar quantity of nodes by level in POE, PDE, PME for actions and fact nodes than the regular case. But the distribution of the PME action and fact nodes are distributed in large intervals. Also this graph has a faster growth

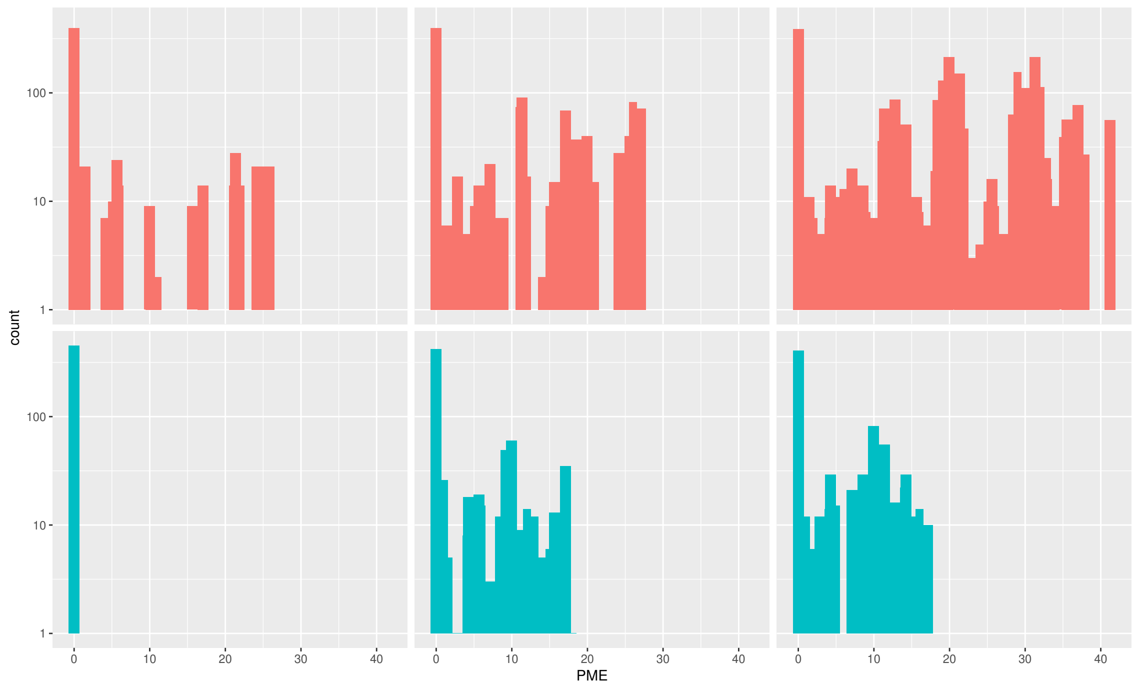


(a) Distribution of POE in instance *schedule-7-0*.

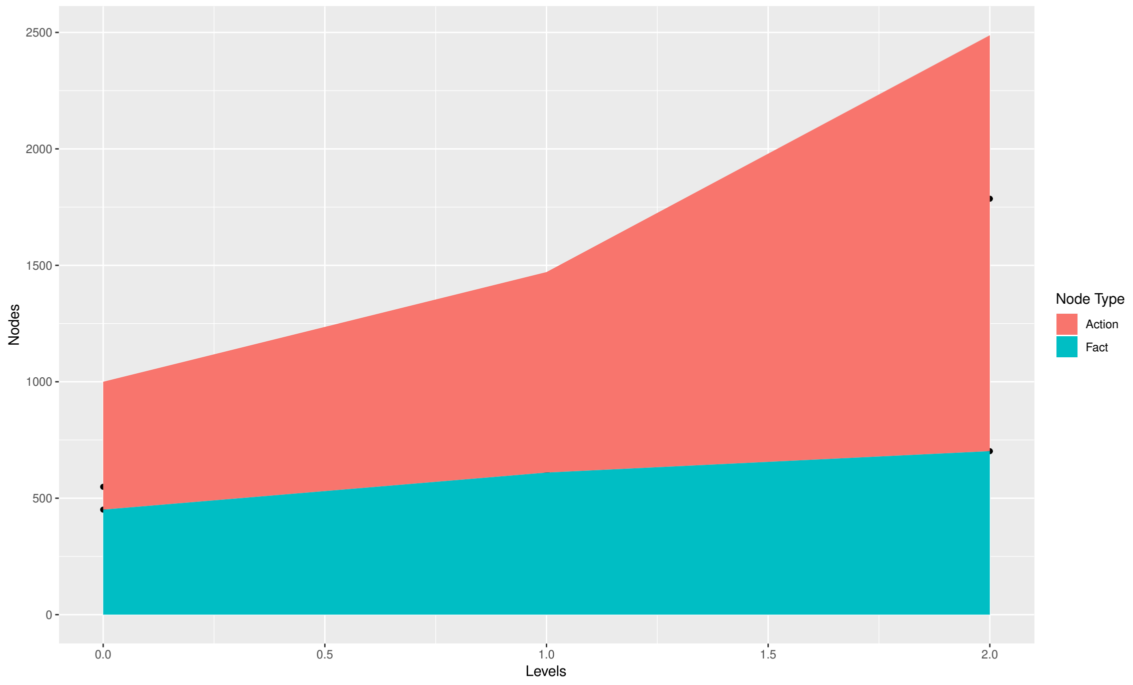


(b) Distribution of PDE in instance *schedule-7-0*.

Figure 5.16: Exploratory analysis of instance *schedule-7-0*.

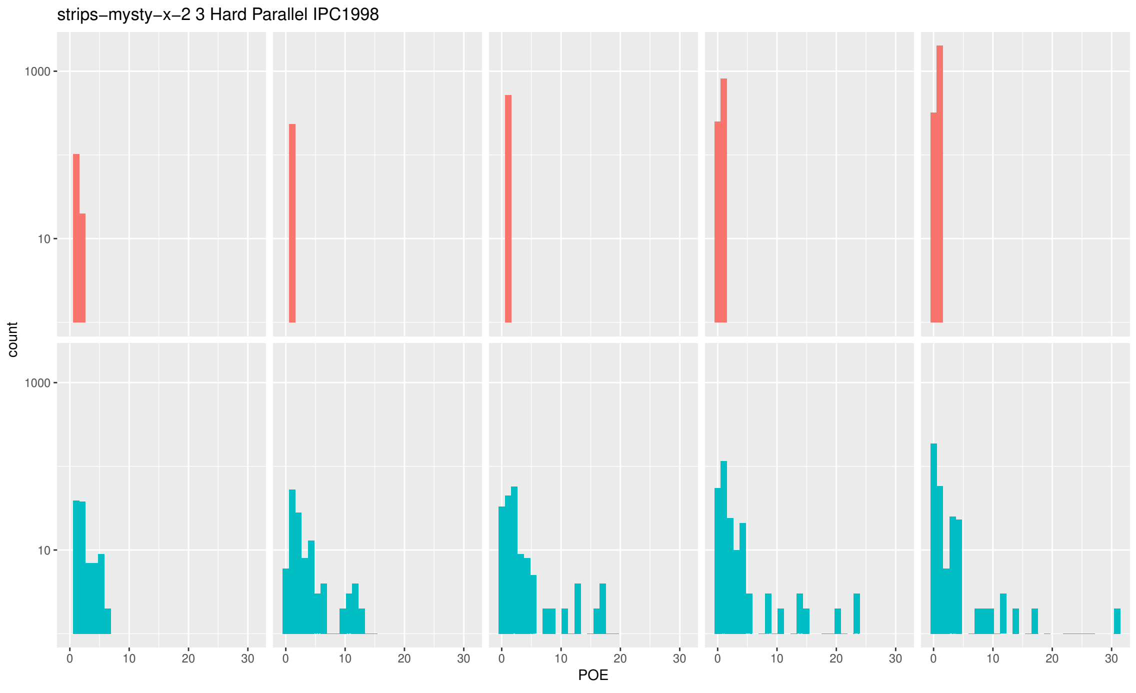


(a) Distribution of PME in instance *schedule-7-0*.

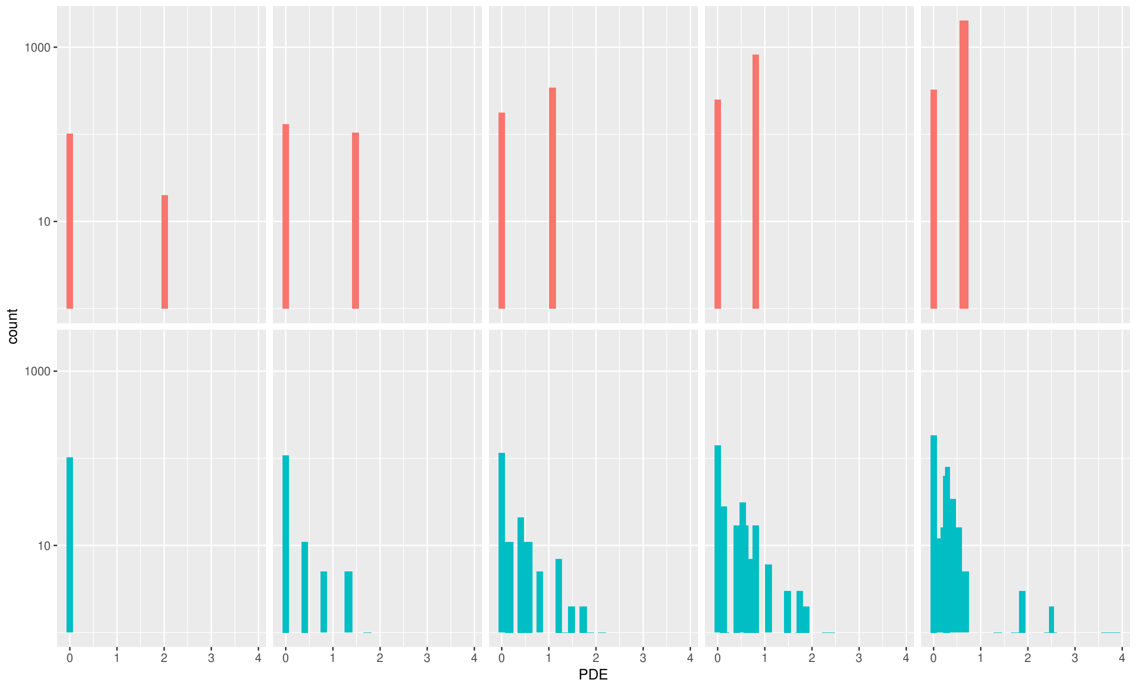


(b) Node growth by level in instance *schedule-7-0*.

Figure 5.17: Exploratory analysis of instance *schedule-7-0*.

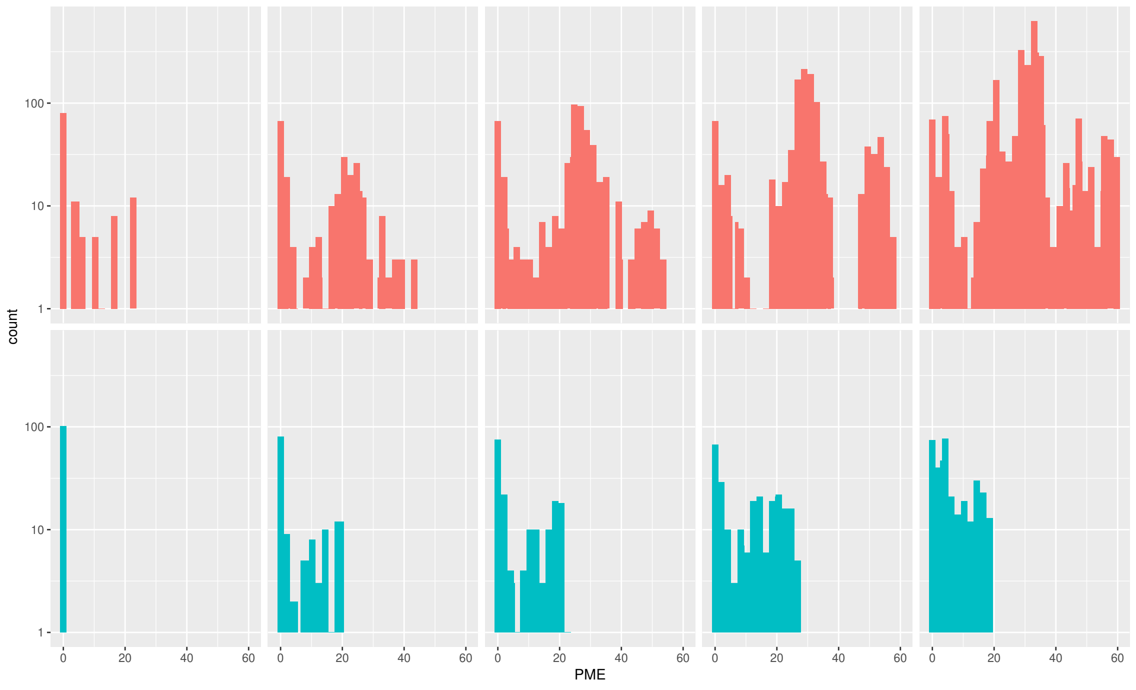


(a) Distribution of POE in instance *mystery 2*.

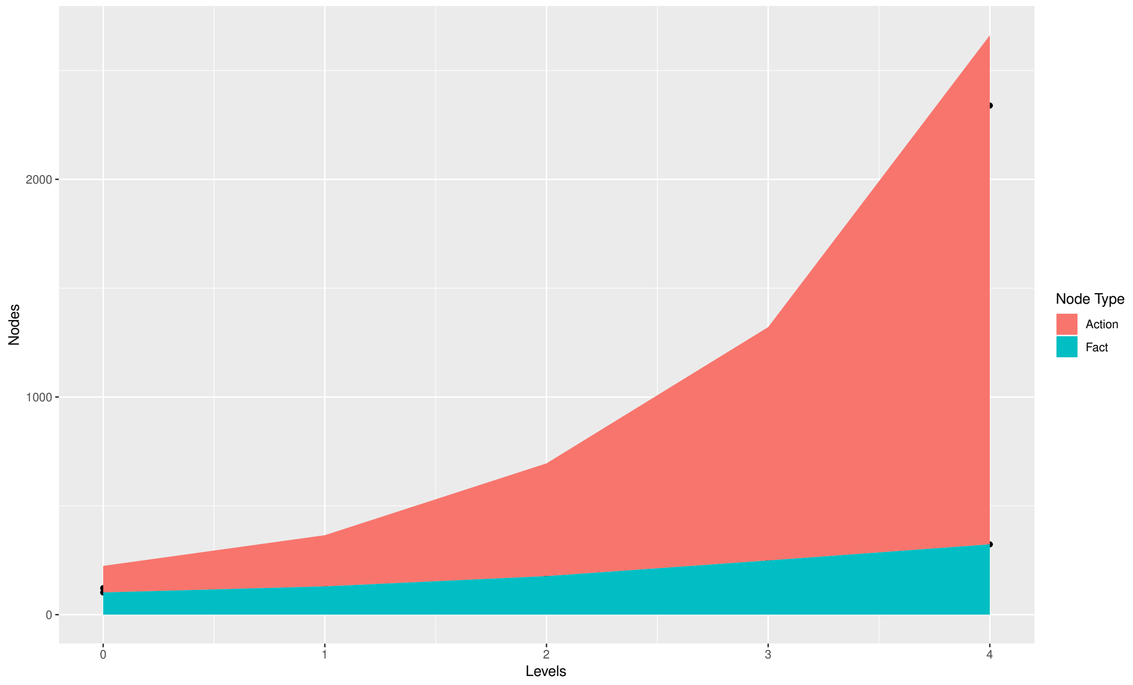


(b) Distribution of PDE in instance *mystery 2*.

Figure 5.18: Exploratory analysis of instance *mystery 2*.



(a) Distribution of PME in instance *mystery 2*.



(b) Node growth by level in instance *mystery 2*.

Figure 5.19: Exploratory analysis of instance *mystery 2*.

rate for the action nodes and the fact node remain almost the same.

After the exploratory analysis, some properties of the measurements distributions could be tested and is verified if they have some impact in the difficulty classification for the solved instances, or with the unsolved instances. These properties are crossed with each type of node among the classifications and divided based on the parallelism type of the graph. The properties to test in each measurement are:

Mode: on the last level of the graph, the mode of the distribution of the percentage of edges of the nodes.

Mode Percentage of Nodes: the percentage of nodes on the last level that have the percentage of edges equal to the mode. This metric can reflect the modality of the distribution: a mode with high percentage of nodes is uni-modal and a mode with low percentage of nodes is more likely to be multi-modal.

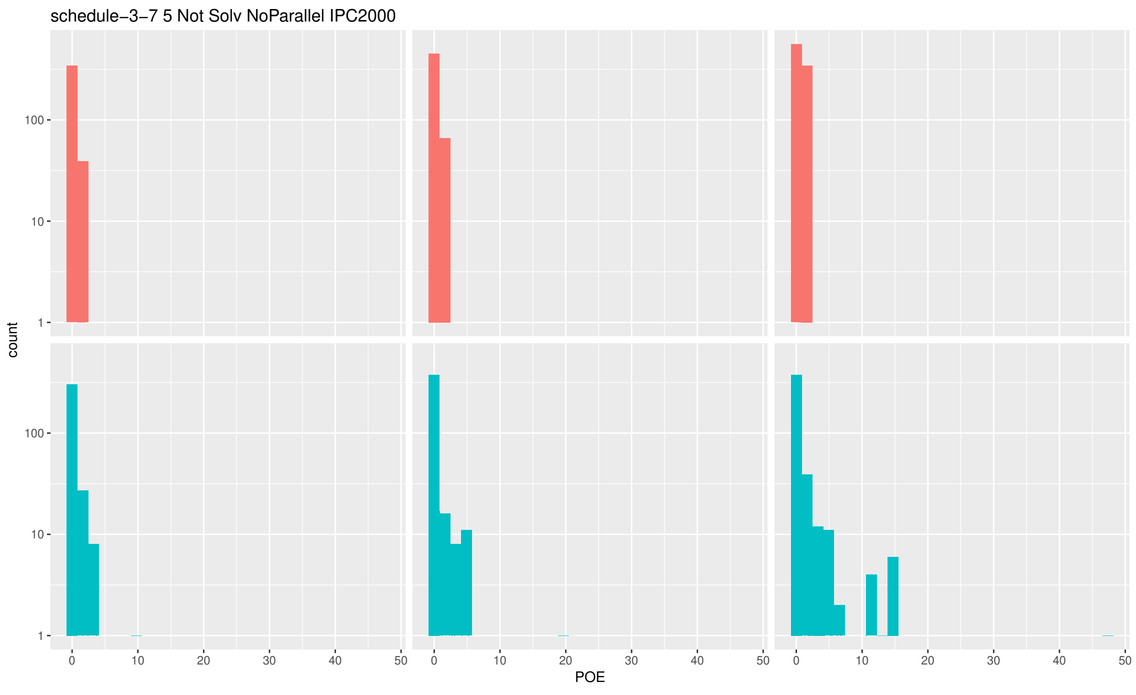
Total Nodes: the quantity of nodes in the graph.

Total Nodes in Last Level: the quantity of nodes in the last level of the graph.

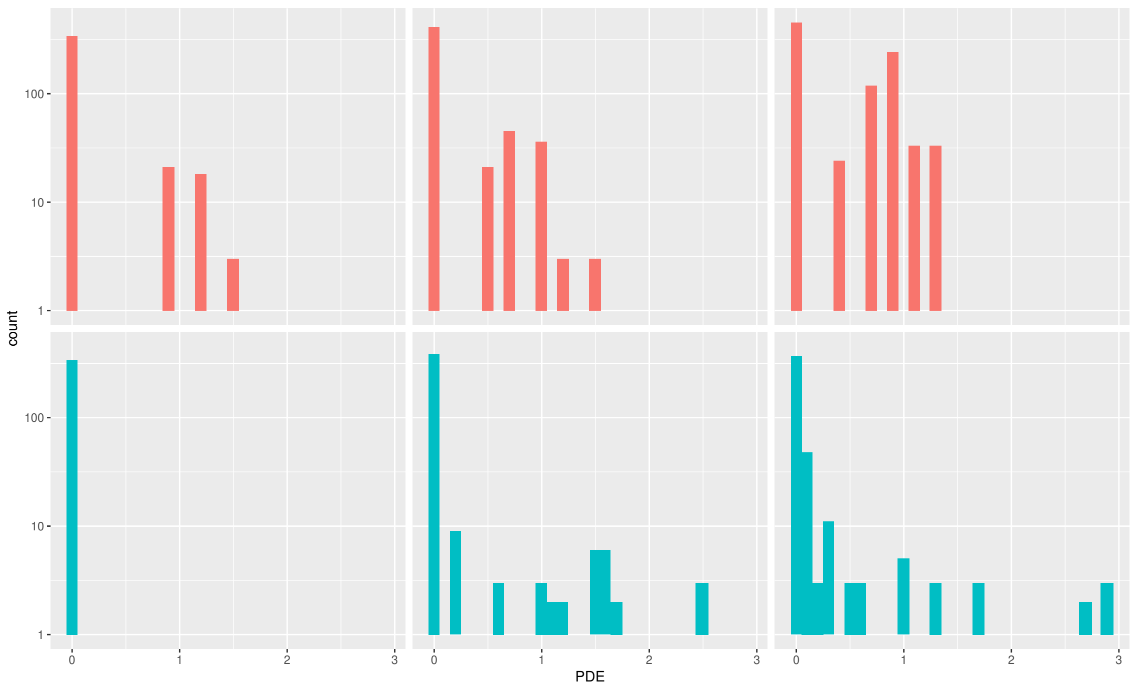
Growth Percentage: the number of times that the last level of the graph is bigger than the first level.

Median Rate Growth: the quantity of nodes added after the first level until the end divided by the number of levels in the graph.

Using these properties, some statistical tests are carried to identify if they are factors in the suggested classification.

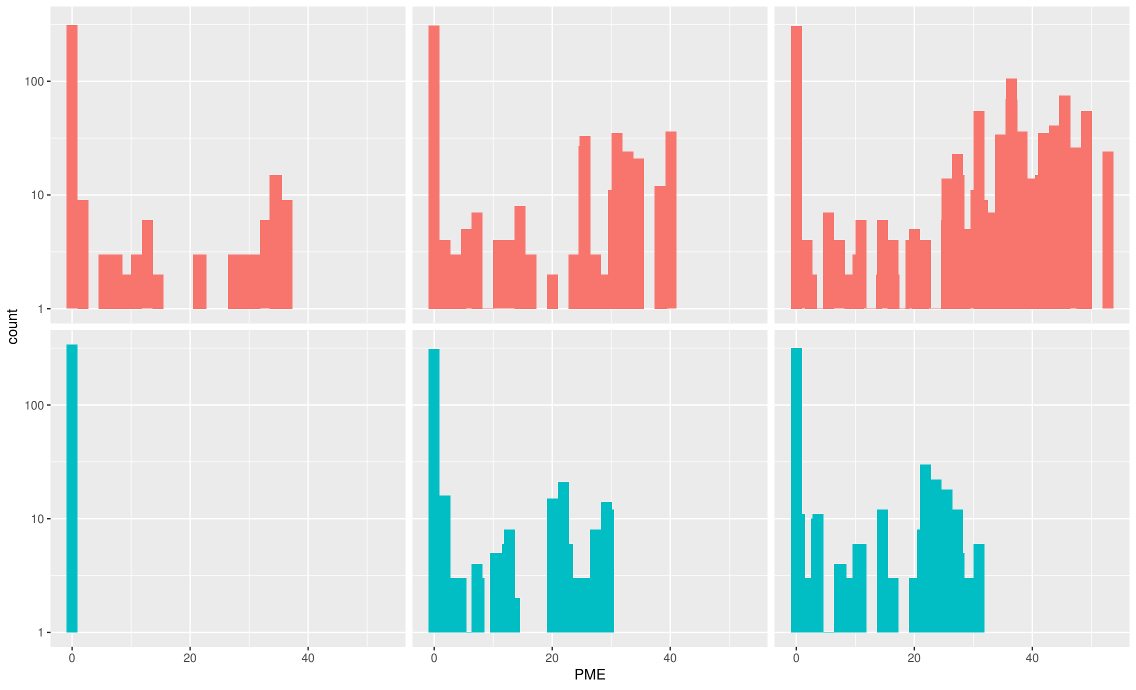


(a) Distribution of POE in instance *schedule-3-7*.

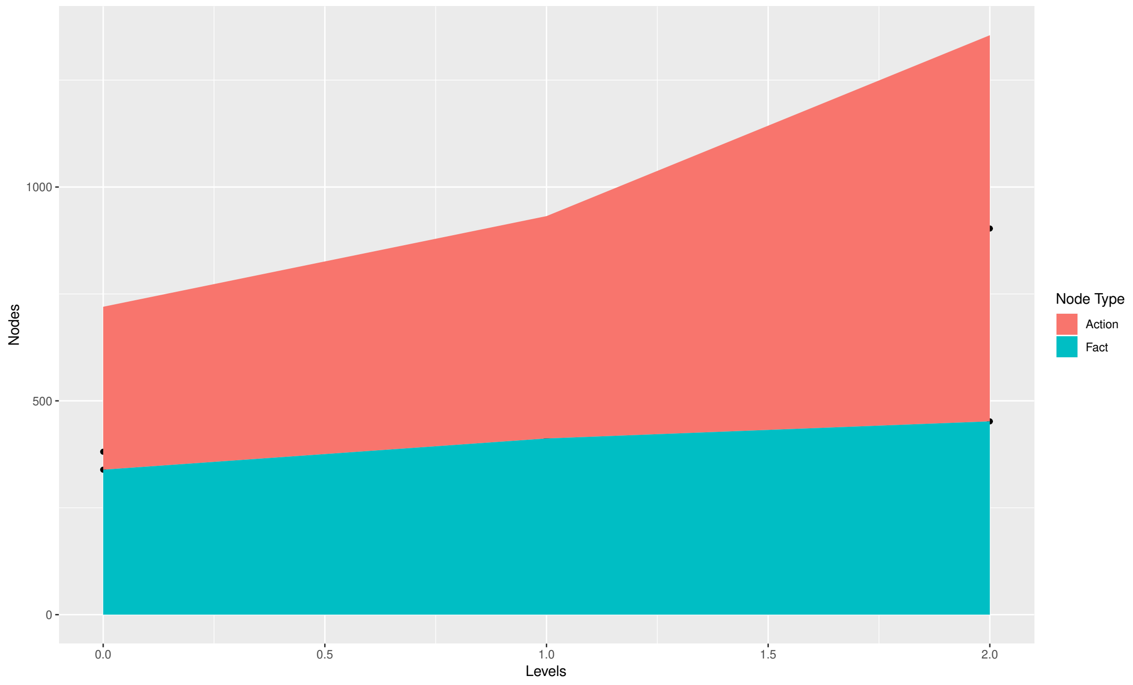


(b) Distribution of PDE in instance *schedule-3-7*.

Figure 5.20: Exploratory analysis of instance *schedule-3-7*.



(a) Distribution of PME in instance *schedule-3-7*.



(b) Node growth by level in instance *schedule-3-7*.

Figure 5.21: Exploratory analysis of instance *schedule-3-7*.

Chapter 6

Experiments

In this chapter, the experiments used to find a relation between the difficulty of the instances problems and the graph properties measured are described. The data and tests used are presented followed by a discussion of the results.

6.1 Design of Experiments

The objective of the experimental design is to examinate possible relations between the proposed difficulty measure and the proposed graph metrics based on the instance classification. The Spearman and Kendall tests are used to measure the relation between the measurements and the classification, but just the Spearman test are discussed as the two result in almost the same values. Then, a median analysis is used to analyze the variations among the classifications. The data is available in the git repositoryof the present work.

Table 6.1 provides a summary of the data used: the number of total instances, the number of solved and unsolved instances, the number of extracted graphs and the number graphs with parallel and non-parallel actions.

Figures 6.1 and 6.2 show the Spearman tests between the measurements of the action and fact nodes and the classified parallel instances. Measures related with the

Table 6.1: Graph summary

	IPC1998	IPC2000	Total
Competence Total	150	810	960
Graph Extracted	110	484	594
Graph Not Extracted	40	326	366
Solved	82	263	345
Not Solved	28	219	247
Classified	65	197	262
Not Classified	17	66	83

growth of the graph show similar correlations, where for instances with high growth rates, a high quantity of nodes in the last level or in the total graph the instances also increase. These instances are more likely to be unsolvable than the instances with low growth rates and are more likely to solve. The growth percentage show a low correlation for some classifications or it is discarted by its significance. For the classified instances, they have low correlation values and weakly suggest a relation among the classifications and the tested properties but all measures reinforce the relation for the unsolved instances. For the measures of POE, PDE, and PME, the values related to the mode of the distribution show that for high values of POE in the action nodes, the instance is more likely to be solvable and for low values it is more likely to be unsolvable. The mode of PDE and PME have a low correlation value and low significance. Finally the percentage of nodes for POE, PDE, and PME with the maximum node value are correlated, the low values are related with the unsolved instances this means that the graphs with multi-modal distributions are unsolvable while the graphs with one or two modes are more likely to be solvable.

Figures 6.3 and 6.4 show the Spearman tests between the measurements of the action and fact nodes and the classified non-parallel instances. The correlation values are similar as for the parallel instances, reinforcing the results of the instances with parallel actions with difference in the mode values of the fact nodes where the

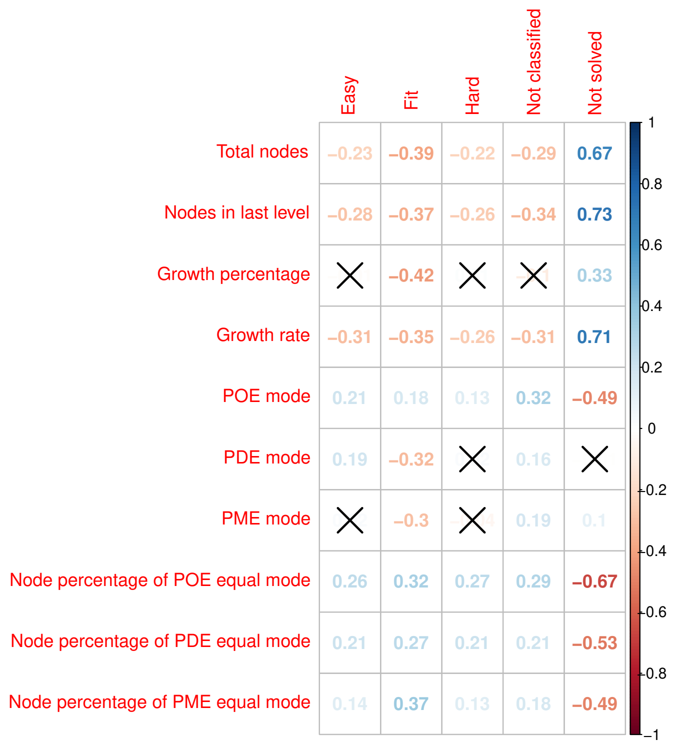


Figure 6.1: Correlation test by Spearman method of the measurements on the action nodes with parallel classified instances, the crossed cells are the ones that had a p-value > 0.05, the numbers in the cells are the τ values.

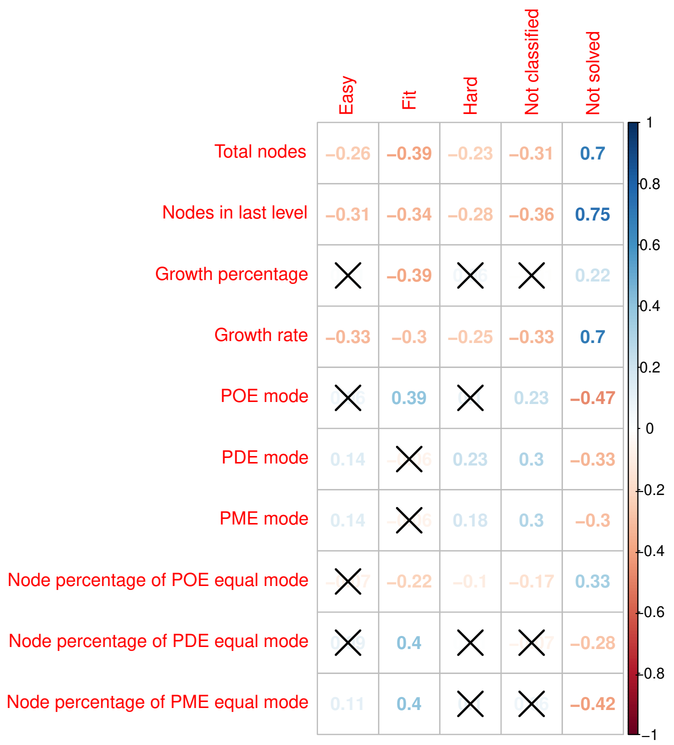


Figure 6.2: Correlation test by Spearman method of the measurements on the fact nodes with parallel classified instances, the crossed cells are the ones that had a p-value > 0.05, the numbers in the cells are the τ values.

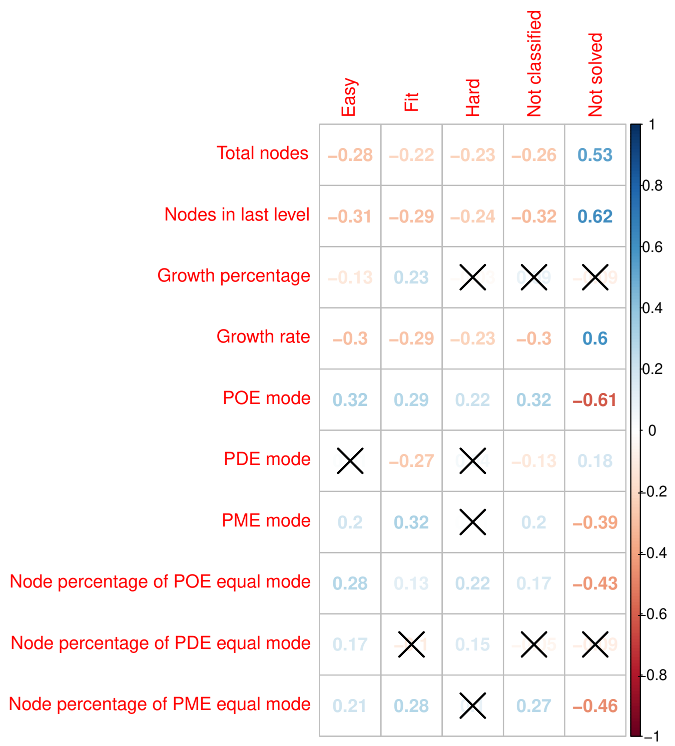


Figure 6.3: Correlation test by Spearman method of the measurements on the action nodes with non-parallel classified instances, the crossed cells are the ones that had a p-value > 0.05, the numbers in the cells are the τ values.

correlation values suggest that when the mode of the fact nodes is low for PDE and PME, the instance is unsolvable and with high modes it is solvable.

6.2 Results

Based on the correlation results, the differences of the metrics with the classifications are identified. All the distributions are tested with ANOVA, but they are not normaly distributed, so a Kruskal-Wallis test is done to corroborate if the medians are equal. Later, a Dunn test is used to identify the differences.

Figures 6.5 and 6.6 show the comparison of the growth rate of the action nodes in parallel instances. It is easy to see that the solved instances have a growth rates

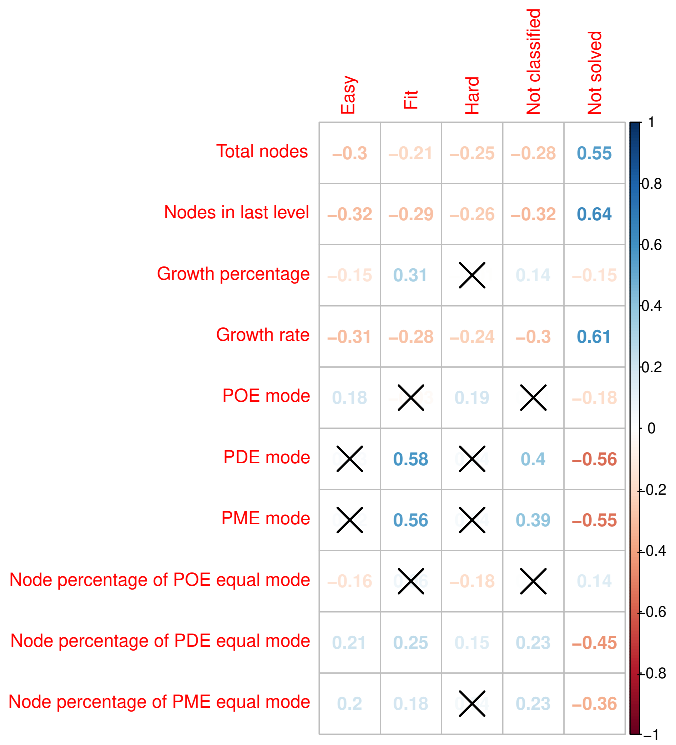


Figure 6.4: Correlation test by Spearman method of the measurements on the fact nodes with non-parallel classified instances, the crossed cells are the ones that had a p-value > 0.05, the numbers in the cells are the τ values.

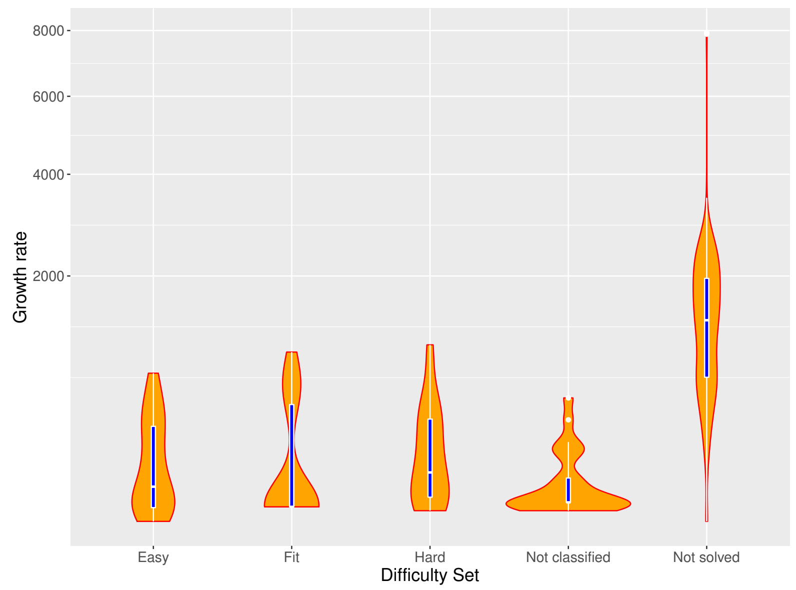


Figure 6.5: Comparison of the growth rate of action nodes value of the graphs among the instance classification, Kruskal-Wallis p-value $= 0.0001$, Dunn's p-value $= 0.0001$ for all classifications with unsolved

below of 1000 nodes per level while the unsolved instances have growth rates above than that. The grow rate of facts in non-parallel instances show a similar result but with a fewer value of 125 nodes.

Figures 6.7 and 6.8 compare the percentage of nodes in the last level of the graph with a percentage of mutex edges equals to the mode value among the classifications for parallel and non-parallel instances. The unsolved instances with a low percentage show that multi-modal distributions of PME are more likely to be unsolvable while the higher values that have few modes are more likely to be solvable.

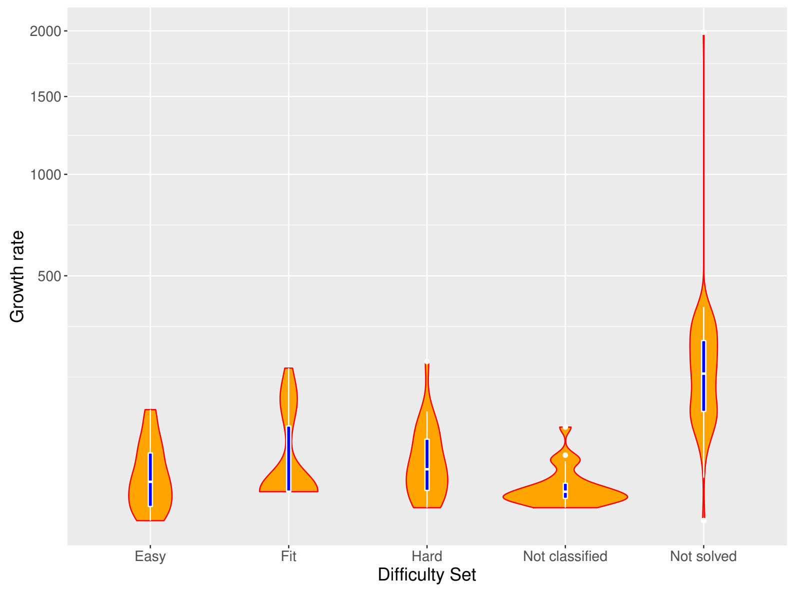


Figure 6.6: Comparison of the Growth Rate value of the graphs among the classification instances, Kruskal-Wallis p-value $= 0.0001$, Dunn's p-value $= 0.0001$ for all classifications with Not Solved

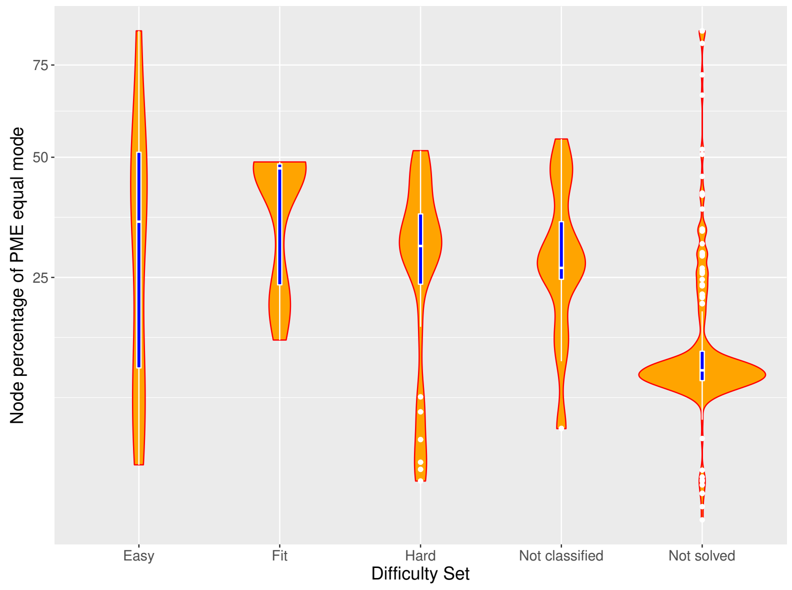


Figure 6.7: Comparison of the percentage of nodes in the last level of a graph with its percentage of mutex edges equals to the mode value of the distribution and the instance classification for parallel instances, Kruskal-Wallis p-value $= 0.0001$, Dunn's p-value $= 0.0001$ for all classifications with unsolved

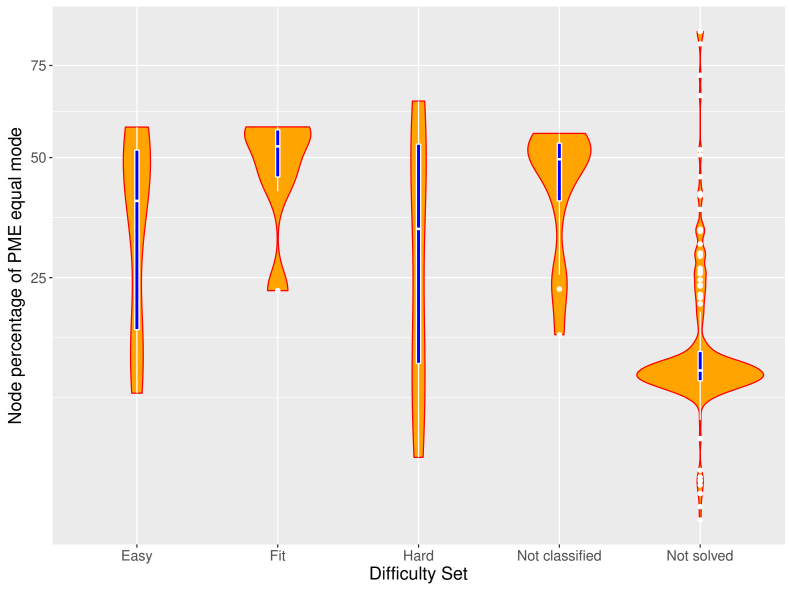


Figure 6.8: Comparison of the percentage of nodes in the last level of a graph with its percentage of mutex edges equals to the mode value of the distribution and the instance classification for non-parallel instances, Kruskal-Wallis p-value $= 0.0001$, Dunn's p-value $= 0.0001$ for all classifications with unsolved except with hard that have a p-value $= 0.01$

Chapter 7

Conclusions

The present work proposes a classification method to distinguish among the solved and unsolved instances using data from IPC results and the extracted planning graphs. A set of tools was developed to obtain the planning graphs with modification of the Blackbox planner. The use of data-mining techniques and a non-relational database allow us to store, transform, and analyze the graph data in a fast and efficient way. Finally, the results show that the metrics taken from the planning graph can be used to differentiate between solved and unsolved instances. The growth rates and the mode in the distribution of the edge percentage in the last level of the graph are examples of metrics that identify the instance classification.

The introduction of additional features and the use of classification methods often than others are of interest as future work, so as to help further differentiate between the easy and the hard instances. Also additional and more complex graph properties can be extracted and included into the set of structural characteristics

Bibliography

[1] Christer Bäckström. Equivalence and tractability results for SAS+ planning. In *Proceedings of the 3rd International Conference on Principles on Knowledge Representation and Reasoning*, pages 126–137, 1992.

[2] Christer Bäckström and Bernhard Nebel. On the computational complexity of planning and story understanding. In *Proceedings of the 10th European Conference on Artificial Intelligence*, 1992.

[3] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[4] Christer Bäckström, Peter Jonsson, and Simon Ståhlberg. Fast detection of unsolvable planning instances using local consistency. In *Sixth Annual Symposium on Combinatorial Search*, 2013.

[5] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, (90):281–300, 1997.

[6] John Bresina, Ari Jonsson, Paul Morris, and Kanna Rajan. Activity planning for the mars exploration rovers. In *ICAPS 2005 — Proceedings of the 15th International Conference on Automated Planning and Scheduling*, 2005.

[7] Daniel Bryce and Subbarao Kambhampati. A tutorial on planning graph based reachability heuristics. *Artificial Intelligence Magazine*, (28):47–83, 2007.

[8] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, (69):165–204, 1994.

[9] Reinarhd Diestel. *Graph Theory*. Springer, San Francisco, CA, USA, electronic edition, 2000. ISBN 978-3-96134-005-7.

[10] Kutluhan Erol, Dana S Nau, and VS Subrahmanian. On the complexity of domain-independent planning. In *AAAI*, volume 1, pages 381–386, 1992.

[11] Richard E Fikes and Nils J Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[12] Alfonso E Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5):619–668, 2009.

[13] C Cordell Green and Bertram Raphael. The use of theorem-proving techniques in question-answering systems. In *Proceedings of the 1968 23rd ACM national conference*, pages 169–181. ACM, 1968.

[14] Naresh Gupta and Dana S Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3):223–254, 1992.

[15] Nicholas G Hall and Marc E Posner. Performance prediction and preselection for optimization and heuristic solution procedures. *Operations Research*, 55(4): 703–716, 2007.

[16] Malte Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262.

[17] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[18] Malte Helmert. On the complexity of planning in transportation domains. In *Sixth European Conference on Planning*, pages 120–126, 2014.

[19] Jörg Hoffmann. Local search topology in planning benchmarks: An empirical analysis. In *IJCAI*, pages 453–458, 2001.

[20] Jörg Hoffmann. Where ignoring delete lists works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research*, 24:685–758, 2005.

[21] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253–302, 2001.

[22] Adele E Howe and Eric Dahlman. A critical assessment of benchmark comparison in planning. *Journal of Artificial Intelligence Research*, 17(1):1–33, 2002.

[23] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1194–1201, 1996.

[24] Henry Kautz and Bart Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *AIPS98 Workshop on Planning as Combinatorial Search*, volume 58260, pages 58–60, 1998.

[25] Fangzhen Lin, Malte Helmert, XuanLong Nguyen, Zaiqing Nie, Ullas Nambiar, and Romeo Sanchez. Aips'00: Planning competition the fifth international conference on artificial intelligence planning and scheduling systems. *Artificial Intelligence Magazine*, (22), 2001.

[26] Drew McDermott. The 1998 AI planning systems competition. *Artificial Intelligence Magazine*, (21):35–55, 2000.

[27] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edition, 2004. ISBN 978-0-08-049051-9.

[28] Christos M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994. ISBN 0201530821.

[29] John R Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 65–118. 1976.

[30] Jussi Rintanen. *Automated planning: Algorithms and complexity.* PhD thesis, Habilitation thesis, Albert-Ludwigs-Universität Freiburg, 2005.

[31] Stuart J. Russell and Peter Norving. *Artificial Intelligence, a modern approach.* Pearson Education, 3 edition, 2010. ISBN 978-0-13-604259-4.

[32] John R Searle. Is the brain's mind a computer program. *Scientific American*, 262(1):26–31, 1990.

[33] Kate Smith-Miles and Jano van Hemert. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61(2):87–104, 2011.

[34] Kate Smith-Miles, Jano van Hemert, and Xin Yu Lim. Understanding TSP difficulty by learning from evolved instances. In *International Conference on Learning and Intelligent Optimization*, pages 266–280, 2010.

[35] Mauro Vallati, Lukas Chrpa, Marek Grześ, and McCluskey. Description of participant planners of the deterministic track. In *The Eighth International Planning Competition*, 2014.

[36] Daniel S Weld. Recent advances in AI planning. *Artificial Intelligence Magazine*, 20(2):93, 1999.

[37] Gerlind Wisskirchen, Blandine Thibault Biacabe, Ulrich Bormann, Annemarie Muntz, Gunda Niehaus, Guillermo Jiménez Soler, and Beatrice von Brauchitsch. Artificial intelligence and robotics and their impact on the workplace. Technical report, IBA Global Employment Institute, 2017.

[38] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[39] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. SATzilla-07: the design and analysis of an algorithm portfolio for SAT. In *International Conference on Principles and Practice of Constraint Programming*, pages 712–727, 2007.

Index

Resumen autobiográfico

José Anastacio Hernández Saldaña

Candidato para obtener el grado de
Maestría en Ciencias en Ingeniería de Sistemas

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Tesis:
Structural characterization of planning problems

Nacido el 28 de febrero de 1986 en la ciudad de Monterrey, México. Terminó la Licenciatura en Ciencias Computaciones en la Facultad de Ciencas Físico-Matemáticas de la Universidad Autónoma de Nuevo León en el año 2012. Desarrollándose en instituciones públicas y privadas dentro de las áreas de base de datos, análisis de sistemas y desarrollo de software.