Universidad Carlos III de Madrid

TESIS DOCTORAL

Portfolio Approaches for Problem Solving

Autor

Sergio Núñez Covarrubias

Directores

Dr. D. Daniel Borrajo Millán y
Dr. D. Carlos Linares López

Departamento de Informática. Escuela Politécnica Superior

Leganés, 22 de Julio de 2016

Tesis Doctoral

Portfolio Approaches for Problem Solving

Autor: Sergio Núñez Covarrubias

Directores: Dr. D. Daniel Borrajo Millán y Dr. D. Carlos Linares López

Tribunal Calificador		Firma
Presidente:	....	....
Vocal:	....	....
Secretario:	....	....

Calificación: ....

Leganés, ... de .... de 2016

Quality means doing it right
when no one is looking

Henry Ford

Agradecimientos

A mis directores de tesis, Carlos Linares López y Daniel Borrajo Millán, porque para mí es todo un honor haber realizado este trabajo bajo su dirección. Ellos me dieron la oportunidad de empezar en el mundo de la investigación y, lo más importante, siempre me apoyaron y ayudaron cuando me hizo falta sin dudarlo, por lo que les estaré siempre muy agradecido.

A mi familia, por tener a los mejores padres que uno puede tener, por su dedicación, trabajo y sacrificio, porque sin ellos esto no hubiese sido posible. A mi hermano, por escucharme cada vez que lo he necesitado, por compartir conmigo sus experiencias y por darme su apoyo en cada momento de mi vida. Y por supuesto, a Erik, cuyos consejos sobre algoritmia avanzada y contribuciones teóricas han hecho posible que hoy esté escribiendo los agradecimientos de mi tesis.

A mis amigos, especialmente a Carlos, alias *Pache*, por estar siempre conmigo tanto en los buenos como en los malos momentos. A Anthony y Rosalía, porque siempre han estado ahí cuando les he necesitado y han conseguido que les sienta muy cerca de mí a pesar de la distancia. A Jesús, Nerea y Moisés, los que empezaron siendo compañeros de trabajo y se han convertido en grandes amigos, porque ellos han sido un gran apoyo día tras día. Y por último, y no menos importante por ello, a Alicia, cuyo cariño y apoyo ha sido fundamental para terminar la tesis.

A mis compañeros de universidad, AGC, JAI, MAHA, AGG, CGB, ACG, AMV, FCMR y AVG, por compartir conmigo tantos momentos y aventuras, y en especial a *Pani*, por estar a mi lado siempre que me ha hecho falta, a pesar del paso de los años.

A todas las personas con las que he coincidido en la Universidad Carlos III de Madrid. En especial a todos con los que he pasado buenos momentos, tanto dentro como fuera de la universidad. A todos los miembros del PLG, porque he aprendido mucho de vosotros y con vosotros. Sobre todo me gustaría agradecer a Álvaro, Ezequiel, José Carlos, Vidal, etc., el haber estado siempre dispuestos a ayudar y a compartir buenos momentos.

A todos los que habéis compartido este reto conmigo, ya seáis parte del pasado o del presente, esto es posible gracias a vosotros. A todos, muchas gracias.

Resumen

Durante los últimos años el concepto *portfolio* ha sido recuperado de la Teoría Moderna del Portfolio con el objetivo de mejorar el rendimiento de los solucionadores actuales. El concepto de portfolio aplicado a la resolución de problemas ha demostrado ser muy efectivo aprovechando las virtudes complementarias de los diferentes solucionadores. Sin embargo, todavía no se han estudiado en profundidad sus límites y posibilidades. Esta Tesis trata el problema de la configuración automática de portfolios secuenciales para resolver problemas combinatorios. Esto comprende, entre otros, tres retos principales: cómo seleccionar los solucionadores que conformarán el portfolio; cómo definir el tiempo asignado a cada solucionador que forma parte del portfolio; y cómo decidir el orden en el cual los solucionadores deberían ser ejecutados.

La mayoría de aproximaciones que estudian portfolios son totalmente empíricas. Por lo tanto, nosotros proponemos GOP, un nuevo método teóricamente fundamentado, basado en Programación Entera Mixta. Este método obtiene de forma automática la configuración óptima del portfolio (es decir, el conjunto de solucionadores que componen el portfolio y el tiempo asignado a cada uno de ellos) para una métrica específica y un determinado conjunto de entrenamiento. La configuración obtenida sólo es óptima para el conjunto de entrenamiento. Sin embargo, experimentos con datos de la Competición Internacional de Planificación y la Competición de SAT muestran que GOP supera de forma significativa al resto de aproximaciones bajo las mismas condiciones de experimentación. De hecho, MIPSAT, el portfolio secuencial que ha sido automáticamente configurado con GOP, ganó la medalla de plata en la categoría *open* de la Competición de SAT del año 2013. Además, MIPLAN, el sistema de planificación capaz de generar de forma automática una configuración de portfolio para un determinado dominio de planificación utilizando GOP, ganó la categoría de aprendizaje de la Competición Internacional de Planificación del año 2014.

El conjunto de entrenamiento utilizado para obtener portfolios secuenciales afecta a la calidad del portfolio generado y al tiempo necesario para configurarlo. Por lo tanto, en esta Tesis analizamos el impacto de la composición y del tamaño del conjunto de entrenamiento en el proceso de configuración del portfolio. Específicamente, nosotros utilizamos GOP para realizar un análisis *a posteriori* con el objetivo de seleccionar los problemas de entrenamiento que proporcionan la información más relevante para la técnica de configuración de portfolios. Los resultados de la experimentación realizada sugieren que el mejor conjunto de entrenamiento debería estar compuesto por un pequeño número de problemas, los cuales sólo unos pocos solucionadores deberían ser capaces de resolver.

Esta Tesis también aborda el problema relacionado con el orden de los solucionadores en los portfolios secuenciales. En la literatura no aparecen trabajos que estudien la relación entre el orden de los solucionadores en el portfolio y su rendimiento a lo largo del tiempo. Nosotros proponemos ordenar los solucionadores en los portfolios secuenciales de forma que el portfolio ordenado resultante maximice la probabilidad de obtener el mejor rendimiento en cualquier instante de tiempo. Además, hemos presentado un algoritmo *greedy* y otro óptimo para resolver este problema. Nuestros resultados demuestran que la aproximación *greedy* obtiene de forma eficiente soluciones muy cercanas a la óptima. Además, esta aproximación generaliza mejor que el algoritmo óptimo, el cual padece sobre-aprendizaje.

En resumen, esta Tesis estudia varios problemas relacionados con el diseño automático de portfolios secuenciales. Nosotros hemos diseñado un nuevo método para configurar portfolios y hemos tratado el problema del orden en el que los solucionadores de los portfolios secuenciales deberían ser ejecutados. Los experimentos realizados muestran que los portfolios obtenidos son extraordinariamente competitivos y con mucha frecuencia superan al resto de aproximaciones.

Abstract

In recent years the notion of portfolio has been revived from the Modern Portfolio Theory literature with the aim of improving the performance of modern solvers. This notion of portfolio applied to problem solving has shown to be very effective by exploiting the complementary strengths of different solvers. However, a deeper understanding of the limits and possibilities of portfolios is still missing. In this Thesis, we deal with the problem of automatically configuring sequential portfolios for solving combinatorial problems. It comprises, among others, three main challenges: how to select the solvers to be part in the portfolio; how to define the runtime allotted to each component solver; and how to decide the order in which the component solvers should be executed.

Most approaches to the study of portfolios are purely empirical. Thus, we propose a new theoretically-grounded method based on Mixed-Integer Programming named GOP. It automatically derives the optimal portfolio configuration (i.e., the set of component solvers and the time allotted to each one) for a specific metric and a given training set. Optimality is only guaranteed for the given training set. However, experimental results both with data from the International Planning Competition and the SAT Competition show that GOP significantly outperforms others under the same conditions. Indeed, MIPSAT, the sequential SAT portfolio automatically configured with GOP, won the silver medal in the Open track of the SAT Competition 2013. In addition, MIPLAN, the planning system which is able to automatically generate a portfolio configuration for a specific planning domain using GOP, won the learning track of the International Planning Competition 2014.

The training benchmark used to derive sequential portfolios affects the quality of the resulting portfolio and the time required to compute it. Hence, we analyze in this Thesis the impact of the composition and the size of the training benchmark in the portfolio configuration process. Specifically, we use GOP to perform *a posteriori* analysis with the goal of selecting the training instances which provide the most relevant information to the portfolio configuration technique. Empirical results suggest that the best training benchmark should be composed of a small number of instances that only a few solvers are able to solve.

This Thesis also addresses the problem related to the order of the component solvers in a sequential portfolio. In the literature, not much work has been devoted to a better understanding of the relationship between the order of the component solvers and the performance of the resulting portfolio over time. We propose to sort the component solvers in a sequential portfolio, such that the resulting ordered portfolio maximizes the probability of providing the largest performance at any point in time. We also introduce a greedy and optimal algorithms to solve this problem. Moreover, we show that the greedy approach efficiently obtains near-optimal performance over time. Also, it generalizes much better than an optimal approach which has been observed to suffer from overfitting.

In summary, this Thesis studies different issues related to the automated design of sequential portfolios. We design a new portfolio configuration method and address the problem of the order in which the sequential portfolios should be executed. Experimental results show that the resulting portfolios are highly competitive and often outperform other state-of-the-art approaches.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This Chapter introduces the scope in which this work has been developed and defines the objectives of this Thesis. First, we introduce the notion of portfolio applied to problem solving. Next, we define the main goals of this Thesis and the challenges faced to accomplish them. Finally, we summarize the structure of this document.

1.1 The Portfolio Approach

Combinatorial problems arise in many areas of Computer Science and application domains like finding models of propositional satisfiability formulae (SAT), planning, scheduling, protein structure prediction, etc. The challenge of these tasks is to find an optimal or satisficing solution in the potentially large space of possibilities.

The AI community has been actively seeking new ways to improve the heuristics and search algorithms used for solving combinatorial problems. However, the inherent difficulty of solving these problems using domain-independent solvers implies that no single solver dominates all others in every domain; i.e., different solvers perform best on different problems. Also, it has been shown empirically in some areas of Artificial Intelligence like Automated Planning that if a solver does not solve a problem quickly, it is very unlikely that it will solve it at all (Howe and Dahlman, 2002).

These facts led to revive the notion of portfolio from the Modern Portfolio Theory literature (Markowitz, 1952). This notion of portfolio applied to problem solving is based on the following idea: several solvers are executed (in sequence or concurrently) with specific timeouts, expecting that at least one of them will find a solution in its allotted time. In case of solving problems optimally, the portfolio halts as soon as one solver finds a solution; otherwise, all solvers are invoked and the best solution is returned.

The portfolio approach has been applied to some problem solving tasks with remarkable results. Indeed, the organizers of the SAT Competition 2013[1] defined a specific track for algorithm portfolios and alternative approaches, where the winner was CSHC PAR8 (Malitsky et al., 2013c), an 8-core parallel portfolio. Also, the results of the International Planning Competition 2014 (IPC 2014)[2] show that three awarded planners and twenty nine out of sixty seven participant planners in the deterministic tracks were portfolios or planners that consisted of a collection of solvers.

[1]`http://www.satcompetition.org`
[2]`http://ipc.icaps-conference.org`

A number of successful portfolio approaches have been introduced in several areas over the last years. In fact, even tools aimed to easily generate algorithm portfolios have been recently presented in order to encourage the common adoption of the portfolio approach (Samulowitz et al., 2013). Most of the successful approaches rely on some collections of features with the aim of capturing the structure of each instance (Xu et al., 2008; Cenamor et al., 2014). Then, they exploit that knowledge to select the most suitable solver for each instance to be solved. However, the usefulness or the predictive power of the instance features can not be theoretically measured. Indeed, they are manually selected based on the empirical performance. On the other hand, other approaches focus on greedy techniques to generate portfolios that run several solvers to solve each test instance (Helmert et al., 2011). These approaches may not capture all the knowledge from the training data, which affects the quality of the resulting portfolio configuration. Thus, in this Thesis, we will define a new theoretically based technique (termed GOP) to generate sequential portfolios and we will theoretically address (when possible) some relevant issues related to the automated design of portfolios. The objectives of this Thesis are detailed in the next section.

1.2 Objectives

The main goal of this Thesis is to improve the current state-of-the-art of the design of sequential portfolios for problem solving. For this purpose, we mainly focus on two issues: the automated portfolio configuration process and the execution order of the sequential portfolios. The particular objectives which we aim to accomplish in this Thesis are:

- Design a new algorithm to automatically configure sequential static portfolios for Automated Planning. With the purpose of improving the current techniques, a new theoretically based method will be developed to compute the best linear combination of solvers according to a given training data set and an optimization criteria.

- Analyze the influence of the training instances in the quality of the planning portfolios. In order to increase our understanding of the relationship between the training data used to configure portfolios and the quality of the resulting portfolio, the influence of the benchmark used in the portfolio configuration process will be studied.

- Study the relationship between the performance of a sequential planning portfolio over time and the order in which its component solvers are executed. A formal definition will be proposed for the problem of ordering the component solvers in a sequential portfolio as a function defined over time. In addition, greedy and optimal algorithms will be developed to solve this problem.

- Generalize the contributions presented for AI planning to SAT. With the aim of contributing to other areas of problem solving, the contributions described for Automated Planning will be generalized and assessed on SAT.

1.3 Thesis Outline

This Thesis is organized as follows: first, Chapter 2 discusses the state-of-the-art in the automated design of portfolios for Automated Planning and SAT. Next, Chapter 3 describes GOP, the new technique which is able to compute the optimal portfolio configuration for a given benchmark. This chapter also shows the generalization capability of the portfolios derived with GOP and how to analyze the *utility* of the training instances in the portfolio configuration process using GOP. Chapter 4 formally defines the problem related to the order in which the component solvers in a sequential portfolio should be executed. Also, this chapter introduces an optimal and a greedy approaches to tackle this problem. Finally, Chapter 5 presents the conclusions of this Thesis and introduces future work.

Chapter 2

State of the Art

This Chapter describes the background and the related work of this Thesis. In particular, we first focus on presenting the background related to the portfolio approach and the fields of interests in this research. Then, we discuss previous work about the automated construction of portfolios for Automated Planning, SAT and other problem solving tasks.

2.1 Background

In this Section, we first give some definitions concerning algorithm portfolios. Next, we describe the portfolio configuration and execution process. We then formally define Automated Planning and SAT tasks, two of the most prominent Artificial Intelligence (AI) fields. Finally, we define the experimental setup which we have used to assess our contributions.

2.1.1 Portfolio Definitions

A portfolio can be classified by its behaviour and by the computational resources that the portfolio uses to solve each input problem. Therefore, a portfolio is termed *static* if its configuration is not modified once it has been computed —i. e., neither the component solvers and their allotted time nor the order of the execution sequence can be altered. If the portfolio has the ability to define a new configuration for each input instance, then it is termed *dynamic*. Also, if the portfolio invokes solvers in sequence, then it is termed as *sequential*, as opposed to *parallel* portfolios, which run multiple solvers concurrently. The base solvers executed by sequential and parallel portfolios can be, in principle, sequential or parallel. It does not affect to the classification of the portfolio.

In this Thesis attention is restricted to sequential portfolios and preemptive mode (i.e., the ability to stop execution and resume it later on if necessary) is not allowed. Thus, a sequential portfolio can be formally defined as shown below.

Definition 2.1 (Sequential Portfolio). *A sequential portfolio ρ is a collection of n pairs $\langle s_i, t_i \rangle_{i=1}^{n}$, where s_i is a component solver, t_i is the time allotted to its execution and the total allotted time in the portfolio, $\sum_{i=1}^{n} t_i$, should be less than or equal to a given time limit T.*

Using these definitions, the automated process of configuring static sequential portfolios can be defined as follows:

Definition 2.2 (Portfolio Configuration Task). *Given a set of candidate solvers S, a set of training instances I and a fixed time limit T, the portfolio configuration task consists of automatically deciding the portfolio configuration ρ of solvers $s \in S$ such that it maximizes the performance of the resulting portfolio for the given benchmark I and its total allotted time does not exceed T, where performance is measured with different metrics.*

2.1.2 Portfolio Configuration and Execution Process

This Thesis focuses on the automated design of algorithm portfolios. Hence, in this Section, we describe a general scheme for configuring and running portfolios. This scheme can be divided into three phases. First, the design phase aims to made several relevant decisions about the design of the portfolio. These decisions will help to define the portfolio configuration task. Second, the offline phase focuses on generating data which will be either input data for the portfolio configuration approach (dynamic approach) or the portfolio configuration (static approach). Finally, the online phase assesses the resulting portfolio using test instances. Each phase is detailed next.

2.1.2.1 Design Phase

Several decisions related to the design (such as target, scope and runtime behaviour among others) should be made before configuring the portfolio. The most relevant decisions are described next.

Scope The ideal portfolio should achieve a high performance over every benchmark. However, there are applications whose goal is only to perform well over a specific set of tasks (those that belong to the same domain) (Rodríguez-Molins et al., 2010). Therefore, it is necessary to determine if the portfolio should be configured for achieving a high performance over every possible benchmark (domain-independent) or, on the contrary, for obtaining a high performance over a specific benchmark (domain-dependent).

Target The current portfolio configuration tasks only focus on optimizing an objective function for a fixed time limit. This function is usually taken from international problem solving competitions. For instance, SAT portfolios aim to maximize the number of instances solved in a fixed time limit (Belov et al., 2014). However, the target of a portfolio is not only restricted to achieve a high performance at a given time value. It can also consider the growth in performance of the portfolio over time. Thus, the target of the portfolio should be clearly defined since the solvers included in the resulting portfolio, the time allotted to each solver and the order of each solver in the portfolio should be defined according to this target.

Computational resources The number of available core processors constraints the resulting portfolio. If there is a single core processor available, the portfolio will execute the component solvers in sequence. Thus, the portfolio configuration approach should know whether it can optimize the portfolio configuration to fully exploit the multiple-core facilities or not. The portfolio configuration is also defined according to the time limit and the available main memory. Therefore, these computational resources should be defined in the design phase.

Runtime behaviour A portfolio can use all the available time to attempt solving the input instance (static approach) or it can spend a piece of the available time to generate a specific portfolio configuration for solving the input instance (dynamic approach). Deciding whether using a static or dynamic portfolio is a difficult task. It may seem that the dynamic approach is clearly the best choice. However, there are some issues behind this choice which should be addressed before making a decision: How much runtime should be spent on configuring the portfolio? What is the relevant data that should be gathered for the configuration task? Why are these data relevant? What techniques can be used with the gathered data to configure the portfolio within the time limit defined for this task? What happens if it can not configure a portfolio within the time limit? Is the dynamic approach worth it? The answer to the last question is that it depends on several factors. The dynamic approach is not necessarily better than the static one. It usually depends on the data gathered to generate the portfolio configuration, among other aspects.

Information sharing The solvers executed by a portfolio approach could share information among them. This fact may seem to be essential in every portfolio approach. However, it implies some constraints and challenges which should be considered before making a decision:

- Identify the information to be shared. This information should be relevant and independent of the particular techniques used by every solver. For instance, in planning, the best solution cost found so far is usually shared among the component solvers (Seipp et al., 2015).
- Candidate solvers. Every candidate solver to be part in the portfolio should be able to take the shared information as input, use this information for its purpose (e.g. to solve the input instance) and generate updated information for the next solver as output.
- Configuration technique. The portfolio configuration technique should be able to optimize the portfolio configuration to exploit this skill.
- Communication among solvers. The communication process is very simple and cheap in the case of sequential portfolios. Each solver only receives the shared information when it starts its execution, since there is only one solver executing at the same time. However, parallel portfolios can run several solvers concurrently. Hence, solvers may send and receive messages continuously. This overhead introduced by the communication process may result in a degraded performance. Therefore, the balance between communication and solving task must be deeply analyzed. Also, the question if the information to be shared would be centralized or distributed should be addressed.

Candidate Solvers One of the most relevant tasks in the automated design of portfolios is to choose the set of solvers that should be considered to be part of the portfolio since its composition affects the quality of the resulting portfolio. It may seem that a good solution would be to consider all the available solvers. However, the number of candidate solvers affects the time required to generate the portfolio configuration. Also, increasing the number of candidate solvers does not necessarily improve the results. In addition, this set should be consistent with the previous decisions. For instance, if the portfolio has been designed for sharing information among its solvers, every candidate solver should be able to process and generate useful data.

In the literature, some approaches consider the participant solvers in the last international competition (Núñez et al., 2013; Malitsky et al., 2014). Others built new solvers to configure the portfolio (Seipp et al., 2012; Xu et al., 2010) or choose these solvers at hand (Gerevini et al., 2014; Valenzano et al., 2012) because there is neither a theoretical study nor an empirical technique to select the best candidate solver set. However, the questions addressed above should give some insight about its composition.

Training instances Every portfolio configuration approach requires a set of training instances to derive a portfolio. The composition of this set also affects the time required to configure a portfolio and the quality of the resulting configuration. For example, if the training instances are not solved by any candidate solver, it is very likely that the configuration approach will not be able to distinguish an empty portfolio from any other portfolio configuration.

Similarly to the set of candidate solvers, the previous decisions constrain the composition of the set of training instances. For instance, if the scope of the portfolio has been defined as domain-dependent, only instances from the same particular domain should be considered.

Selecting the training instances is a real challenge. In the literature, there are approaches which took competition benchmarks to compose the training set (Cenamor et al., 2014). Other approaches used random generators to generate training instances (Rizzini et al., 2015). These generators usually require domain expertise for the parameters of each domain generator. Also, these generators do not guarantee that the generated instances are solvable. Thus, some approaches focus on developing generators which avoid these limitations (Fern et al., 2004; Fuentetaja and Borrajo, 2006).

Execution sequence The problem of ordering portfolios (Núñez et al., 2015b) is a relevant issue which should be considered while configuring the portfolio. However, it is usually addressed as an independent task. The execution sequence defines the order in which the component solvers of a portfolio should be executed. In case of parallel portfolios, this sequence could also describe the execution order of each core processor.

As it will be discussed in Chapter 4, a portfolio should be sorted if its performance over time is relevant (target of the portfolio). As a consequence, in SAT and optimal planning, the average time required to solve problems can be significantly reduced while in satisficing planning, better solutions can be found more quickly, while preserving coverage or quality score respectively.

Portfolio configuration task and portfolio configuration approach The last step in the design phase is to define the portfolio configuration task and propose a portfolio configuration approach. The configuration task defines the particular problem of automatically configuring portfolios. This definition should include the scope, runtime behaviour, target, etc. of the designed portfolio. On the other hand, the configuration approach describes how to solve a particular portfolio configuration task. Therefore, several approaches can solve the same configuration task. In other words, the configuration task is the problem to be solved and the configuration approach is the technique proposed to solve the problem.

For instance, the work by Núñez *et al.* (Núñez et al., 2015a) defines *"the automated process of configuring static sequential portfolios as follows: Given a set of candidate solvers and a set of training instances automatically find the portfolio configuration of solvers such that it maximizes the performance of the resulting portfolio in the given benchmark"*. This configuration task is solved (among others) by ASPEED (Hoos et al., 2012) and CPHYDRA (O'Mahony et al., 2008), which will be described in detail in Section 2.2.

2.1.2.2 Offline Phase

Once the portfolio has been designed, the offline phase aims to generate input data for the portfolio configuration task. In case of static approaches, this phase also performs the process of configuring the portfolio, as it can be seen in Figure 2.1. Thus, the output data of this phase will be the configuration of the static portfolio which will be run for every test instance in the online phase.

Every approach in the automated design of portfolios runs every candidate solver with every training instance in the offline phase. The execution results are processed because planner perfor-

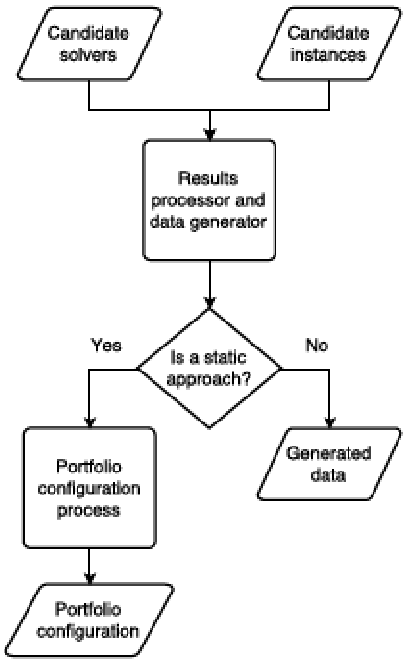


Figure 2.1: Overview of the offline phase of the portfolio configuration and execution process.

mance is usually part of the input data for the configuration task. The rest of the input data depends on the particular configuration approach. Some approaches also extract additional knowledge to generate alternative encodings (Vallati et al., 2014). Others extracts features from the training instances with the goal of learning predictive models (Gebser et al., 2011; Cenamor et al., 2013) or computing the distance among the instances (Kadioglu et al., 2011; Rizzini et al., 2015).

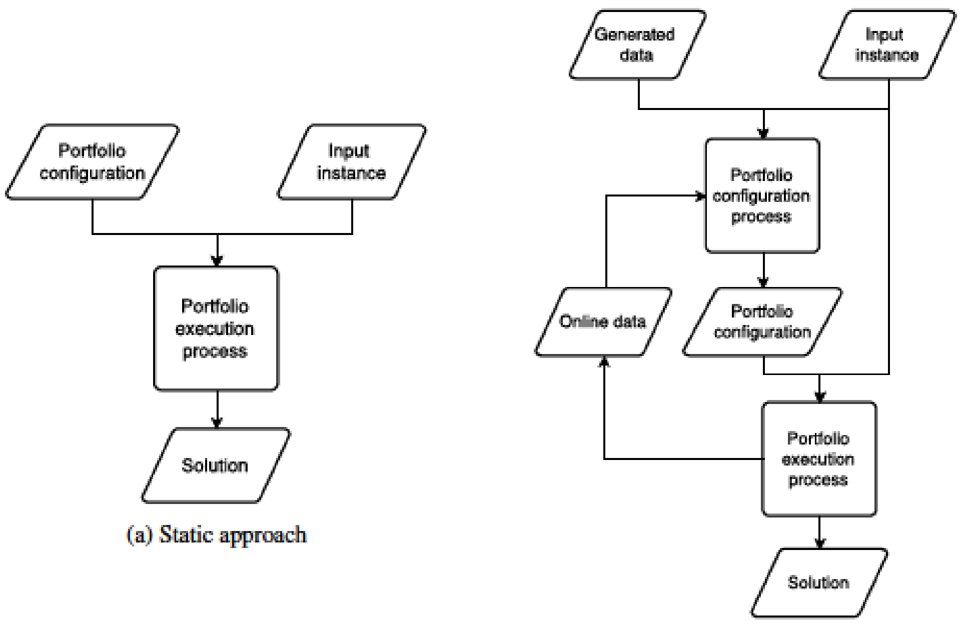
2.1.2.3 Online Phase

The goal of the online phase is to assess the resulting portfolio. In case of dynamic approaches, the portfolio configuration task should be performed at the beginning of the phase, as it can be seen in Figure 2.2. This task takes the data generated in the previous phase and the input instance, and then it generates the portfolio configuration. Once this task is over, or in case of static approaches, at the beginning of the phase, the portfolio is executed with the input instance.

A dynamic portfolio may be able to derive a portfolio configuration several times while solving the given instance. Therefore, it can gather data from the execution and then compute a more efficient portfolio configuration to solve the input instance. However, the total time available should be the same for static and dynamic approaches.

2.1.3 Automated Planning

Automated Planning is an area of Artificial Intelligence that studies the process of generating the sequence of actions that an agent should execute to achieve a set of goals from an initial state (Russell and Norvig, 2010). It is not only a theoretical field since Automated Planning leads to many



(a) Static approach

(b) Dynamic approach

Figure 2.2: Overview of the online phase of the portfolio configuration and execution process.

real world applications (Ghallab et al., 2004). Indeed, the diversity of applications for Automated Planning is quite broad: contact center (Kumar et al., 2014), mining operations (Burt et al., 2015), web-service composition (Traverso and Pistore, 2004), etc.

There are several planning models, which are based on various assumptions about the environment. For instance, classical planning studies deterministic and fully observable environments. Temporal planning studies planning tasks with concurrent actions. Planning with continuous actions focuses on environments where the set of states is not finite since the effects of the actions are continuous.

In this Thesis we are interested on classical planning which assumes that the environment is deterministic (the effect of actions is always known in advance), static (only the execution of an action can change the environment), finite (the number of states and actions is limited) and fully observable (the state is always fully-known). Formally, a classical planning task is defined as a tuple *P=(F,A,I,G)* where:

- *F* is a set of atomic propositions (also known as facts).
- *A* is the set of grounded actions. Each action $a \in A$ is defined as a triple *(pre(a), add(a), del(a))* (preconditions, add effects and delete effects) where *pre(a), add(a), del(a)* $\subseteq F$.
- $I \subseteq F$ is the initial state.
- $G \subseteq F$ is the set of goals.

A plan π is a sequence of applicable actions $\pi = (a_1, \ldots, a_n), \forall a_i \in A$ that allows for the transition from the initial state I to a final state g where all goals are satisfied $G \subseteq g$. The cost of a plan π is computed as the sum of the cost of every action in the plan, since actions can have non-unitary costs. Therefore, a distinction must be made between satisficing planning, that tries to find a plan preferring those of lower cost, and optimal planning, where the best plan (i.e. the plan with the lowest cost) must be found. In this work we have focused on both satisficing and optimal planning.

2.1.3.1 International Planning Competition 2011

The International Planning Competition is a competitive event organized in the context of the International Conference on Automated Planning and Scheduling (ICAPS). This competition provides an empirical environment for assessing planning systems under the same conditions.

The IPC 2011 (Linares López et al., 2015) was composed of three parts:

- Deterministic Part: It only considers deterministic actions where the transition between two states after any sequence of actions is fully predictable.
- Planning and Learning Part: Planners automatically extract domain dependent knowledge during an offline training phase and exploit this knowledge in the test phase.
- Uncertainty Part: It considers non-deterministic and probabilistic actions in fully observable, partially observable or unobservable domains.

The Deterministic part was split into two categories: Sequential and Temporal. In this work we have focused on the satisficing and optimal tracks of the Sequential Deterministic Part and on the learning track (Planning and Learning Part) of the Planning Competition. The main differences between satisficing planning and optimal planning can be summarized as follows:

- Satisficing planners can generate more than one solution, each with a different cost, whereas optimal planners generate at most one solution. Thus, while the performance of optimal planners is qualified with a binary variable (whether a particular planning task is optimally solved or not), satisficing planners are qualified with a list of timestamps (when each particular solution was generated) along with their cost.
- Commonly, satisficing planners maximize the sum of the quality score of each problem (which is defined below) over a set of planning tasks while optimal planners maximize coverage; i.e., the number of solved problems.

The goal of every participant planner in every track of the competition was to maximize a given score. Several equations were used to determine the score of the planners. Each one assigned a value in the range $[0, 1]$ for each triple $\langle planner, planning\ task, time\ limit\rangle$. For instance, in the sequential optimization track, a score equal to 1 was assigned if and only if the planner found the lowest plan cost for the given planning task within the time limit. In any other case, it set a score equal to 0.

Specifically, in the optimal and satisficing tracks, the score of the resulting plans was computed using Equation (2.1) for every task. If a planner s did not solve an instance i, the quality score of this task was set to zero. Otherwise, the quality score of a solved instance was computed as the lowest plan cost found by any planner in the competition in time less or equal to t, divided by the lowest plan cost found by the participant planner s within the same time bound. In case of solving planning tasks optimally, the quality score of the resulting plans was a binary value, since plans have to be optimal. The planner with the highest *total quality score* was declared the winner in each track. The total quality score, $Q(s, I, t)$, was computed as the sum of the quality score over all instances $i \in I$.

$$quality\_score(s, i, t) = \begin{cases} \frac{\text{cost}_i^*}{\text{cost}_{s,i}}, & \text{if } i \text{ is solved by } s \text{ within } t \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

The set of planning tasks defined for both tracks was divided in 14 planning domains, each one consisting of 20 planning tasks. Each participant planner was allotted 30 minutes for every planning task on a single core processor and a memory limit equal to 6 GB.

2.1.3.2 International Planning Competition 2014

The IPC 2014 (Vallati et al., 2015) preserves the same structure and evaluation criteria used in the IPC 2011. However, in this edition, three novelties were introduced. The first one is related to the available computational resources and core features that participants have to support. Each participant planner had a memory limit of 4 GB to solve each planning task. Also, every planner should support conditional effects and negative preconditions. The second novelty is relative to the awards. Specifically, a special award for innovative planning techniques was introduced. It mainly focused on planners which exploit techniques that are "new" for deterministic planning and performed reasonably well in the competition. The last novelty was the agile track, which was included in the Deterministic part.

The agile track aims to minimize the time required for finding a satisficing solution of a planning task. The available time to solve each instance is 5 minutes. The evaluation metric does not consider the cost of the resulting solutions. If a planner s does not solve a problem i within the time limit, s achieves zero points for the problem i. Otherwise, the score of the planner s which solves the instance i in time t (within the time limit) is computed in the range $[0, 1]$ using Equation (2.2), where t^* is the minimum time in seconds required by the fastest planner in the competition to solve the problem within the time bound, rounding all times to the ceiling value. Therefore, smaller time values stand for better scores. The planner with the highest sum of time scores over all planning tasks is declared the winner of the track.

$$time\_score(s, i, t) = \begin{cases} 1, & \text{if } t \leq 1s \\ \frac{1}{1+\log_{10}(t/t^*)}, & \text{if } t > 1s \end{cases} \tag{2.2}$$

2.1.4 SAT

SAT is one of the most prominent AI challenges. It deals with propositional reasoning tasks using SAT solvers. These solvers provide a generic combinatorial reasoning and search platform that can often solve hard structured problems with over a million variables and several million of constraints (Harmelen et al., 2008). Many problems typically have multiple translations to SAT since SAT stands at the crossroads among Logic, Graph Theory, Computer Science, Computer Engineering and Operations Research (Biere et al., 2009). Therefore, SAT has a good amount of practical applications in areas like Software and Hardware Verification (Gupta et al., 2006), Scheduling (Horbach et al., 2012) and Test Pattern Generation (Czutro et al., 2014), among others.

Propositional Satisfiability is the problem of deciding whether the variables of a propositional formula in Conjunctive Normal Form (CNF) can be assigned a boolean value in such a way that the formula evaluates to true. This is a classic NP-complete problem (Cook, 1971).

A propositional formula F is in CNF form if it is a conjunction (AND, $\wedge$) of clauses, where each one is a disjunction (OR, $\vee$) of literals. Each literal can only be a propositional variable or its negation (NOT, $\neg$). For instance, $F = (a \vee \neg b) \wedge (\neg b \vee c \vee \neg d \vee \neg e) \wedge (a \vee e) \wedge (\neg c \vee f)$ is a CNF formula with six variables and four clauses. An assignment for a propositional formula such that all clauses evaluate to true is said to be a satisfiable (SAT) solution. But if there is no assignment satisficing all clauses, the propositional formula is said to be unsatisfiable (UNSAT). Following the example, a SAT solution for the formula F shown above can be defined as the model $M = \{a = \top, b = \bot, c = \top, d = \top, f = \top\}$, where $\top$ and $\bot$ refer to the true and false values respectively, which can be assigned to a propositional variable. Note that not all the variables have to be assigned in the model, as shown in the example.

2.1.4.1 SAT Competition 2013

The International SAT Competition is a biennial event for Boolean Satisfiability solvers. It is organized in the context of the International Conference on Theory and Applications of Satisfiability Testing (SAT). The aim of this competitive event is to assess the progress in state-of-the-art procedures for solving Boolean Satisfiability problems.

The SAT Competition 2013[1] was split into the Main track and the Minisat Hack track. The goal of the first one is to determine whether a propositional formula in CNF is satisfiable or not. The second one aims to see how far the performance of Minisat (Eén and Sörensson, 2003) can be improved by making minor changes in that solver.

The Main track was composed of several tracks, where each one was defined as a combination of:

- Type of solver. The organizers defined two type of solvers: *Core solvers* and *Alternative approaches*. The first one only allows participants to use solvers that employ at most two different SAT solving engines for all runs and at any time in one track. The second one represents any solver not covered by the definition of a *core solver*.

- Computational resources. Resources classified the participant solvers into *sequential solvers* or *parallel solvers*. The first ones only use one core for 5000 seconds CPU time and 7.5 GB of main memory to solve each SAT instance. The second ones have 8 cores, 15 GB of main memory and 5000 seconds wall-clock time available to solve each SAT instance.

- Benchmarks. The organizers defined three categories of benchmarks: *Application*, that encode several application problems in CNF, *Hard-Combinatorial*, whose instances aim to reveal the limits of current SAT solver technology, and *Random*, where its problems can be fully characterized.

  Also, each category defines three tracks according to the solution of its instances: SAT, SAT+UNSAT and certified UNSAT. In the case of satisfiable formulas, solvers are required to output a model of the formula. Solvers are only required to emit an unsatisfiability proof in certified UNSAT tracks.

An instance is solved by a solver if the solver generates the complete answer within the allocated resources. The solver with the highest number of instances solved is declared the winner in each track. Ties are broken in favour of lower CPU running times in the sequential tracks and lower wall-clock times in parallel tracks.

[1] http://satcompetition.org/2013/

2.1.5 Methods of Empirical Evaluation

The challenges addressed in this Thesis require an empirical environment to assess the resulting portfolios. In this Section we describe the methods that we used to perform the empirical evaluation of the portfolio approaches. Fortunately, the AI community has made many efforts to develop methods and tools to evaluate and compare the performance of different solver techniques. The International Planning Competition and the SAT Competition aim to assess the progress in the state-of-the-art procedures for solving planning and boolean satisfiability problems respectively. Both competitions are intended to evaluate solvers under the same conditions and under settings as fair as possible. Also, both competitions define benchmark suites, a common framework, evaluation metrics and the state-of-the-art solvers. Thus, both competitions allow us to analyze how good a portfolio is and to compare the performance of different portfolios. Specifically, we have evaluated the contributions of this Thesis on the SAT Competition 2013 and on the IPCs 2011 and 2014, since they define challenging benchmarks for both areas.

2.1.5.1 Benchmark suites

Every IPC is characterized by a benchmark suite, which is composed of a set of domains with their problem sets. This has allowed the community to create an extensive and diverse benchmark to use in our own experimentation. Also, the SAT Competition defines a set of diverse benchmark instances in each competition to test the capabilities of solving problems of different kinds. Thanks to that, there is a good set of instances to test solver capabilities.

Table 2.1 describes all the benchmarks used in the empirical evaluation of this Thesis. The *Benchmark* column is the *id* of each benchmark. Next, the *Source* column shows the competition for which the instances of each benchmark were defined. The *#instances* column details the number of instances of each benchmark. Finally, the last column shows the planning domains or SAT categories of the instances which compose each benchmark. It is important to remark that the planning domain termed THOUGHTFUL was not first used in the IPC 2014. However, it was considered as *new* in the satisficing benchmark from that competition because it was not included in the satisficing track of the IPCs 2008 and 2011.

2.1.5.2 Common framework

The settings of the international competitions are usually taken as a *de facto* standard. Therefore, the settings defined for the IPC 2011 have been used to evaluate portfolios in Automated Planning (see Section 2.1.3.1). On the other hand, the settings used to assess portfolios in SAT have been taken from the open track of the SAT Competition 2013, the track that was specifically defined for portfolio approaches. Hence, each portfolio had eight core processors available for 5000 seconds wall-clock time and 15 GB of memory to solve each SAT instance.

Besides, both competitions release the software used in the competition. Thus, this software has been used to validate the results of the empirical evaluation of this Thesis. Specifically, all the generated plans have been validated with VAL, the automatic validation tool (Howey et al., 2004) while every SAT model has been verified with the EDACC verifier (Balint et al., 2011).

In addition, all the experiments of this Thesis have been executed with the same hardware. We have used a cluster of Intel Xeon 2.93 GHZ quad core processor with 16 GB of RAM.

Benchmark	Source	#Instances	Planning domains or SAT categories
ALL OPTIMAL IPC DOMAINS 2014	IPC 2014	20 per domain	BARMAN, CAVE, CHILDSNACK, CITYCAR, FLOORTILE, GED, HIKING, MAINTENANCE, OPENSTACKS, PARKING, TETRIS, TIDYBOT, TRANSPORT, VISITALL
NEW OPTIMAL IPC DOMAINS 2014	IPC 2014	20 per domain	CAVE, CHILDSNACK, CITYCAR, GED, HIKING, MAINTENANCE, TETRIS
ALL SATISFICING IPC DOMAINS 2014	IPC 2014	20 per domain	BARMAN, CAVE, CHILDSNACK, CITYCAR, FLOORTILE, GED, HIKING, MAINTENANCE, OPENSTACKS, PARKING, TETRIS, THOUGHTFUL, TRANSPORT, VISITALL
NEW SATISFICING IPC DOMAINS 2014	IPC 2014	20 per domain	CAVE, CHILDSNACK, CITYCAR, GED, HIKING, MAINTENANCE, TETRIS, THOUGHTFUL
ALL IPC DOMAINS 2011	IPC 2011	20 per domain	BARMAN, ELEVATORS, FLOORTILE, NOMYSTERY, OPENSTACKS, PEG-SOLITAIRE, PARKING, PARCPRINTER, SCANALYZER, SOKOBAN, TIDYBOT, TRANSPORT, VISITALL, WOODWORKING
NEW IPC DOMAINS 2011	IPC 2011	20 per domain	BARMAN, FLOORTILE, NOMYSTERY, PARKING, TIDYBOT, VISITALL
OPTIMAL IPC DOMAINS 2008	IPC 2008	30 per domain	ELEVATORS, OPENSTACKS, PARCPRINTER, PEG-SOLITAIRE, SCANALYZER, SOKOBAN, TRANSPORT, WOODWORKING
SATISFICING IPC DOMAINS 2008	IPC 2008	30 per domain	CYBERSECURITY, ELEVATORS, OPENSTACKS, PEG-SOLITAIRE, PARCPRINTER, SOKOBAN, SCANALYZER, TRANSPORT, WOODWORKING
OPEN TRACK SC 2013	SAT 2013	100 per SAT category	Application, Hard-combinatorial, Random
FULL SC 2013	SAT 2013	1000 in total	Application, Hard-combinatorial, Random
FULL SC 2011	SAT 2011	1200 in total	Application, Crafted, Random

Table 2.1: Benchmark suite defined for the planning and SAT experiments.

2.1.5.3 Evaluation metrics

Two metrics have been selected to measure the performance of the sequential portfolios, which consider time as discrete with a discretization of one second. In case of solving problems optimally in planning or SAT problems, the performance is measured by the coverage. Otherwise, the quality score of every solution found is computed to determine the performance.

- Coverage, $C(s, I, t)$: number of problems in the benchmark I solved by the solver s in time less or equal than t per instance. It is the official metric for the SAT competitions and for the optimal track of the last three IPCs (since 2008).
- The total quality score, $Q(s, I, t)$, is computed as the sum of the quality score over all instances (see Section 2.1.3.1). It is the metric for the satisficing track of the last three IPCs (since 2008).

The current practice in the portfolio literature is to use the single best solver (SBS) and the virtual best solver (VBS) to define an upper bound on the solvers performance for a particular international competition. Thus, we have also used these upper bounds to analyze the performance of the portfolio approaches. SBS is the winner of the respective category of an international competition while VBS is an oracle which selects the best participant solver for each instance (Xu et al., 2012a). The VBS typically achieves much better performance than the SBS. However, VBS is not a real portfolio since it cannot be run on new instances (Xu et al., 2012a).

2.1.5.4 Solvers

All approaches for configuring portfolios use a set of (publicly available) solvers. We would like to acknowledge and thank the authors of the individual solvers for their contribution and hard work. In this Thesis we have used a large number of single solvers and several portfolios. For the sake of clarity, all of these solvers are described in Annex A.

2.2 Related Work

In this Section we describe the state-of-the-art techniques in the automated design of portfolios with particular references to the portfolio classification and the general scheme to design portfolios defined in Section 2.1.2. We first discuss the related work on Automated Planning. Then, we present the related work on SAT. Finally, we discuss existing portfolio generation techniques for other problem solving tasks like Constraint Satisfaction Problems (CSP) and Answer Set Programming (ASP).

2.2.1 Automated Planning

During the last years, several portfolios have been developed for solving planning tasks. Most of these portfolios have participated in the last planning competitions (IPC 2008, IPC 2011 and IPC 2014). The competition results show that the participant portfolios on the deterministic track, learning track and multicore track have often achieved the best scores. These planning portfolios and several relevant works from the planning literature are described in detail next.

2.2.1.1 BUS

The work by Howe *et al.* (Howe et al., 2000) described BUS, one of the first portfolio approaches for planning. In this work, the authors also empirically tested the three assumptions that support its design, which are described next:

1. No single solver outperforms others in every domain. Six planners were executed with 263 instances. The results showed that the best planner solved 110 instances out of the 176 planning tasks that were solved by at least one of the six planners considered in the experiment.

2. Most planners either solve a planning task quickly or fail at solving the instance. Their empirical study confirmed significant differences in time between success and failure solving planning tasks.

3. The performance of planning solvers can be predicted with domain and problem features. Their analysis of planner performance suggested that a subset of five basic features extracted from domains and problems can be used to predict if a planner will solve an instance and if so, the time required to solve it.

BUS run 6 planners in a round-robin scheme until a solution was found. In particular, it considered STAN (Fox and Long, 1998), IPP (Koehler et al., 1997), SGP (Weld et al., 1998), BLACKBOX (Kautz and Selman, 1998), UCPOP (Penberthy and Weld, 1992) and PRODIGY (Veloso et al., 1995). Most of these planners were based on GRAPHPLAN (Blum and Furst, 1997). However, BLACKBOX focused on translating planning tasks to SAT and solving them using several SAT techniques. This approach is currently exploited by state-of-the-art satisficing planners like SATPLAN (Kautz et al., 2006) and MADAGASCAR (Rintanen, 2014). Indeed, MADAGASCAR achieved impressive results for several planning domains in the IPC 2011.

For each input instance, BUS extracts some features from the given problem and planning domain. The component planners are then sorted in descending order of the ratio $\frac{P(A_i)}{T(A_i)}$, where $P(A_i)$ is the expected probability of success of algorithm A_i, and $T(A_i)$ is the expected run time of algorithm A_i. Both estimations are provided by linear regression models based on instance features. Once the ordering task is finished, BUS runs the planners as follows. It takes the first planner from the queue and allocates the expected time that the current planner needs to solve the input instance. If the planner solves the planning task, the portfolio ends. If the planner terminates unexpectedly, it is discarded. However, if the time allotted to the planner is reached without success or failure, BUS studies whether to assign additional time to the current planner and the proceeding planners until either the problem was solved or reached the predicted time to the next planner.

2.2.1.2 The works by Roberts *et al.*

In this Section we describe three works by Roberts *et al.* which were inspired by BUS (see Section 2.2.1.1). The work reported in (Roberts and Howe, 2006) computed two classification models to predict if a planner will solve a planning task (success) and if so, the time required to solve it (time). These models were generated using 1839 planning tasks and 57 features automatically extracted from these problems and domains. The authors attempted to find a subset of features which minimize the cost required to compute them while preserving the accuracy of the model. It was found that the model to predict time can be learned without expensive features and these features are slightly informative to model success. On the other hand, the accuracy for predicting time was assessed. The results indicated that this model was not accurate enough. Therefore, it was discarded to compute the time allotted to each planner in the portfolio. Instead, the authors computed a predefined sequence of slices of time, which are then used on a round-robin strategy. The component planners were sorted (for each test instance) in decreasing order of the probability of success. In addition, this work also addressed the issue of selecting the planners which should be run by the portfolio. The greedy set cover approximation algorithm was applied to compute a subset of non-dominated planners based on success performance. The results suggested that at least 14 out of the 23 candidate planners may be discarded. In the empirical evaluation of the portfolio approach, two dynamic portfolios were executed using the round-robin strategy. The first one executed the full set of planners while the second portfolio only considered the non-dominated planners. The results showed that the second portfolio solved more instances.

The second work by Roberts *et al.* (Roberts and Howe, 2007) performed a study of the planner performance using 28 planners (with their default parameters) and 4726 instances from 385 planning domains. The authors realized that most of these planning tasks were no longer challenging. Hence, the study focused only on problems which were solved by at most three planners and whose median time to solve them was greater than one second. It resulted in a set of 1215 solvable problems from 41 domains. The performance data resulting of that study was used to learn several models to predict success and time. The inputs to the learning algorithms were restricted to planner performance and a set of 32 features automatically extracted from problems and planning domains. The evaluation results of the computed models showed that success can be quite accurately predicted while time is harder to predict (for instance, high time values are uncommon and therefore very hard to predict). Finally, the learned models were applied to a portfolio approach which excluded dominated planners using a greedy set covering algorithm as in the previous work. Thus, the portfolio only considered 10 out of the 28 planners included in the performance analysis. The authors proposed four strategies to sort the execution order for each input instance, three of which used the learned models:

- *cover* used the set covering order.
- *pSuccess* discarded the planners that were predicted to fail and used the predicted probability of success to sort the remaining planners in decreasing order.
- *predTime* sorted planners in increasing order of the predicted time.
- *SimonKadane* used the ratio $\frac{pSuccess}{predTime}$ to sort planners in decreasing order (Simon and Kadane, 1975).

Moreover, three strategies were defined to compute the time allotted to each planner and run the portfolio, where only one exploited the learned models:

- *avgPlanner* allotted to each planner its average time to succeed and executed planners in sequence without preemptive mode.
- *predTime* assigned to each planner its predicted time for a given planning task (using the learned models) and it also executed the component planners in sequence without preemptive mode.
- *confInt* defined a sequence of time slices to each planner since it used a round-robin strategy to run the portfolio. First, it sorted the time required to solve each training instance and then it performed the analysis for each quartile $\{25, 50, 75, 80, 85, 90, 95, 97, 99\}$.

The empirical results indicated that the best portfolio ranked planners using the *SimonKadane* strategy and used the *confInt* technique to allocate runtimes, which empirically confirms the lower accuracy to predict time.

The third work by Roberts *et al.* (Roberts and Howe, 2009) extended the previous one by making a deeper analysis of planner performance using similar sets of planners and planning tasks. The analysis focused on a good number of issues and questions about the planner performance, which were examined with several tools like Sammon map, t-test, Wilcoxon signed-rank tests and log-likelihood ratio test among others. The difficulty of the problems was measured by the number of planners that are able to solve problems and the time required to solve them. It was found that the full set of over 4000 problems is *not very* difficult despite the difficulty of the problems proposed over the years has significantly progressed (which may bring some insight about the composition of the training set used to configure portfolios). Also, it was noticed that solvable problems are usually solved very quickly. On the other hand, the progress of the planner capabilities were analyzed. An exhaustive analysis showed that planners have been considerably improved over time. However, not every old planner should be considered obsolete or subsumed by others. The performance analysis noticed that some older planners performed best on some (older too) domains (although these planners did not perform well on recent IPCs). Besides, the set of problems solved by each planner was not a subset of the problems solved by any other. Another issue addressed in this work is related to the need to have challenging problems. The authors followed the methodology described by Taillard (Taillard, 1993) to generate new problems for existing domains and also for a new planning domain. The results showed that the challenging problems were successfully generated since they increased the difficulty of the previous problems.

2.2.1.3 PBP

PBP (Gerevini et al., 2014) was the winner of the learning tracks of IPC 2008 and IPC 2011. It was a portfolio-based planner with macro-actions, which automatically configured a static sequential portfolio of domain-independent planners for a specific domain. The portfolio configuration generated by PBP, called cluster of planners, was composed of an ordered set of triples. Each triple contained a component planner, a possibly empty set of macro-actions computed for the component planner in a particular domain and a sequence of increasing slices of times (called time slots). This sequence of runtimes was used by the round-robin scheme to run the portfolio. In particular, the round-robin scheme followed a circular order to run each component planner. In each iteration, it run a particular planner with its set of macro-actions for the corresponding time slot. The first time that the planner was executed, the first slice of time defined the time allotted to the planner, the second time the total time allotted to the planner was defined by the second time slot and so on. Each slot defined the time that the planner should be executed from the start. Therefore, in each iteration a planner resumed its execution until its accumulated time was equal to the current time slot. If a planner did not terminate within its allotted time, it was added to the end of the set and the next one was executed. Otherwise, the planner was removed from the round-robin scheme.

In this work, the portfolio configuration process consisted of five steps. First, a number of sets of macro-action were computed for each candidate planner in the given planning domain using WIZARD (Newton et al., 2007) and MACRO-FF (Botea et al., 2005). Second, the performance of each candidate planner was measured with and without the sets of macro-actions computed for each planner. In particular, PBP measured the number of solved problems, time required to generate each solution and the quality score of every generated plan. Third, the sequence of time slots for each candidate planner was computed using a variation of the time allocation strategy proposed by Roberts *et al.* (Roberts and Howe, 2007) (see Section 2.2.1.2). The fourth step focused on selecting a cluster of planners. It simulated the round-robin execution of each possible cluster of at most k planners ($k = 3$ in their experiments) with the same training instances used in the second step. The simulation results were compared by a statistical analysis based on the Wilcoxon sign-rank test. Using the statistical results, a similar graph to the one used by Long *et al.* (Long and Fox, 2003) (to show the IPC 2002 results) was computed with the goal of selecting the cluster of planners for the portfolio configuration. The last step in the configuration process sorted the component planners of the selected cluster in ascending order of the first time slot of each planner.

The most recent version of PBP considers nine candidate planners: METRIC-FF (Hoffmann, 2003), YAHSP (Vidal, 2004), MACRO-FF, MARVIN (Coles and Smith, 2007), SGPLAN5 (Chen et al., 2006), FAST DOWNWARD, LAMA 2008 (Richter and Westphal, 2010), LPG-TD (Gerevini et al., 2006) and PARLPG (Vallati et al., 2013). Since PARLPG was based on running LPG (Gerevini et al., 2003) with a domain-specific parameter configuration, PBP performed an additional step (after the first step of the portfolio configuration process) to compute the parameter configuration for LPG on the given planning domain. On the other hand, PBP is able to compute two variants of PBP. PBP.S focuses on speed while PBP.Q aims to improve the quality score of the generated plans. Both versions exploit configuration knowledge for a given planning domain. Nevertheless, PBP can be used without this knowledge. In that case, the portfolio runs all candidate planners (without macro-actions) using a round-robin strategy and it assigns predefined time slots to each planner before being randomly sorted.

2.2.1.4 Fast Downward Stone Soup

Fast Downward Stone Soup (FDSS) (Helmert et al., 2011) was one of the awarded planners in the IPC 2011. In particular, two portfolios (denoted as FDSS-1 and FDSS-2) were generated using the FDSS technique to the sequential (optimal and satisficing) tracks of IPC 2011. This technique explores the space of static portfolios that can be configured with a set of candidate planners using a hill-climbing search algorithm. It takes a set of domain-independent planners as the initial portfolio, and allocates zero seconds to every planner. In each step, the algorithm generates the set of possible successors. Each successor increases the allotted time of one planner by a slice of the total time. To evaluate the successors, the algorithm uses the IPC evaluation metric (see Section 2.1.3.1) and a set of training problems: the whole collection of planning tasks used in all the past IPCs (a total of 1116 training instances ranging from 1998 to 2008). The best successor is selected as the current portfolio for the next iteration and it continues until the total time has been reached.

In satisficing planning, all candidate planners considered all actions to be of unit cost when computing the heuristics and, for weighted-A* (Pohl, 1970), the g-values. FDSS-1 and FDSS-2 communicate the cost of the best solution found among the component planners in the sequential portfolio. Thus, the component planners can prune states using that cost as an upper bound for the g-value.

The FDSS portfolios sort the execution sequence. The FDSS-1 portfolio for the satisficing track sorts planners by decreasing order of coverage. The component planners of FDSS-1 for optimal planning are sorted by decreasing memory usage. The remaining portfolios were sorted by other arbitrary orderings, which were not detailed by the authors.

In addition, all configurations for satisficing planning modify the configuration of the sequential portfolio once the first solution is found. Initially, all search algorithms ignore action costs in both variants. Once the first solution is found, FDSS-1 re-runs the successful planner and the remaining planners in the portfolio using all actions with their real cost. Instead, when the first solution is found, FDSS-2 discards all planners in the portfolio and it runs an anytime search algorithm with the same heuristic and search algorithm that successfully found the first solution using RWA* (Richter et al., 2010).

In a nutshell, the FDSS technique derives sequential domain-independent portfolios maximizing the IPC evaluation metric for a fixed time limit. The resulting portfolios sort the component solvers using empirical criteria. Also, these solvers share the best solution found so far among them. Moreover, the FDSS portfolios for satisficing planning are able to change their configuration using predefined rules once the planning task has been solved.

2.2.1.5 The work by Seipp *et al.*

The work by Seipp *et al.* (Seipp et al., 2012) automatically built static sequential portfolios learning them from domain-independent tuned planners. These tuned planners were restricted to configurations of the Fast Downward planning system (Helmert, 2006) which provides several state-of-the-art search algorithms and planning heuristics. This work can be split into two parts. First, a planner was automatically configured (by using the parameter tuning framework ParamILS (Hutter et al., 2009)) to excel in each of the 21 planning domains used in past IPCs (1998-2006). Harder problems were excluded with the goal of accelerating the tuning process. The easiest problems were also discarded because they are not helpful for the tuning process. ParamILS starts with an initial configuration. The authors took the initial configuration of the configuration process of FD-Autotune (Fawcett et al., 2011) since Fast Downward has not a default configuration.

In the second part, the authors used seven methods to learn portfolios of those tuned planners:

- *Stone soup*, the technique defined by FDSS (see Section 2.2.1.4), searches a good portfolio configuration on the space of static portfolios using a hill-climbing search.
- *Uniform* distributes the overall allotted time uniformly among all the planners.
- *Selector* employs brute force to compute the best subset of planners assigning the same amount of time to each planner within the subset and then it picks up the best one.
- *Cluster* applies the k-means clustering algorithm (MacKay, 2003) to the set of tuned planners (generating k clusters of planners) and selects the best planner from each cluster. All the available time is uniformly distributed among the k selected planners.
- *Increasing Time Limit* is the technique which iteratively increases the portfolio time limit by a slice of the total time. In each iteration, it increases the allotted time of the planner which maximizes the score of the instances that can be solved within the current time limit (excluding the instances that are already solved by the portfolio).
- *Domain-wise* is the iterative technique that in each iteration selects the most promising planning domain and then includes the planner that requires less time to improve the score on that domain.
- *Randomized Iterative Search* takes an initial portfolio configuration and then improves the portfolio using a randomized local search method. This iterative technique continues until the score of the portfolio does not improve after a large number of iterations.

All the portfolio generation techniques focused on the total quality score (see Section 2.1.3.1). However, for the sake of fairness, the authors made some adjustments to this metric. In particular, the scores were normalized by the number of instances on each domain. Also, the score of the tuned planners on domains which they were tuned was discarded. On the other hand, the authors did not detail neither if the portfolios were able to share information among their component planners nor the order of the component planners in the resulting portfolios.

All the resulting portfolios were assessed on the IPC 2011 sequential satisficing track. Interestingly, the results showed that configuring planners for a set of known domains can be very helpful to build portfolios that perform well on unknown domains. It was also found that the portfolio learned with the uniform method did not achieve a remarkable training performance. However, it was the best portfolio on the test set. On the other hand, as a result of this work, the portfolio derived with the uniform technique was submitted to the satisficing track of the IPC 2014. It was termed Fast Downward Uniform Portfolio (Seipp and Garimort, 2014) and was the sixth classified on the competition.

2.2.1.6 ArvandHerd

ArvandHerd (Valenzano et al., 2012) was the winner of the multi-core track of the IPC 2011. It was a manually configured parallel portfolio which run several configurations of lama 2008 (Richter and Westphal, 2008) and Arvand (Nakhost et al., 2011). lama 2008 was a planning system based on heuristic search with landmarks (Porteous et al., 2001) that won the satisficing track of the IPC 2008. Since this planner was able to use several heuristics to guide the search, the version included in the portfolio considered three heuristics: the landmark count heuristic (Richter et al., 2008) and two versions of the FF heuristic (Hoffmann and Nebel, 2001), a version which ignores action costs and another version that considers action costs. Also, this version used random operator ordering and restarts. In particular, if the planner exceeded a memory bound without finding a plan, it was set to restart with a new random seed and a different configuration. On the other hand, Arvand was a stochastic planner which used heuristically evaluated random walks. The version included in the portfolio was a particular parallelized version of the planner which included a shared walk pool and a shared UCB (Auer et al., 2002) configuration selector. In each core, a particular configuration executed an independent search episode. Once the search finished, it sent the trajectory to the shared walk pool and the reward for the current configuration to the shared UCB system. Then, it received the trajectory (from the shared walk pool) which should be executed with the configuration received from the UCB shared system.

ArvandHerd executed a single configuration of its version of lama 2008 at a time with the aim of avoiding memory partitioning issues. If the configuration exceeded a fixed memory bound, the planner was set to restart with another configuration and a new random seed in the same core. The remaining cores executed the parallelized version of Arvand. The set of Arvand configurations used in the portfolio was manually selected and it was based on the expertise of the authors. The authors designed the portfolio with these planners for several reasons. First, lama 2008 was the state-of-the-art planner. Second, this planner had a high memory consumption while Arvand had low memory requirements. Finally, the weakness of one planner was usually complemented by the strengths of another.

2.2.1.7 The works by Cenamor *et al.*

In this Section, we describe two works by Cenamor *et al.* related to the use of predictive models. Cenamor *et al.* (Cenamor et al., 2012) focused on analyzing the IPC 2011 results from a Data Mining perspective. It followed the CRISP-DM methodology (Chapman et al., 2010) with the goal of generating predictive models. First, a set of features was automatically extracted from the problems and domains defined in the competition. The SAS$^+$ problem representation (Bäckström and Nebel, 1995; Helmert, 2009) and its induced graphs were also used to extract useful features. Moreover, the time required to generate every plan in the competition and the quality score of the resulting plans were collected. Second, the set of features and the planner performance were processed to compute the input data for the learning algorithms. Third, a good number of learning algorithms were executed with the purpose of selecting the best classification model to predict success and the best regression model to predict time. In particular, the regression models aimed to predict three different time values: the time required to solve problems (i.e. first plan found), the time needed to find the best solution and the median time value of the plans generated. Finally, the learned models were assessed over problems from unknown domains (using the leave-one-domain-out technique) and over new problems from known domains (using the cross validation approach). The results showed that the accuracy significantly worsened when the models were evaluated on new domains. In addition, the authors performed a semantic analysis of the computed models.

The work (Cenamor et al., 2013) extended the previous one by focusing on the deployment phase of the CRISP-DM methodology. This phase aimed to configure dynamic portfolios using the classification and regression models. The best classification model was generated by the J48 algorithm (Quinlan, 1993) while the best regression model was computed by instance-based learning (Briscoe and Caelli, 1996) with $k = 3$. In this work, the regression model focused only on predicting the time expected to find the best plan. On the other hand, five strategies were defined to compute portfolios, which sorted the component planners by the confidence of the predictive models:

- *Equal Time* applied the uniform method (which allocated the same amount of time to each planner) to configure the portfolio.
- *Best Confidence Estimation* selected the candidate planner with the highest confidence value provided by the classification model for each test instance. In case of a tie, all the planners with the highest confidence value were included in the portfolio and the overall allotted time was uniformly distributed among them. Otherwise, it selected the planner with the lowest confidence value of failure for the test instance.
- *Best 5 Confidence* applied the uniform method to the five planners with the highest confidence value of success for each test instance.
- *Best 10 Confidence* is the same strategy than the previous one but it selected the ten planners with the highest confidence value (instead of 5).
- *Best 5 Regression* selected the five planners with the highest confidence value of success (regarding each test instance) to be part in the portfolio. Next, it computed the sum of the predicted time for each planner using the regression model. Finally, the time allotted to each planner was a linear proportion of its predicted time with respect to the sum of the predicted times and the total time.
- *Best 10 Regression* is the same strategy than *Best 5 Regression* but selecting 10 planners instead of 5.

All strategies were applied to the satisficing track of the IPC 2011 using the split evaluation and the leave-one-domain-out approach. The first technique aimed to assess the generalization capability of the resulting portfolios to new problems of known domains (which were used to configure the models). The second approach evaluated the portfolios on new domains. The results showed that the strategies based on predictive models achieved good performance over problems of known domains. However, the performance of the resulting portfolios on unknown domains was worse.

2.2.1.8 ASAP and AGAP

The work by Vallati *et al.* described ASAP (Vallati et al., 2014), a domain-dependent static portfolio configuration approach. It was based on exploiting different encodings or reformulations of a given planning domain. In particular, it considered macro-actions (Chrpa, 2010) and (outer and inner) entanglements (Chrpa and McCluskey, 2012) to create alternative encodings. A macro-action is a planning operator which comprises a sequence of actions that can be executed at one time while entanglements refer to relationships between planning predicates and operators. ASAP was able to compute at most four different encodings since macro-operators and entanglements could not be found on some planning domains. Specifically, it considered one encoding with macro-operators and three encodings with the three possible combinations of the outer and inner entanglements.

The offline phase of the portfolio configuration process aimed to select the best pair $\langle encoding, planner\rangle$ for a given planning domain. For this purpose, first, a knowledge extraction process was performed (macro-operators and both entanglements). Second, it generated three encodings considering only outer, only inner and both entanglements. Also, an encoding with macro-operators was computed and the resulting useless single operators were discarded. Third, all the possible combination of pairs $\langle encoding, planner\rangle$ were generated and then executed with every training instance (e.g. for each pair, the planner was executed with every training instance reformulated with the considered knowledge). Fourth, the performance of each pair was measured. Since ASAP was able to generate two different versions, ASAPs and ASAPq (which focused on maximizing runtime and quality score respectively), it measured the time required to generate the best plan for each planning task, the length of every resulting plan and the number of solved problems. Finally, the best pair $\langle encoding, planner\rangle$ was selected to be executed over every test instance. ASAPs selected the pair that achieved the highest IPC time score (see Section 2.1.3.2) while ASAPq used the total quality score (see Section 2.1.3.1). Tie-breaking was addressed by using a second criteria, which considered coverage, the number of problems in which each pair was the fastest and the mean runtime of solved instances.

ASAP considered a manually selected set of candidate planners. It was composed of LAMA 2011 (Richter and Westphal, 2010), LPG (Gerevini et al., 2003), METRIC-FF (Hoffmann, 2003), MP (Rintanen, 2012), PROBE (Lipovetzky and Geffner, 2011), SATPLAN (Kautz et al., 2006) and SGPLAN (Chen et al., 2006). On the other hand, ASAP was empirically analyzed on a selection of planning domains. Moreover, ASAP was assessed against the most recent version of PBP (see Section 2.2.1.3) on the domains defined for the learning track of the IPC 2011. The results showed that ASAPq outperforms PBP.Q while the time score achieved by ASAPs was slightly lower than the time score of PBP.S.

AGAP (Chrpa and Vallati, 2014) was an improved version of ASAPq which took part in the learning track of the IPC 2014. There are two main differences between both approaches. First, AGAP considered the same set of candidate planners but excluding SATPLAN. Second, AGAP used different encodings for the given domain. It was also able to generate at most four encodings considering entanglements and macro-operators. However, it considered one encoding with only outer entanglements, two encodings using macro-operators which exploit outer entanglements (Chrpa et al., 2014) and another one using macro-operators from inner entanglements (Chrpa et al., 2013).

2.2.1.9 AllPACA

AllPACA (Malitsky et al., 2014) was a dynamic sequential portfolio of domain-independent planners that took part in the optimal track of the IPC 2014. It was based on the algorithm selection technique, which aims to select the most suitable solver for each input instance and which has been successfully applied to the portfolio approach in SAT, MaxSAT and CSP. AllPACA was designed to select the planner that solves the input instance in the shortest time span. The choice relied on predictive models (random forest) which were generated using planner performance and instance features. Specifically, it measured the time required by each participant planner in the optimal track of the IPC 2011 to solve every available planning task from all the previous IPCs. Also, it extracted a set of 65 features from each available problem.

For each test instance, AllPACA extracted its set of instance features and it then predicted the planner which will be able to minimize the time required to solve the given instance. The selected planner was executed until the total time was exceeded or the planner generated the optimal plan. In the latter case, the solution was validated. If the plan was valid, AllPACA ends. Otherwise, it executed a default planner for the remaining time. In particular, it run the Fast Downward planner with the LM-CUT heuristic.

2.2.1.10 IBACoP and LIBACoP

The IBACoP (Cenamor et al., 2014) planning portfolios were the winners of the satisficing track and the runner-up of the multi-core track of the IPC 2014. Two versions of IBACoP were configured for each track. The first version, IBACoP, was a static sequential portfolio. It was configured by a two step strategy. First, a Pareto efficiency analysis (Censor, 1977) was applied to all the participant planners of the satisficing track of IPC 2011, plus LPG-TD using the whole collection of instances defined for that track. The analysis considered the quality score of the best plan generated and the time required to generate the first solution for each training instance. It resulted in a set of 12 planners, where each dominated all others in at least one training domain. Second, the portfolio configuration strategy applied the uniform method to the resulting set of planners. The order of the component planners in the portfolio was arbitrary. On the other hand, a variation of the IBACoP portfolio was made to participate in the agile track. The configuration strategy allotted to each component planners its average time to solve training instances within 300 seconds instead of using the uniform method. Since the time available to solve each problem in the agile track was equal to 300, not all the component planners were executed in the portfolio. The resulting portfolio was ordered by the time allotted to each planner in ascending order and the planners were executed in sequence until the total time was exceeded.

IBACoP 2 was a dynamic sequential portfolio configured with the methodology described in Section 2.2.1.7 with a few improvements. It considered the set of planners selected by the Pareto analysis performed by IBACoP instead of all the participant planners of the satisficing track of the IPC 2011. Also, the set of training instances used to learn the predictive models was extended. In particular, it considered all the available planning tasks from the IPC 2005 to IPC 2011 excluding those instances which were not solved by any planner. The set of features extracted from the problem and domain definitions and from the SAS$^+$ formulation and its induced graphs was also improved. Moreover, the authors used Random Forests (Breiman, 2001) to generate the classification model, since it achieved a 99.83% of accuracy in the training instances. Finally, the *Best 5 Confidence* (see Section 2.2.1.7) strategy was selected to generate a portfolio configuration for each input problem.

Two sequential dynamic portfolios, termed LIBACoP and LIBACoP 2 (Cenamor et al., 2014) were also configured for the IPC 2014 learning track following the methodology used by IBACoP with two differences. First, the Pareto analysis was applied to all the participant planners from the satisficing track of the IPC 2011 plus LPG-TN and SGPLAN. This analysis selected a subset of 15 planners. Second, since both versions were derived for the learning track, the predictive models were learned for each domain in which both versions should be evaluated. The authors used Random Forests in the classification task and Decision Tables (Kohavi, 1995) to predict time. LIBACoP applied the *Best 5 Confidence* strategy to generate a portfolio configuration for each input planning task in a particular planning domain. LIBACoP 2 used the same strategy to select the 5 planners to be part in the portfolio. However, it used the regression model to assign the runtime to each planner instead of the uniform method. The base configuration of both versions (i.e. the configuration of the portfolios without exploiting domain knowledge) resulted from applying the uniform method to the set of planners selected by the Pareto analysis.

2.2.1.11 CEDALION

The work by Seipp *et al.* (Seipp et al., 2015) described CEDALION, a new portfolio configuration approach which addressed the solver configuration problem in the automated process of configuring sequential portfolios. The classical algorithm configuration problem aims to find a configuration of a parameterized algorithm such that it maximizes a performance metric on a given benchmark. However, CEDALION made the time allotted to each planner in a sequential portfolio part of the solver configuration space. Thereby, it used an algorithm configuration method to greedily generate the portfolio configuration such that each component added to the portfolio maximizes the portfolio performance per additional time spent. Moreover, CEDALION provided theoretical performance guarantees.

CEDALION defined an iterative method to generate sequential portfolios for a given highly parameterized planning algorithm, a training benchmark, a performance metric and a time limit. It starts with an empty portfolio. In each iteration, it uses an algorithm configuration method to generate a new solver configuration and its allotted time with the goal of improving the performance of the current portfolio. If the portfolio obtained by adding the new solver configuration does not improve the performance, CEDALION terminates. Otherwise, the new configuration and its allotted time are added to the current portfolio. Also, all the training tasks solved by the current portfolio with the best quality regarding the given performance metric are removed from the training benchmark. CEDALION iterates until the total time exceeds the given time limit.

In the empirical evaluation, CEDALION used SMAC (Hutter et al., 2011), the sequential model-based algorithm configuration method that uses predictive models to guide the search in the parameter configuration. Also, the authors restricted the solver configuration space to be the same of the Fast Downward planning system defined in (Fawcett et al., 2011) with the aim of fairly assessing CEDALION against FDSS, FD-AUTOTUNE and the work (Seipp et al., 2012) (see Section 2.2.1.5). Moreover, the authors generated the training and test sets from the collection of planning tasks defined in the IPC 2011. It results in two different sets (without overlapping) which contains instances from all the planning domains defined in the competition. Therefore, the experiments only evaluated the generalization capability of the portfolios to new problems of known domains. On the other hand, all the generated portfolios allowed their component planners to share the best cost found so far among them. The results of the empirical evaluation using the satisficing, optimal, agile and learning settings showed that the portfolios generated by CEDALION outperform (in some cases) the state-of-the-art Fast Downward portfolios.

2.2.1.12 The work by Rizzini *et al.*

The work by Rizzini *et al.* (Rizzini et al., 2015) described four techniques to configure dynamic sequential portfolios of domain-independent planners for optimal planning. These techniques were based on planner performance from training executions and the set of 311 instance features described in (Fawcett et al., 2014). The planner performance was measured by the time required to generate optimal solutions. However, if a planner did not solve an instance within the time limit, its performance was defined by the Penalized Average Runtime score which determines a penalty equal to ten times the time limit (PAR10). On the other hand, all the portfolios derived by the proposed techniques executed four phases for each instance to be solved. First, the pre-solving phase aimed to solve the easiest instances using a static portfolio defined for 1.11% of the available time. Second, if the input instance was not solved in the previous phase, a feature extraction process was performed. Third, the main phase focused on generating and running the portfolio configuration to solve the input instance. Finally, the backup solving phase, which was executed in case of failure, run a set of planners for the remaining time.

The four configuration approaches presented in this work differed from the technique used in the main phase to generate the portfolio configuration. The similarity-based approaches, *instance-set-core-based* and *weight-based*, computed the Euclidean distance between the input instance and each training problem in the feature space. *Instance-set-core-based* focused on iteratively selecting the planner that maximized the performance of the training instances closest to the input instance (e.g. those training problems whose distance to the given instance was lower than a given threshold). Unlike the previous technique, *weight-based* assigned to each training problem a weight equal to its distance with respect to the input instance. Finally, it iteratively selected the planner that showed the best weighted sum of the PAR10 scores on each training problem. Both techniques allotted to the selected planner a runtime that maximized the ratio of instances solved per runtime spent. Also, they discarded (after each iteration) the training instances solved by the selected planner within its runtime. On the other hand, the iterative model-based approaches, *simplified model-based* and *full model-based*, learned a random decision forest model to predict the next planner that should be executed and a regression forest model to predict its runtime. *Full model-based* also considered a second classification model which was learned including features from the planners that failed solving the input instance.

The proposed techniques were evaluated against two static configuration approaches. A particular version of FDSS that restricted the number of the component planners and the offline greedy portfolio configuration method defined by Streeter *et al.* (see Section 2.2.2.1). The authors also included PLANZILLA in the empirical evaluation, an adaptation of SATZILLA to optimal planning. The evaluation was performed to analyze the generalization capability of the resulting portfolios to new problems from known domains and to new planning domains. In the first scenario, the results showed that only the portfolios derived by the model-based approaches frequently outperform the considered static portfolios. However, it was found in the second scenario that all the techniques proposed in this work derived portfolios which generalized better than PLANZILLA and the static portfolios considered.

2.2.1.13 Summary

This Section summarizes the main differences among the approaches described in the related work for Automated Planning. For each approach, Table 2.2 shows the scheduling strategy (which can be sequential, sequential round robin or parallel), how many components solvers were included in the generated portfolio, the granularity for which the approach generates portfolios and the main technique used to derive portfolios.

For the sake of clarity, all the techniques based on predictive models have been classified as Empirical Performance Models (EPM). Also, Table 2.2 only describes the best technique from each work included in the related work, which has been selected according to empirical results or results from the IPCs.

2.2.2 SAT

Algorithm selection (Rice, 1976) has been shown to be very useful to solve SAT instances. It aims to select the most suitable solver (from a collection of candidates) for each input instance. The portfolio approach based on this problem typically runs all the candidate solvers with a set of training instances gathered from previous SAT competitions. Next, a number of features that describe the structure of each training instance are extracted. Then, the expected time required to solve every training instance by each candidate solver is learned using the instance features (offline phase). Finally, it generates a portfolio of solvers that maximizes the probability that a specific instance is solved (online phase). Originally, only one solver was selected to solve each input instance. However, modern approaches have introduced the pre-solvers and the backup solver. Pre-solvers refer to a particular static portfolio defined for a limited amount of time with the aim of solving the easiest

Approach	Scheduling Strategy	Portfolio generated	#Components	Technique
BUS	Sequential RR	per-instance	Some solvers	EPM (linear regression models)
Roberts *et al.*	Sequential RR	per-instance	Some solvers	EPMs and quartile analysis
PBP	Sequential RR	per-domain	Some solvers	Statistical analysis
FDSS	Sequential	per-several-domains	Some solvers	Greedy algorithm
Seipp *et al.*	Sequential	per-several-domains	Some solvers	Uniform method
ARVANDHERD	Parallel	per-several-domains	Some solvers	Manually configured
IBACOP 2	Sequential	per-instance	Some solvers	EPM (Random Forest) and uniform method
ASAP	Sequential	per-domain	Single solver	Best training performance
AGAP	Sequential	per-domain	Single solver	Best training performance
AllPACA	Sequential	per-instance	Single solver + backup	EPM (Random Forest)
CEDALION	Sequential	per-several-domains	Some solvers	Greedy algorithm using algorithm conf.
Rizzini *et al.*	Sequential	per-instance	Pre. + some solvers + backup	K-NN variation and EPM (Random Forest)

Table 2.2: Summary of the state-of-the-art portfolio approaches for Automated Planning.

test instances, while the backup solver is the one to be executed in case of failure. It is typically the solver which achieved the best performance on the training benchmark.

Among others, there exist approaches that focused either on computing a static schedule of solvers (Núñez et al., 2013; Streeter et al., 2007) or selecting the most promising solver (Kadioglu et al., 2010; Nikolic et al., 2013) to solve each test instance. In this Section, we describe a broad range of the most relevant works about SAT portfolios. For an extensive overview on algorithm selection, we refer to a recent survey (Kotthoff, 2014).

2.2.2.1 The work by Streeter *et al.*

The portfolio considered in the work by Streeter *et al.* (Streeter et al., 2007) interleaved the execution of the component solvers using the preemptive mode (i.e., the ability to stop execution and resume it later on if necessary). Additionally, the total time allocated in the portfolio was not restricted by any threshold. Hence, the overall time allotted to the portfolio was less or equal to the maximum time the authors were willing to spend on any single solver when solving any particular instance multiplied by the number of component solvers.

This work introduced an optimal and greedy algorithms for computing the optimal sequential portfolio for a training benchmark and proved that the problem is NP-complete. The optimal algorithm only worked (in polynomial time) with a small number of candidate solvers k since the optimal portfolio was learned by computing a shortest path in a graph, whose vertices were arranged in a k-dimensional grid. The offline greedy algorithm aimed to maximize the number of instances solved per runtime spent. In each iteration, it added to the portfolio the pair $\langle solver, time\rangle$ that maximized the ratio $\frac{C(solver,t_s)}{time}$, where $C(solver, t_s)$ was the number of training instances solved by *solver* within t_s, the total allotted time to *solver* in the portfolio ($t_s >= time$). Each pair meant the *time* that *solver* should be executed after resuming its execution. At the end of each iteration, the training instances that were already solved by the current portfolio configuration were removed from the training benchmark. The algorithm iterated until the training benchmark was empty. The authors experimented with both algorithms on SAT and planning (among others). The empirical results showed the strength of the portfolio approach for problem solving.

2.2.2.2 SATzilla

SATzilla was a portfolio-based algorithm selection for SAT which won ten medals in the SAT Competitions 2007 and 2009, and also won the SAT Challenge 2012. It was improved over the years (Xu et al., 2008; Xu et al., 2009; Xu et al., 2012b). The most recent version, SATzilla 2012, introduced an algorithm selector based on cost-sensitive classification models. The offline phase of this version can be split into four main steps. First, it selected a set of pre-solvers and their shorter runtimes based on training performance. Second, the backup solver was selected. It was the solver that achieved the best training performance on instances with expensive feature computation time and that were not solved by the pre-solvers. Third, a classification model (decision forest) was learned using features with a low computation cost. This model was aimed to predict whether the computation time required to extract a more comprehensive set of features is lower than or equal to a given threshold. Finally, SATzilla 2012 computed a cost-sensitive classification model (decision forest) for every pair of solvers in the portfolio to predict which performs better on a given instance.

For each input instance, SATzilla predicted whether the feature computation time for the test instance was too costly or not. Then, the pre-solvers were executed with their allotted times. If the test instance was not solved by the pre-solvers, a feature extraction process was performed. Next, SATzilla selected the best candidate solver for the input instance using instance features and the classification models generated in the offline phase. Finally, it executed the selected solver for the remaining time. If the feature computation time was too costly or, in case of failure extracting the features or running the selected solver, the backup solver was executed.

The authors also proposed to use techniques to automatically generate portfolios for analyzing individual solvers (Xu et al., 2012a). They analyzed the contributions of individual SAT solvers measuring their contribution to SATzilla. Interesting conclusions were found, e.g., that the solvers that contributed most to SATzilla were the solvers that exploited novel strategies instead of solvers with best performance.

2.2.2.3 Hydra

Hydra addressed the algorithm configuration and the algorithm selection problem together (Xu et al., 2010). It was an anytime algorithm that automatically built solvers to complement a sequential portfolio. These solvers were generated by applying an algorithm configuration procedure to a highly parameterized algorithm. Specifically, the iterative Hydra algorithm took five inputs: a parameterized algorithm, a training benchmark, an algorithm configuration procedure, a performance metric and a portfolio configuration method based on algorithm selection. It started with an empty set of component solvers. In each iteration, it used the algorithm configuration procedure to generate a new solver that maximized the performance on the training benchmark. For each instance, the performance of the new solver was defined as the best performance (according to the input metric) achieved by itself or by the current set of component solvers. Next, the new solver was added to the set of component solvers and a portfolio was computed. Hydra iterated until a given condition was satisfied. Also, Hydra was able to discard solvers that were added to the set of component solvers on previous iterations.

In the empirical evaluation, Hydra was applied to SAT producing high-performance portfolios using SATzilla (Xu et al., 2008). The set of component solvers was generated by applying the FocusedILS (Hutter et al., 2009) procedure to SATenstein (KhudaBukhsh et al., 2009), a highly parameterized solver that can be configured to instantiate a broad range of high-performance stochastic local search based SAT solvers. The authors assessed Hydra against 17 SAT *challenger* solvers and the best portfolios of those solvers. The results indicated that the portfolios derived by

HYDRA outperformed all *challengers* and achieved at least the same performance of the considered portfolios.

2.2.2.4 ISAC and ISAC+

The work by Kadioglu *et al.* described ISAC, an instance specific algorithm configuration (Kadioglu et al., 2010). It aimed to provide a configuration of a given parameterized algorithm for each instance to be solved. The authors continued the idea of stochastic offline programming described in (Malitsky and Sellmann, 2009). This approach defined an iterative offline method that performed three tasks. First, the k-means clustering technique was applied to the training benchmark using a distance metric in the space of the instance features. Second, a parameter configuration for the given parameterized algorithm was computed for each cluster using local search. Finally, the distance metric in the feature space was adjusted. This metric was based on the difference in performance of solving a training instance from one cluster with a parameter configuration computed for another cluster. The method iterated until the metric did not improve anymore.

The offline phase of ISAC took a highly parameterized algorithm, a training benchmark and a collection of features extracted from each training instance as inputs. Then, the instance features were normalized in the range $[-1, 1]$ by scaling and translating the values of each feature. Next, the g-means cluster algorithm was applied to the training benchmark using the normalized features. This clustering technique automatically determined the number of clusters (Hamerly and Elkan, 2003). Then, the clusters that contained less instances than a manually selected threshold were re-distributed, starting with the smallest cluster. Finally, an algorithm configuration was computed for each cluster using GGA, a gender-based genetic algorithm for the automatic configuration of algorithms (Ansótegui et al., 2009). In addition, ISAC computed a parameter configuration for the entire training benchmark.

For each test instance, ISAC extracted and normalized a collection of features. Next, it determined the cluster that the test instance belonged to. Finally, the parameter configuration computed for that cluster was used to solve the input instance. In case the test instance was not near enough to any cluster, the configuration computed for the entire training benchmark was used. The authors assessed ISAC on SAT, MIP and set covering with remarkable results.

ISAC+ and ISAC+ 2014 won nine tracks in the MaxSat Competitions 2013 and 2014. ISAC+ (Ansótegui et al., 2014) was an improved version of ISAC. Specifically, it used the ISAC offline phase with the aim of generating one parameter configuration for each cluster. However, ISAC+ used the CSHC algorithm selector (see Section 2.2.2.8) to select the best algorithm configuration for each test instance instead of the same clusters generated in the offline phase.

2.2.2.5 ArgoSmArT k-NN

The work by Nikolic *et al.* (Nikolic et al., 2013) introduced ArgoSmArT k-NN, a *simple* algorithm selection portfolio based on the k-nearest neighbours method (Duda et al., 2000). It was considered *simple* for several reasons. First, neither a backup solver nor pre-solvers were considered. Second, it did not predict feature computation time. Third, any knowledge about the structure of the instances families was not assumed in advance. Fourth, the approach used neither feature selection nor feature generation techniques. Finally, ArgoSmArT k-NN was independent of the distribution of the training benchmark.

In the offline phase, ArgoSmArT k-NN computed a PAR10 penalty score for each candidate solver on every training instance. It was computed as the time required to solve each training instance (within the time limit), considering a penalty equal to ten times the time limit for each unsolved instance. Next, the best k value for the k-nearest neighbours technique was selected using the *leave one out* method on the training instances. This method iteratively extracted one instance from the full training benchmark and then it attempted to solve that instance using the rest of instances as training benchmark. In the online phase, a subset of the features defined by SATZILLA was extracted from the input instance. In particular, ArgoSmArT k-NN only considered 29 features with a low computation cost. Next, it selected the k training instances closest to the test instance in the feature space. The distance between instances was computed (without feature normalization) using the distance measure defined in (Nikolic et al., 2009). Finally, the solver with the lowest penalty value on the selected instances was executed with the input instance. Tie-breaking was resolved by selecting the solver that achieved the best performance over the entire training benchmark.

The authors performed an empirical evaluation with the aim of fairly assessing ArgoSmArT k-NN against SATZILLA 2009 on the instances defined for the SAT Competition 2009. The same training benchmark and the same 13 candidate solvers considered by SATZILLA were used to configure ArgoSmArT k-NN. The results indicated that ArgoSmArT k-NN (with both k values used, $k = 1$ and $k = 9$) outperformed SATZILLA.

2.2.2.6 Satisfiability Solver Selector

Satisfiability Solver Selector (3S) (Malitsky et al., 2012b; Kadioglu et al., 2011) was a dynamic sequential portfolio that won seven medals in the SAT Competition 2011. It scheduled a set of candidate solvers using algorithm selection with a fixed-split solver schedule approach. The offline phase of the competition version was composed of three tasks. First, 3S extracted a collection of 48 features from each training instance and run the 38 candidate solvers with all the 6667 training instances. The second phase aimed to compute a desirable size k of the local neighbourhood for a given instance using cross validation. Finally, the third phase derived the pre-solver schedule by solving a MIP task to compute the optimal sequential schedule of candidate solvers that maximizes the number of solved training instances using 10% of the available time. The MIP task was solved using the column generation approach. Therefore, the solutions found were not optimal though they were near-optimal in practice. In the online phase, a collection of features was extracted from the input instance. Then, the instance features were normalized. Next, 3S selected the subset of k most similar training instances to the input one using the Euclidean distance in the feature space. The best candidate solver for these k instances was selected. Finally, 3S executed the fixed schedule of candidate solvers for 10% of the available time and the selected solver for the remaining time.

3S was generalized to the parallel case and parallel solvers were considered as candidate components of the portfolio (Malitsky et al., 2012a). The authors generalized the MIP task considering the processor where each solver should be executed such that the total makespan should be within the time limit. Also, the MIP task restricted the number of solvers included in the resulting schedules. The parallel version of 3S required to solve the MIP task in both (offline and online) phases. First, the MIP task was solved in the offline phase to compute the pre-solver schedule considering all the training instances for 10% of the available time. Second, it was solved at runtime to compute the parallel schedule that maximizes the number of problems solved on the set of the k training instances closest to the given instance. The performance of the parallel version of 3S was assessed and it was found that this dynamic parallel portfolio significantly increases the ability to solve SAT instances.

2.2.2.7 The work by Hutter *et al.*

The work by Hutter *et al.* (Hutter et al., 2014) focused on empirical performance models for predicting runtime on SAT, MIP and the Travelling Salesperson Problem (TSP). First, the authors aimed to improve the runtime prediction accuracy for parameterized algorithms. Specifically, they described a new method that aimed to encode categorical parameters as real valued parameters. Thus, all existing models could be extended to manage those inputs. Also, new techniques based on Random Forests and approximate gaussian processes were introduced. Second, the authors focused on defining new relevant features for SAT, MIP and TSP problems. Third, an extensive empirical evaluation was performed to analyze the performance of previous modeling approaches and the proposed techniques on three different problems: to predict the runtime of standard solvers on unseen instances (with their default configurations), to predict the runtime of new configurations of a parameterized solver on a particular instance, and both problems together. The results showed that the best predictions on each of the three aforementioned problems were achieved by the techniques introduced in their work. Finally, the authors improved the Random Forest based technique by using statistical methods to manage better the data obtained from executions that terminated prematurely.

2.2.2.8 CSHC PAR8

CSHC PAR8 (Malitsky et al., 2013c) was the winner of the open track of the SAT Competition 2013, the track that was specifically defined for portfolio approaches (*alternative approaches*, see Section 2.1.4.1). It was an 8-core parallel dynamic portfolio. Specifically, it always run LINGELING 587 (Biere, 2011) on four core processors, CCASAT (Cai and Su, 2012) on one core, and three different versions of the sequential dynamic portfolio CSHC on one core each. Each version only differed from the category of the training benchmark used for its configuration (e.g. instances from the industrial, crafted or random categories). CSHC was based on algorithm selection. It used the same pre-solver scheduler technique defined by 3S (see Section 2.2.2.6) and multi-class classification models (based on cost-sensitive hierarchical clustering) to select the best solver for each given instance. CSHC generated several classification models with the aim of improving stability. Since there were a number of models to predict the best solver for each test instance, it used a Penalized Average Runtime technique (PAR10) to aggregate the different classification information. In a nutshell, it gathered from each model the training instances of the cluster that the test instance belongs to. Then, it selected the solver that achieved the best average runtime (with timeouts penalized as ten times the time limit) on the set composed of all the gathered instances.

The offline phase of CSHC aimed to generate the cost-sensitive multi-class classification models. In the competition, CSHC considered two different sets of features. Thus, in the online phase, it attempted to extract the first set of features for a time limit of 400 seconds. In case of exceeding the time limit, it tried to compute the second collection of features for 100 seconds. In case of failure, it executed a manually selected backup solver. Otherwise, CSHC executed the 3S pre-solver schedule for 10% of the available time and the predicted solver for the remaining time. Moreover, in case of detecting a low confidence in the prediction, CSHC executed a recourse action (Malitsky et al., 2013b). In particular, it increased the time allotted to the pre-solvers.

The learning method termed cost-sensitive hierarchical clustering was also defined by the authors of CSHC (Malitsky et al., 2013a). They empirically showed that CSHC was less sensitive to the features with little predictive power than 3S. Also, it was found that the time required to configure a CSHC portfolio was orders of magnitude lower than SATZILLA 2012. Moreover, an empirical assessment indicated that the CSHC portfolios outperformed both 3S and SATZILLA 2012 on different SAT and MaxSAT benchmarks.

2.2.2.9 AutoFolio

The work by Lindauer *et al.* described AutoFolio (Lindauer et al., 2015), a portfolio approach that applied algorithm configuration to a highly parameterized algorithm selection framework. Specifically, it used the algorithm configuration method termed SMAC (Hutter et al., 2011) in order to automatically configure CLASPFOLIO 2 (Hoos et al., 2014) for a given algorithm selection scenario. CLASPFOLIO 2 was a portfolio framework which implemented several algorithm selection approaches and the parameters of the respective machine learning techniques. It took as input an algorithm selection scenario, which was composed of performance data, instance features, a set of solvers and a set of instances among other data.

The configuration space considered by AutoFolio was mainly defined by three groups of parameters. First, the group of parameters which is composed of those that set the algorithm selection approach, the machine learning technique and the configuration of that learning technique. It provided at least three machine learning techniques for each of the six algorithm selection approaches considered. Second, the collection of parameters which define the preprocessing techniques to be used. It supported three techniques to preprocess the performance data and four methods for feature preprocessing. Third, it defined several parameters related to the pre-solving schedules such as the number of pre-solvers considered and the time allotted for presolving. The authors evaluated AutoFolio on 13 algorithm selection scenarios from the Algorithm Selection Library (Bischl et al., 2015). These scenarios included several problem solving tasks like SAT, MaxSAT, CSP and ASP among others. The empirical results indicated that AutoFolio improved the performance on 7 scenarios while matching the performance of the previous state-of-the-art approaches on the remaining scenarios.

2.2.2.10 Summary

This Section concludes the related work for SAT summarizing the main differences among the state-of-the-art approaches. For each approach, Table 2.3 shows the scheduling strategy (which can be sequential, sequential round robin or parallel), how many components solvers were included in the generated portfolio, the granularity for which the approach generates portfolios and the main technique used to derive them.

Table 2.3 describes the greedy algorithm proposed by Streeter *et al.* since their optimal algorithm only worked with a small number of candidate solvers. On the other hand, the work by Hutter *et al.* focused on predictive models and instance features so that no portfolio is generated. Therefore, this work has been excluded from the summary. In general, this summary only includes the most representative approach from each work described in the related work.

Approach	Scheduling Strategy	Portfolio generated	#Components	Technique
Streeter *et al.*	Sequential RR	per-several-instances	Some solvers	Greedy algorithm
SATzilla 2012	Sequential	per-instance	Pre. + single solver + backup	EPM (CS Random Forest)
Hydra	Sequential	per-instance	Pre. + single solver + backup	SATzilla and algorithm conf. (greedy)
ISAC+	Sequential	per-instance	Single solver	CSHC and algorithm conf. (*g*-means)
ArgoSmArT k-NN	Sequential	per-instance	Single solver	K-NN
3S (SC 2011)	Sequential	per-instance	Pre. + single solver	MIP (CG) and K-NN
CSHC PAR8	Parallel	per-instance	Pre. + single solver + backup	EPM (CSHC) and 3S MIP
AutoFolio	Sequential	per-instance	Pre. + single solver + backup	Algorithm conf. to portfolio framework

Table 2.3: Summary of the state-of-the-art portfolio approaches for SAT.

2.2.3 Portfolio Approaches for other Problem Solving Tasks

The portfolio approach has been successfully applied to several areas of problem solving. Thus, in this Section, we describe some relevant works from CSP and ASP, two prominent and widely studied fields.

2.2.3.1 The work by Gomes *et al.*

The work by Gomes *et al.* (Gomes and Selman, 2001) was one of the first works on analyzing the effectiveness of the algorithm portfolio approach applied to problem solving. It considered portfolios composed of stochastic algorithms for solving hard combinatorial search problems. In particular, the authors focused on Constraint Satisfaction Problems and Mixed-Integer Programming problems. Also, three different strategies for running portfolios were analyzed without allowing the component algorithms to share information among them. First, the algorithms included in a portfolio were concurrently executed on a parallel machine. Second, all the component algorithms were executed on a single core processor by interleaving their execution (like in a round-robin strategy). Finally, the portfolio executed the same stochastic algorithm with short timeouts using different random seeds. The results showed that the last strategy outperforms others when only one core processor was available. Moreover, the authors discussed various theoretical results concerning optimal portfolios. In addition, they provided results of the computational advantage of the portfolio approach on hard combinatorial search and reasoning problems.

2.2.3.2 CPHYDRA

An example of optimal sequential portfolio was CPHYDRA (O'Mahony et al., 2008), the winner of the CSP Competition 2008. It won four out of the five categories defined in the competition and was the second best participant in the fifth one. CPHYDRA was based on Case-Based Reasoning, a lazy machine learning approach that uses past experiences to solve new problems. These experiences, termed *cases*, contain a description of a past execution and its respective solution. For each instance to be solved, this approach selects similar cases from the case base (e.g., the whole collection of cases) and use it to solve the input instance.

CPHYDRA built a case base of problem solving experiences. The description of each case was composed of a collection of features that describe a training instance. On the other hand, the solution of every case was defined by the time required by each candidate solver to solve the training instance. For each test instance, CPHYDRA extracted a collection of features. Then, the most similar k cases to the input instance were selected from the case base using the k-nearest neighbours technique. In particular, k was set to 10 and the Euclidean distance was used to measure the similarity between cases. Next, CPHYDRA used a constraint model based on the knapsack problem to compute the portfolio configuration schedule for the k similar cases. The authors weighted the cases in the objective function according to their distance to the input instance. The constraint problem was solved by a simple complete search procedure. In case of detecting that the generated portfolio was useless, CPHYDRA applied an alternative procedure to discard dominated solvers and to distribute the remaining time. Finally, the resulting portfolio configuration was executed to solve the test instance.

2.2.3.3 ASPEED

Another optimal approach for configuring (sequential and parallel) portfolios was called ASPEED (Hoos et al., 2015). It formulated the problem of computing the optimal static portfolio according to a set of training data as a multi-criteria optimization problem using Answer-Set Programming (ASP). The approach was split into two steps. The first step computed the optimal portfolio minimizing the L^2-norm on the vector defined by the time allotted to each solver. This norm led to a significant reduction of candidate portfolios and it also resulted in portfolios with a more homogeneous distribution of time slices. The second step aimed to sort the component solvers in the portfolio with the purpose of minimizing the total execution time in the training data. Therefore, the resulting portfolio achieved the best performance in the training set and minimized the execution time in that instance set. The authors applied ASPEED on ASP, CSP, MaxSAT, QBF and SAT with successful results.

2.2.3.4 CLASPFOLIO 1

CLASPFOLIO 1 (Gebser et al., 2011) was an algorithm selection portfolio that won the NP track of the ASP Competition 2011. It was inspired by an earlier version of SATZILLA. However, the approach relied on Support Vector Regression (Basak et al., 2007) and several manually selected configurations of the CLASP solver (Gebser et al., 2007) instead of Ridge Regression and a set of different candidate solvers. In the offline phase, CLASPFOLIO 1 aimed to generate predictive models using the aforementioned machine learning method. The performance of every configuration on each training instance was computed as $\frac{t^*(i)}{t_k(i)}$, where $t_k(i)$ was the time required by the configuration k to solve the instance i and $t^*(i)$ was the minimum time required by any of the considered configurations to solve the same instance. In the online phase, CLASPFOLIO 1 performed four tasks. First, a logic program was instantiated by the ASP grounder GRINGO. Second, CLASPRE, a light-weight version of CLASP, was used to collect a set of features from the input instance and to solve the easiest test instances. Third, in case the test instance was not solved by CLASPRE, the learned models were used to predict the performance of each configuration. Finally, the configuration with the highest predicted performance was selected to solve the input instance.

2.2.3.5 Summary

This Section concludes the analysis of the state-of-the-art with a summary of some representative approaches for CSP and ASP. Similar to previous summaries, for each approach, Table 2.4 shows the scheduling strategy (which can be sequential, sequential round robin or parallel), how many components solvers run the generated portfolio, the granularity for which the approach generates portfolios and the main technique used to derive portfolios.

The work by Gomes *et al.* has been excluded from the summary because it does not propose techniques to generate portfolios. This work analyzes the effectiveness of the algorithm portfolio approach and proposes different strategies to execute the component solvers, among other contributions.

Approach	Scheduling Strategy	Portfolio generated	#Components	Technique
CPHYDRA	Sequential	per-instance	Some solvers	Case-Based Reasoning and Constraint Programming
ASPEED	Sequential	per-several-instances	Some solvers	ASP
CLASPFOLIO 1	Sequential	per-instance	Single solver	EPM (Support Vector Regression)

Table 2.4: Summary of some representative portfolio approaches for CSP and ASP.

Chapter 3

Automatic Construction of Sequential Portfolios

The study of the state-of-the-art shows the portfolio approach as a promising avenue. This approach exploits the complementary strengths of different solvers in several ways. We focus on static sequential portfolios (see Section 2.1.1, page 5). In particular, we propose to derive the best achievable performance for a given benchmark with a linear combination of candidate domain-independent solvers solving a MIP task. To do so, we define an objective function which consists of a weighted combination of quality score and runtime to assess the performance of solvers. This metric is then used to compute the best portfolio with respect to the selected combination of parameters and performance criteria.

Specifically, in this Chapter, we present a new approach (termed GOP— Generation of Optimal Portfolios) which derives the optimal static sequential portfolio (to be denoted as OSS portfolio) for a specific metric and a given training set; i.e., the optimal combination of solvers for a particular performance criteria with regard to the set of candidate solvers and the training benchmark considered. Actually, the resulting portfolio defines an upper bound on the solvers performance for the given training data set. Using this upper bound, the performance of any solver can be analyzed since it shows how far a solver is from the best performance achievable with a linear combination of solvers for the instances set. Also, this approach helps better understanding the performance of new solvers with respect to existing systems. It is important to remark that optimality is only guaranteed for the given training set; empirically, we show that the resulting portfolios also perform very well on unseen instances.

Additionally, we have studied the *utility* of the training instances since most approaches configure portfolios using all the available instances. We revisit in this Chapter the well known problem in the machine learning literature of the impact of training instances in the result. We study the convenience of using all training instances, and show that using a smaller training set our approach generates portfolios whose performance is equivalent to the one obtained by using all instances. These results could lead in the future to more efficient ways of selecting training instances and generating portfolios.

The Chapter is organized as follows. First, Section 3.1 describes GOP. Next, Section 3.2 presents the utility analysis of the training instances. Then, in Section 3.3, GOP is empirically assessed on Automated Planning and SAT. Finally, Section 3.4 concludes with a summary and Section 3.5 shows the list of the published works related to this Chapter.

3.1 GOP: Automatically Generating Optimal Portfolios

GOP automatically derives an optimal combination of solvers with regard to the set of candidate solvers and the set of training problems. By optimal we mean that it is guaranteed to provide the best quality (in terms of the selection of weights discussed below) when running over the same set of training instances, and in sequence without any exchange of information among them —so that the followed order is not relevant.

GOP consists of three steps. First, every candidate solver is executed with every training problem from the input set to generate raw data. Second, raw data are processed to compute the parameters of the MIP model. Third, the MIP task generates the optimal configuration of a static sequential portfolio for the collection of training problems and candidate solvers. This optimal configuration is the best linear combination of the candidate solvers with regard to the objective function (maximize coverage or quality score or minimize time) of the MIP model. Each task is described in detail next.

3.1.1 Inputs to GOP

The inputs to GOP consist of a set of candidate solvers, S, a set of training instances, I, and the available time to solve each instance T. The set of training instances I can be split into subsets called domains or categories. Each candidate solver $s \in S$ is executed with every training instance $i \in I$ to obtain the set R_{si} of solutions. Each solution $r \in R_{si}$ stores a pair $\langle cost_{sir}, runtime_{sir} \rangle$, where $cost_{sir}$ is the cost of the corresponding solution and $runtime_{sir}$ is a timestamp with the time required to find it. In case of solving problems optimally or SAT problems, a candidate solver $s \in S$ generates at most one solution $r \in R_{si}$ for each training instance $i \in I$, since r should be optimal (as in the case of Automated Planning) or because it is required only to find a single solution —as in the case of SAT. Otherwise, the candidate solver can generate an arbitrary number of solutions.

Each execution generates raw data such as the runtime of each solution, or the memory consumption at each time tick. However, we only need a subset of the raw data, which shall be computed to generate the input data for the MIP model.

3.1.2 MIP Model Input Data

GOP processes the runtime spent (in seconds) and the score of each solution $r \in R_{si}$ for each training instance $i \in I$ and candidate solver $s \in S$. The quality score of each solution, denoted as $q(s, i, r)$, is computed according to the official metric used in the IPC since 2008 (see Equation (2.1) on page 12). This equation computes the quality score in the interval $[0, 1]$. In case of solving problems optimally in planning or in SAT tasks, the value 1 means that the training problem $i \in I$ has been solved by the solver $s \in S$. Otherwise, the value 1 means that the solver $s \in S$ has found the best known solution.

Runtime is computed using Equation (3.1). This equation already takes into account whether every training problem has been solved or not. The runtime value of each solution, denoted as $rt(s, i, r)$, is normalized in the interval $[0, 1]$ if the solver s solves the training problem i. Otherwise, the runtime value is higher than 1 with the objective that the MIP discards solution r of solver s for instance i:

$$rt(s,i,r) = \begin{cases} \frac{runtime_{sir}}{T}, & \text{if } i \text{ is solved by } s \text{ within the time bound } T \\ \frac{T+1}{T}, & \text{otherwise} \end{cases} \tag{3.1}$$

3.1.3 Mixed-Integer Programming Model

Linear Programming is a mathematical technique focused on solving combinatorial problems where a linear objective function should be maximized (or minimized). These optimization problems are subject to some linear constraints. The value resulting from evaluating the objective function over a solution assesses its quality.

MIP is Linear Programming where some variables are constrained to be integers. The MIP model defined by GOP aims to assign a candidate solver $s \in S$ and a solution $r \in R_{si}$ from that candidate solver s (in case of multiple solutions) to each training instance $i \in I$ such that this assignment maximizes its objective function. The outcome of the MIP is the runtime allocated to each candidate solver $s \in S$. This runtime is the time required for the solver s to solve each training instance $i \in I$ with the solution $r \in R_{si}$ selected by the MIP.

The MIP model should define the following elements: parameters, decision variables, objective function and constraints. The model input data is stored as two parameters:

$q(s, i, r)$ Normalized plan quality score (see Equation (2.1) on page 12) for the solution $r \in R_{si}$ found by the candidate solver $s \in S$ for the training problem $i \in I$.

$rt(s, i, r)$ Normalized runtime (see Equation (3.1)) spent by the candidate solver $s \in S$ to solve the training problem $i \in I$ with the solution $r \in R_{si}$.

Decision variables store the outcome of the MIP solver which serve to fully characterize the resulting portfolio:

$solved\_by_{sir}$ is an auxiliary variable that stores each decision made for the model to solve each training instance $i \in I$. If the solution $r \in R_{si}$ of the candidate solver $s \in S$ is selected to solve the instance $i \in I$, the variable takes the value 1. Otherwise, it takes the value 0.

$quality_i$ is an auxiliary variable that stores the quality score of the solution selected to solve the training instance $i \in I$.

$time_s$ is the output variable. It stores the allotted time to each candidate solver $s \in S$ in the interval $[0, 1]$.

Constraints are used to model the availability of computational resources, mainly time. First, the sequential execution of all solvers should not exceed the current available time. Since the time required to find every solution is normalized in the interval $[0, 1]$, this constraint is defined as shown in Equation (3.2):

$$\sum_{s \in S} time_s \leq 1 \tag{3.2}$$

On the other hand, each candidate solver $s \in S$ can be assigned by the MIP to solve a subset of training instances $i \in I$, each one with a particular solution $r \in R_{si}$. The time required for the solver s to solve every assigned instance $i \in I$ is the largest execution time that the solver s takes to solve all problems assigned to it. However, this value can not be computed with a linear expression. Therefore, the next constraint is added to guarantee that the allotted time to each solver is equal to or higher than the required time:

$$time_s \geq solved\_by_{sir} \cdot rt(s, i, r), \forall s \in S, i \in I, r \in R_{si} \tag{3.3}$$

The final quality score achieved by the resulting portfolio is computed with another constraint as shown in Equation (3.4). Although it does not constrain the model in any particular way, it is defined here to be used in the objective function later:

$$quality_i = \sum_{s \in S} \sum_{r \in R_{si}} solved\_by_{sir} \cdot q(s, i, r), \forall i \in I \tag{3.4}$$

As we have commented before, the MIP only selects one solution $r \in R_{si}$ found by a solver $s \in S$ to solve each particular training problem $i \in I$ even if other solvers solve it as well. This is useful for computing the overall time used and the total quality score achieved by the resulting portfolio as shown in Equations (3.3) and (3.4), where $solved\_by_{sir}$ is used. Constraint (3.5) is used to enforce the selection of a single solution per training problem.

$$\sum_{s \in S} \sum_{r \in R_{si}} solved\_by_{sir} <= 1, \forall i \in I \tag{3.5}$$

Although the usual goal for configuring portfolios is to maximize the total quality score or the number of solved instances, we consider an objective function that maximizes a weighted sum of overall running time (the sum of $time_s$ for all solvers s) and quality score of the solutions ($quality$).[1] Since runtime should be minimized, the values computed in the MIP model are substracted from 1 as follows:

$$\begin{aligned} maximize: \quad & w_1(\textstyle\sum_{i \in I} quality_i) \\ + \quad & w_2(1 - \textstyle\sum_{s \in S} time_s) \end{aligned}$$

If the objective of the MIP task only consists of optimizing a single value (such as quality score), it just suffices to set the corresponding weight to 1 while setting the rest of weights to zero — e. g., when optimizing only quality score, the following weights should be used directly: $w_1 = 1$, $w_2 = 0$. If, on the other hand, the objective is to maximize a non-null linear combination of the two aforementioned values, the problem becomes harder. Instead of facing this task as a multi-objective optimization problem, we just solve the MIP task in two steps while preserving the value of the objective function from the MIP solution. For example, if the objective is to maximize the quality score while minimizing the overall running time, then $w_1 = 1$ and $w_2 = 0$ so that only the quality score is taken into account. If one solution exists at least, then a second execution of the MIP model is issued to find the combination of candidate solvers that achieves the same total quality score (denoted as Q) while minimizing the overall running time, just by setting $w_1 = 0$ and $w_2 = 1$. To enforce a solution with the same quality, an additional constraint is added: $\sum_{i \in I} quality_i \geq Q - \epsilon$, where ϵ is just any small real value used to avoid floating-point errors. Clearly, a solution is guaranteed to exist, since a first solution was already found in the previous step.

Algorithm 1 shows the steps followed where the quality score was maximized first, and then running time was minimized among the combinations that achieved the optimal quality. Note that this is different than solving the *multi-objective* optimization problem posed by finding the optimal configuration that *simultaneously* optimizes resources. We have used this algorithm since it was empirically found that the MIP solver tends to distribute all the available time among the candidate solvers selected to be part in the portfolio. Running the procedure depicted in Algorithm 1, it is possible to have some slack time which will be distributed uniformly among the selected solvers.

[1]The MIP model defined in this Section is flexible. It allows us to add in the future other resources, like memory consumption.

Algorithm 1 Build a portfolio optimizing quality score and time

```
Input: Candidate solvers $S$, training instances $I$ and solutions sets $R_{si}$
Output: Output variables $\forall s \in S\ time_s$
  $model_1$ := generateMIPmodel($S$, $I$, $R_{si}$)
  $solution_1$,$Q_1$ := solveMIP($model_1$, $w_1 = 1$, $w_2 = 0$)
  if a solution exists then
    $model_2$ := addConstraint($model_1$, $\sum_{i \in I} quality_i \geq Q_1 - 0.001$)
    $solution_2$,$Q_2$ := solveMIP($model_2$, $w_1 = 0$, $w_2 = 1$)
    return $solution_2$
  else
    return no solution
  end if
```

3.1.4 GOP Output

The output of the MIP task is just the allocated runtime to each candidate solver, which can be either zero (i.e., the candidate solver should be excluded from the portfolio) or a positive number. The MIP task does not specify any particular order to execute the solvers. The execution sequence of the obtained portfolios is arbitrary and it is based on the order in which the candidate solvers were initially specified.

3.2 Analysis of the Utility of Training Instances

One of the main factors that influences the time required to solve the MIP task defined in the previous section is the size of the training instance set. Thus, in this Section we analyze the question of whether there is a subset of the training problems that results in a portfolio with a similar or even equal performance (measured in total quality score or coverage) with regard to the same set of candidate solvers. We refer to this particular problem as the analysis of the *utility* of the training instances. For example, when optimizing quality score, if all training problems are solved by all solvers with the same score, they do not provide any *utility* to configure the OSS portfolio since any combination of the candidate solvers will provide the same result. Similarly, the training problems that are not solved by any solver are equally irrelevant since the MIP task will not be able to distinguish between an empty portfolio or a particular combination of the candidate solvers.

We propose to split the training problems in subsets that contain those training instances that are solved by a maximum number of solvers. The first set is composed of all instances that are solved by at most one solver. The second set (which is a superset of the previous one) consists of all instances that are solved by at most two solvers, and so on. In general, the i-th set consists of all training instances that are solved by i solvers or less. This analysis aims to study whether the performance of the portfolios obtained with every subset is closer to the performance achieved by the portfolio computed with the full set of training instances. Thus, each resulting portfolio is assessed on a benchmark which contains (among others) the training instances used to compute it. In this analysis, we evaluate the resulting portfolios over the training set because: first, we are assessing neither the proposed portfolio generation technique nor the performance of the resulting portfolio; secondly, if we would evaluate the resulting portfolios over a different (test) set, results would be biased by other factors, such as generalization capability. Thus, by fixing the training set we expect to highlight the differences due solely to the training instance set considered. We report in the experiments section the results of this analysis, which could lead in the future to efficient algorithms to select *a priori* better training instances to generate high performance portfolios.

3.3 Empirical Evaluation

This Section describes the experiments performed in Automated Planning and SAT. The MIP solver selected to solve the MIP task in all experiments has been SCIP[2], one of the fastest non-commercial MIP solvers, which supports the modeling language ZIMPL[3]. Also, the same model described in Section 3.1 has been used to solve all the MIP tasks. Indeed, this model has also been used by MIPLAN (Núñez et al., 2014a), the winner of the learning track of the IPC 2014.

In satisficing planning, the number of solutions found for a particular planning task depends on each satisficing planner. Therefore, the solution quality score should be modeled as a vector instead of a scalar. However, while ZIMPL allows the definition of multi-dimensional parameters, it is not possible to use multi-dimensional *dynamic* parameters —i. e., each with a different size. As a consequence, all parameters were enforced to have the same length equal to the maximum number of solutions found by any planner to any planning task. For those planners that found less solutions, we just used Equations (2.1) and (3.1) (defined on pages 12 and 38 respectively) for introducing additional entries as if they were unsolved. Note that if not all the solutions found in the training data are considered, the portfolio derived by GOP might not be optimal with regard to the set of candidate solvers and the training benchmark considered. It could be possible to find a different static portfolio configuration which achieves a better score on the training set. As an example, consider $T = 1800$ seconds, two candidate solvers, s_1 and s_2, and two training instances, i_1 and i_2, as input data. Let us assume that the performance of these solvers with respect to the set I of the two planning tasks is:

- s_1 only solves i_1 and it finds two solutions, $R_{s_1i_1} \doteq \{r_1, r_2\}$, where $q(s_1, i_1, r_1) = 0.8$ at $rt(s_1, i_1, r_1) = 10$ and $q(s_1, i_1, r_2) = 1.0$ at $rt(s_1, i_1, r_2) = 1700$.

- s_2 only solves i_2 with two solutions $R_{s_2i_2} \doteq \{r_1, r_2\}$, resulting in $q(s_2, i_2, r_1) = 0.8$ at $rt(s_2, i_2, r_1) = 10$ and $q(s_2, i_2, r_2) = 1.0$ at $rt(s_2, i_2, r_2) = 1700$.

Assume further that only the best solution found by each candidate solver is considered. Thus, GOP would derive a portfolio configuration which would only be able to solve one instance with a quality score equal to 1, since the available time to solve each instance is equal to 1800. However, the static portfolio composed of s_1 and s_2 with $t_1 = 10$ and $t_2 = 1700$ would achieve a total score equal to 1.8. Therefore, the portfolio generated by GOP would not be optimal with respect to the whole training data set.

Algorithm 1 has been used to derive every GOP portfolio. This algorithm is a two steps optimization process. The first step maximizes the quality score and the second one aims to minimize the total allotted time while preserving the optimal score. We have empirically observed that (in satisficing planning) the first step is much faster than the second one. Thus, we report both times for all the satisficing experiments as well as for the experiments for optimal planning and SAT in which the computation times are representative.

In this Section we have performed five sets of experiments. First, in Section 3.3.1, the OSS portfolio has been computed for several international problem solving competitions. Second, Section 3.3.2 details the empirical analysis of the training instances to configure sequential portfolios. Third, the quality of the portfolios automatically derived by GOP has been assessed in Section 3.3.3.

[2]http://scip.zib.de/
[3]http://zimpl.zib.de/

Next, Section 3.3.4 describes the experiments in which GOP has been assessed against the state-of-the-art optimal portfolio approaches. Finally, the quality of the solutions achieved by GOP over time has been analyzed in Section 3.3.5.

3.3.1 OSS Portfolio for Problem Solving Competitions

The first experiment aims to compute the OSS portfolio for different problem solving competitions, which allows us to analyze the performance of any solver on those particular benchmarks. Since in this experiment the performance is tested with the same training instances, the resulting portfolio provides us only with the best achievable performance given the participant solvers and instances of a particular competition. Therefore, this experiment does not assess the performance of the OSS portfolio on a different set of instances. It analyzes the performance of the awarded solvers regarding the upper bound defined by the OSS portfolio.

3.3.1.1 Optimal Planning

In this Section we have computed the OSS portfolio for the sequential optimization track of the IPC 2011 considering all the instances defined for that competition (see Table 2.1 on page 15). The set of candidate planners is composed of all participant planners, where the portfolios have been removed from the selection and their solvers have been added instead as shown in Table A.1 (see Annex A on page 93). In particular, the *Merge-and-Shrink* portfolio and the two variants of FDSS were discarded and the following solvers added: two versions of *Merge-and-Shrink* (Nissim et al., 2011) and the A* search algorithm (Hart et al., 1968) with the blind heuristic.

The OSS portfolio for the IPC 2011 has been configured by running Algorithm 1. It took GOP 10.75 seconds to compute it. Figure 3.1 shows the resulting portfolio, which solves 200 training problems. The benchmark used in the IPC 2011 was composed of 280 planning tasks. However, there were 77 problems that were not solved by any planner. Therefore, there are only three solvable instances that the OSS portfolio is not able to solve. Recall that the OSS portfolio is a static portfolio which has a fixed available time T. Hence, it is usually not able to solve all the solvable instances because, in many cases, a static sequential portfolio which solves all the solvable instances does not exist (due to the time constraint).

The metric used in the competition ranks the participant planners according to their performance (coverage). But it does not provide any information about how good the performance of each participant planner is. However, the OSS portfolio computed in this Section shows how far each planner is from the best linear combination of the participant planners.

For instance, the winner of the IPC 2011 was FDSS-1 (which solved 185 problems) and the runner-ups were *Selective Max* and *Merge-and-Shrink* (both planners solved 169 planning tasks). The OSS portfolio automatically built by GOP solves 200 planning problems. This value defines an upper bound for the number of solved problems on this competition taking into account its competing planners and problem instances. Therefore, it shows that the performance of FDSS-1 is $\frac{185}{200} \cdot 100 = 92.5\%$ of the best performance achievable while the performance of the runner-ups is $\frac{169}{200} \cdot 100 = 84.5\%$ with regard to the same upper bound.

Moreover, this OSS portfolio is a reasonable estimator of the expected performance of state-of-the-art planners. It can be used in other competitions as a reference performance to analyze whether the participant planners result in a significant advance in the state-of-the-art or not.

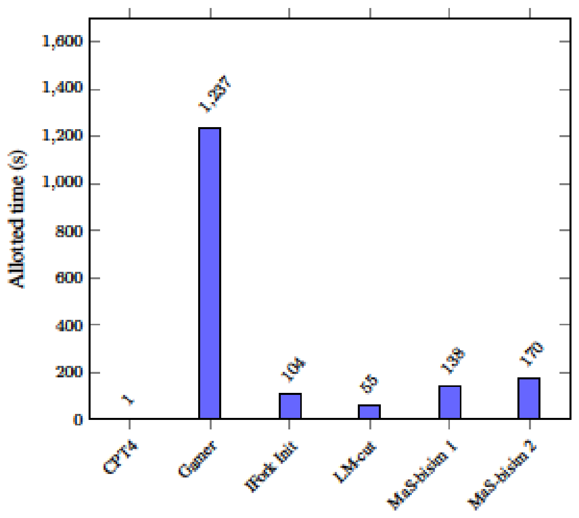


Figure 3.1: OSS portfolio for the sequential optimization track of the IPC 2011.

3.3.1.2 Satisficing Planning

In this experiment, we have used the entire benchmark defined for the IPC 2011 and all the entrants of that competition (for details, see Tables 2.1 and A.3 on pages 15 and 94 respectively). The participant portfolios have been used instead of their component planners since these portfolios implement their own anytime behavior.

Figure 3.2 shows the computed portfolio, which has been configured by running Algorithm 1. It took GOP 382,462.14 seconds, less than 4.5 days (circa 5.5 hours for the first step and about 4.3 days for the second step) to compute it. This portfolio solves 265 training problems with a total quality score of 252.75. Remarkably, 13 problems were not solved by any planner so that the largest number of solvable instances is 267. Therefore, the planner automatically built by GOP falls below only by two instances.

Once the OSS portfolio for the competition has been computed, we are able to analyze the performance of the awarded planners against the upper bound defined by it. The winner of the sequential satisficing track of IPC 2011 was LAMA-2011 (Richter and Westphal, 2010), which solved 250 problems with a total quality score of 216.33. The runner-up was FDSS-1 and it solved 232 problems with a total score of 202.08. These results show that LAMA-2011 is very effective solving planning tasks and finding high quality solutions. The performance of LAMA-2011 measured in coverage is therefore $\frac{250}{265} \cdot 100 = 94.3\%$ while the performance of FDSS-1 is $\frac{232}{265} \cdot 100 = 87.5\%$ with regard to the upper bound defined by the OSS portfolio. Interestingly, LAMA-2011 has more room for improving the quality score since its performance (total score) is equal to $\frac{216.33}{252.75} \cdot 100 = 85.6\%$ according to the total quality score achieved by the OSS portfolio. Nevertheless, the total score achieved by LAMA-2011 is quite high.

3.3.1.3 SAT

The OSS portfolio for the open track of the SAT Competition 2013 has been computed using the results of this competition, removing the disqualified solvers. We have considered all the instances and all the participant solvers (instead of their component solvers) that took part in the open track

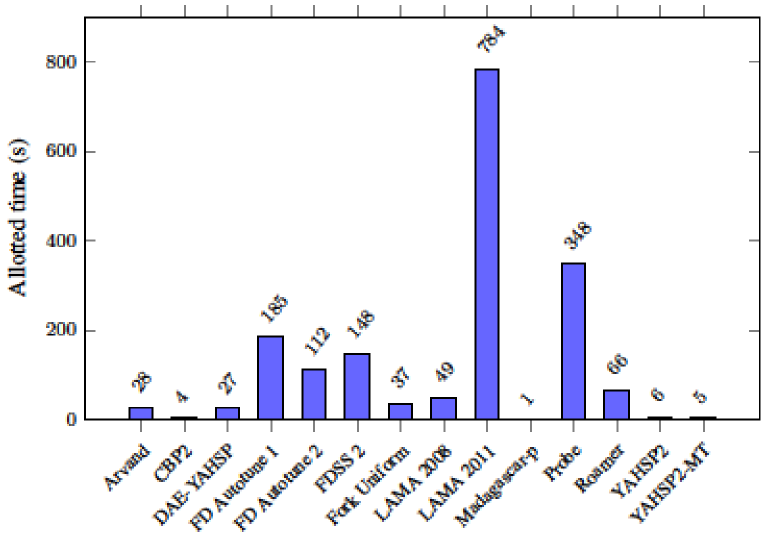


Figure 3.2: OSS portfolio for the sequential satisficing track of the IPC 2011.

(see Tables 2.1 and A.6 on pages 15 and 96 respectively for details).

Algorithm 1 was executed to compute the OSS portfolio, which took GOP 22.06 seconds. The configuration of the resulting portfolio is shown in Figure 3.3. This portfolio solves 234 instances. However, there are 253 instances that were solved by at least one participant solver. Thus, there are 19 solvable instances that the OSS portfolio is not able to solve.

The aim of this experiment is to analyze the performance of the awarded solvers using the upper bound defined by the OSS portfolio. Thus, it shows that the performance of the winner of this track (CSHC PAR8 (Malitsky et al., 2013c)) is $\frac{234}{234} \cdot 100 = 100.0\%$ of the best performance achievable while the performance of MIPSAT (the portfolio automatically configured with GOP that won the silver medal in this track) is $\frac{231}{234} \cdot 100 = 98.71\%$ with regard to the defined upper bound.

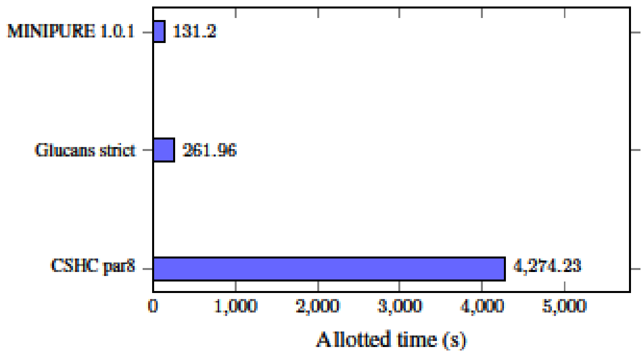


Figure 3.3: OSS portfolio for the open track of the SAT Competition 2013.

3.3.2 Analysis of the Utility of Training Instances

The second experiment aims to empirically analyze the influence of the training instances in the portfolio configuration process, where the utility of every training instance has been considered as discussed in Section 3.2. In particular, this analysis focuses on the influence of the composition and the size of the training set in the quality of the resulting portfolios under the hypothesis that not all instances provide the same information for configuring portfolios.

In a nutshell, this analysis consists of three steps. First, a number of subsets of training instances are defined by splitting the training benchmark as described in Section 3.2. Next, Algorithm 1 is used to derive a sequential portfolio for each subset. Finally, each resulting portfolio is evaluated on the entire training benchmark.

This is an *a posteriori* analysis since it involves the execution time of each solver with every training instance. Also, this analysis does not evaluate GOP. We only use GOP with the aim of empirically analyzing a specific issue in the automated design of portfolios. Moreover, it is important to remark that we only evaluate portfolios over the training set in this empirical analysis.

3.3.2.1 Optimal Planning

In this analysis, we consider the same training data set used in the previous experiment on optimal planning. Namely, the entire benchmark and all the participants from the sequential optimization track of the IPC 2011, discarding portfolios and considering their component planners instead. The results of this analysis are shown in Table 3.1. From the table, it results that the same performance (measured in coverage and time) is achieved by all the sequential portfolios derived using all sets of training problems but the first one (which consists just of those problems that were solved by at most one planner). As it can be seen in Table 3.1, the minimum set of training problems necessary to configure the OSS portfolio for the IPC 2011 is the second one, which contains only 27 problems. This fact empirically confirms our initial intuition: not all training problems provide the same utility.

Planners	Size	Training Score	IPC Score	Computation Time - Step 1	Computation Time - Step 2	Total Time
1	18	17	190	0.01	0.01	1753
2	27	24	200	0.02	0.02	1705
3	29	26	200	0.02	0.02	1705
4	39	36	200	0.03	0.04	1705
5	52	49	200	0.06	0.06	1705
6	58	55	200	0.23	0.07	1705
7	61	58	200	0.14	0.08	1705
8	75	72	200	0.13	0.12	1705
9	91	88	200	0.22	0.18	1705
10	103	100	200	0.12	0.40	1705
11	178	175	200	1.39	1.27	1705
12	203	200	200	9.21	1.54	1705

Table 3.1: Results of the utility analysis of planning tasks for optimal planning. For each subset, identified by the maximum number of planners that solve its instances, the table shows the size of the problem subset, the number of problems solved in the problem subset by the resulting portfolio, the number of problems solved by the resulting portfolio in the IPC 2011, the computation time (seconds) required to obtain the portfolio, and the total allotted time (seconds) in the portfolio.

3.3.2.2 Satisficing Planning

In the analysis for satisficing planning, all the entrants of the IPC 2011 sequential satisficing track and all the planning tasks defined for that competition are considered. The results of this analysis are shown in Table 3.2. As it can be seen, our results indicate that a restricted number of planning instances suffices to derive the OSS portfolio. In fact, the eighth subset achieves the same number of solved problems, though with lower quality score, and the nineteenth set with 226 instances serves to derive the same OSS portfolio that is computed with all planning tasks —280 in total. Remarkably, GOP finds alternative configurations for the fourteenth, fifteenth, and sixteenth sets that solve 266 problems (one less that the maximum feasible) but with lower quality score and that is why that configuration is discarded when considering more planning tasks. The quality score does not grow monotonically as shown in Table 3.2, so we conjecture that the *utility* of planning tasks is not fully captured by our definition of subsets, though it seems to be very accurate.

Planners	Size	Training Score	IPC Score	Computation Time - Step 1	Computation Time - Step 2	Total Time
1	1	1.00/1	65.86/72	0.01	0.01	7
2	3	3.00/3	210.76/236	0.01	0.01	991
3	5	5.00/5	210.21/244	0.02	0.02	1195
4	7	7.00/7	222.04/248	0.04	0.05	1343
5	9	8.69/9	222.90/249	0.06	0.08	1430
6	12	11.58/12	227.83/252	0.13	0.11	1782
7	21	20.15/21	235.35/261	0.62	0.41	1792
8	30	28.27/30	242.57/265	1.10	1.16	1796
9	44	41.56/44	243.03/264	2.55	2.51	1797
10	57	51.77/57	238.23/264	6.47	11.72	1800
11	74	67.02/72	249.73/264	26.30	266.84	1799
12	91	82.51/89	251.89/265	75.26	320.07	1797
13	113	103.16/111	251.84/265	175.40	5,543.47	1800
14	133	122.13/132	252.25/266	312.24	42,111.97	1798
15	158	146.75/157	252.25/266	321.98	49,626.27	1798
16	182	169.92/181	252.25/266	1,046.61	133,542.97	1798
17	204	191.06/202	252.46/265	4,083.19	94,602.84	1799
18	214	200.46/212	252.46/265	8,209.01	946,802.25	1800
19	226	211.89/224	252.75/265	12,186.80	305,287.58	1800
20	231	216.83/229	252.75/265	13,998.41	158,359.14	1800
21	237	222.83/235	252.75/265	16,728.20	196,328.32	1800
22	249	234.80/247	252.75/265	19,448.20	232,121.03	1800
23	253	238.75/251	252.75/265	15,151.51	211,140.53	1800
24	261	246.75/259	252.75/265	12,379.08	256,842.09	1800
25	266	251.75/264	252.75/265	11,835.37	214,669.78	1800
26	266	251.75/264	252.75/265	11,760.60	214,758.56	1800
27	267	252.75/265	252.75/265	14,630.29	370,626.77	1800

Table 3.2: Results of the utility analysis of planning tasks for satisficing planning. For each subset, identified by the maximum number of planners that solve its instances, the table shows the size of the problem subset, the quality score and coverage achieved in the problem subset by the resulting portfolio, the quality score and coverage achieved by the portfolio in the IPC 2011, the computation time (seconds) required to obtain the portfolio, and the total allotted time (seconds) in the portfolio.

3.3.2.3 SAT

The analysis of the training instances has also been applied to SAT. We have retrieved the results of the open track from the SAT Competition 2013 removing the disqualified solvers to perform the empirical analysis. The results are shown in Table 3.3. These results show that the performance achieved by the first portfolio is very high. Indeed, the portfolio derived with the second subset (which is only composed of 37 instances) solves only four instances less than the optimal configuration. The portfolio derived with the sixth subset solves the same number of instances than the OSS portfolio computed in the previous section. However, the number of solved instances does not grow monotonically as it can be seen in Table 3.3. The portfolio configured with the fifth subset solves three instances less than the portfolio derived with the previous subset. Despite of that, the definition of subsets is very accurate for our purpose. Hence, our initial hypothesis is empirically validated since our approach does not need a large training data set to configure high performance portfolios.

Solvers	Size	Training Score	Competition Score	Computation Time - Step 1	Computation Time - Step 2	Total Time
1	10	5	225	0.01	0.01	4431.02
2	37	27	230	0.17	0.04	4651.40
3	68	52	230	0.42	0.51	4651.40
4	81	65	233	0.47	0.23	4893.61
5	94	77	230	0.82	0.88	4651.40
6	106	88	234	1.20	0.93	4837.31
7	116	98	234	1.72	1.39	4837.31
8	151	133	234	1.94	3.16	4837.31
9	189	170	234	3.22	16.40	4667.39
10	253	234	234	6.85	15.21	4667.39

Table 3.3: Results of the utility analysis of training instances for SAT. For each subset, identified by the maximum number of solvers that solve its instances, the table shows the size of the training subset, the number of instances solved in the training subset by the resulting portfolio, the number of instances solved by the resulting portfolio in the SAT Competition 2013, the computation time (seconds) required to obtain the portfolio, and the total allotted time (seconds) in the portfolio.

3.3.3 Assessment of GOP to Configure Sequential Portfolios

The main goal of every portfolio configuration technique is to derive portfolios which achieve the best performance on every test benchmark. Thus, the third experiment aims to evaluate GOP against the state-of-the-art portfolios on several problem solving competitions.

3.3.3.1 Optimal Planning

In this Section, the performance of the sequential portfolios automatically generated by GOP has been evaluated in optimal planning. Specifically, we have assessed GOP against FDSS and we have evaluated GOP on the IPC 2014.

Comparison against Fast Downward Stone Soup The first experiment aims to evaluate GOP against the two versions of FDSS submitted to the IPC 2011. FDSS-1 was the result of applying the portfolio generation technique defined by FDSS to a particular training data (which is described below) while FDSS-2 used the uniform method with a manually selected set of planners. Specifically, FDSS-2 was composed of LM-CUT (Helmert and Domshlak, 2011), BJOLP (Domshlak et al., 2011), the two variants of M&S-bisim (Nissim et al., 2011) and blind search.

Training The set of candidate planners is composed of all the planners considered in the design of FDSS-1[4], which are listed in Table A.2 (see Annex A on page 94). On the other hand, instead of using all the 1163 instances from the IPCs in the range covering the period from 1998 to 2008 (as it was done to configure FDSS-1) we have considered a subset composed of only the 240 planning tasks contained in the benchmark denoted as OPTIMAL IPC DOMAINS 2008 in Table 2.1 (see Section 2.1.5.1 on page 15).

The portfolio generated by running Algorithm 1 is shown in Figure 3.4. The time taken to compute this portfolio was 3.09 seconds. This portfolio, denoted as GOP-1, solves 165 problems in the benchmark defined for the IPC 2008 (training set). However, there are only 167 planning tasks that are solved by some planner considered in the set of candidate planners. Therefore, GOP-1 solves all solvable training tasks but two of them.

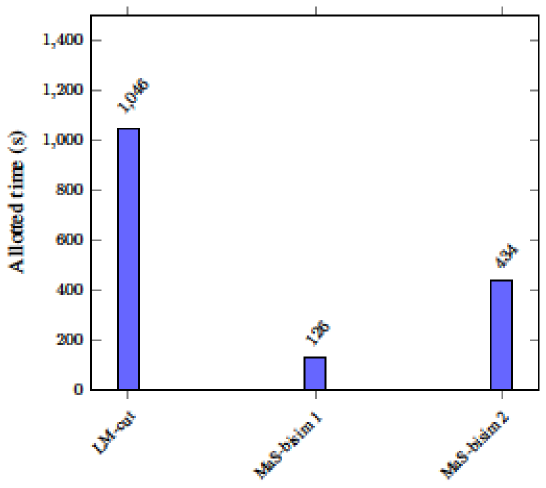


Figure 3.4: Portfolio derived by GOP repeating the experiments of FDSS-1 for optimal planning.

Test All the domains selected in the IPC 2008 are included in the IPC 2011. Hence, in order to test the generalization power of both GOP and FDSS, only the new domains defined in the IPC 2011 have been used (see Table 2.1 on page 15 for details).

GOP-1 solves the same number of problems than FDSS-1 and FDSS-2, 65. Therefore, the sequential portfolio automatically derived by GOP under the same conditions as FDSS, provides the same performance as the winner of the IPC 2011. Also, this result endorses the idea that it is not necessary to use a large number of problems to train a portfolio. Instead, a smaller number of more informative problems can be used. We expect that the time necessary to solve the MIP task is significantly smaller than the time necessary to traverse the state-space of portfolio configurations with a hill-climbing search algorithm as in the case of FDSS. This statement can not be guaranteed because the published experiments of FDSS do not show the time required to compute the portfolios.

[4]The M&S-LFPA algorithms have been considered. However, they failed on all IPC 2008 tasks because they do not support action-costs.

Remarkably, the planners selected for GOP-1 are the same ones as those picked in the configuration of FDSS-1 except for BJOLP.

As discussed above, the metric used in the IPC 2011 does not provide any information about the quality of the achieved performance. Hence, we do not know how good the previous results are. We only know the number of solved problems (65) and the size of the test problems set (120). Therefore, we have computed the OSS portfolio (OSS GOP-1) for the new domains defined in the IPC 2011 using the same set of candidate planners considered by FDSS-1 and GOP-1. Also, we have generated the VBS, the SBS and the OSS portfolio (OSS IPC) for these domains using all the participant solvers in the IPC 2011. The performance of the resulting portfolios is shown in Figure 3.5. As it can be seen, the performance shown by GOP-1 and FDSS is very high since they are equal to $\frac{65}{69} \cdot 100 = 94.2\%$ with regard to the performance of the OSS GOP-1 portfolio. Recall that OSS GOP-1 has been derived using the set of candidate planners considered in the design of FDSS-1 whereas the VBS considers all the participant solvers in the IPC 2011. Thus, in this experiment, OSS GOP-1 is able to solve more instances than VBS.

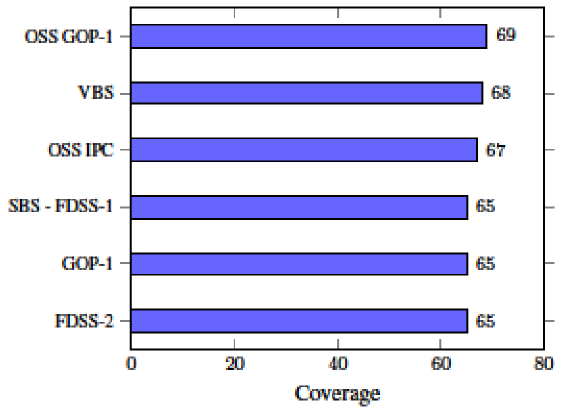


Figure 3.5: Assessment of GOP against FDSS on optimal planning. The figure shows the coverage of VBS, SBS, FDSS and OSS portfolios for all planning tasks defined in the new domains introduced in the IPC 2011 sequential optimization track.

IPC 2014 With the aim of assessing GOP on the IPC 2014, we have generated and submitted two sequential portfolios for the sequential optimization track: MIPLAN and DPMPLAN. The first portfolio, MIPLAN, is the result of applying GOP to the whole collection of planning tasks from the IPC 2011 and a set of candidate planners composed of all the planners considered in the design of FDSS and all the participants in the IPC 2011 (removing portfolios and adding their solvers instead). On the other hand, the idea behind DPMPLAN is to transform the objective function defined by GOP into a temporal objective function so that the MIP task will maximize the original objective function for each instant of time (measured in seconds). For more detailed information, please refer to (Núñez et al., 2014b).

After the competition, we realized that the temporary files were not removed after each execution, as dictated by the competition rules: *"If your planner generates any temporary files, we will automatically clean these up after each planner run, restoring the planner directory to its previous state."*[5] This fact caused that our portfolios achieved a score equal to zero in several planning domains. Therefore, we have re-run the competition under the same empirical conditions defined by

[5]https://helios.hud.ac.uk/scommv/IPC-14/plannersub.html

the organizers. Also, we have used the same pool of instances and the source code of the participant planners which were used to run the original competition (without fixed versions). Specifically, we have executed the five best participant planners according to the official results, MIPLAN and DPMPLAN. We have only executed the five best planners because to the best of our knowledge, this problem only affected the results of our portfolios in the optimization track.

Table 3.4 shows the official competition results and the results obtained by re-running the competition. As it can be seen, the score achieved by our portfolios is more than twice the score obtained in the competition. Specifically, MIPLAN, the $14th$ best planner according to the official results, solves the same number of instances than RIDA (Franco et al., 2014), the fifth classified. Note that the difference in performance between MIPLAN and the four best planners is due to the fact that these planners are a significant improvement in the state-of-the-art of optimal planning prior to the IPC 2014. These participants are not portfolio based approaches.

Participant planner	IPC Official Results	IPC Re-execution Results
SYMBA-2	151	154
SYMBA-1	143	151
CGAMER-BD	120	133
SPM&S	114	126
RIDA	113	117
DYNAMIC-GAMER	99	106
MIPLAN	47	117
DPMPLAN	43	115

Table 3.4: Results of the IPC 2014 sequential optimization track. For each competition results, the table shows the number of problems solved by each participant planner.

3.3.3.2 Satisficing Planning

In this Section, the quality of the sequential portfolios automatically derived with GOP has been assessed against FDSS and the work by Seipp *et al.* under the same conditions. Also, GOP has been empirically evaluated on the IPC 2014.

Comparison against Fast Downward Stone Soup FDSS was designed using a number of heuristics and search algorithms implemented in the Fast Downward planning system (Helmert, 2006). Specifically, it only considered weighted-A* (with a weight of 3) and greedy best-first search, with "eager" (standard) and "lazy" (deferred evaluation) variants of both search algorithms. On the other hand, only four heuristics were considered: additive heuristic ADD (Bonet and Geffner, 2001), FF/additive heuristic FF (Hoffmann and Nebel, 2001; Keyder and Geffner, 2008), causal graph heuristic CG (Helmert, 2004), and context-enhanced additive heuristic CEA (Helmert and Geffner, 2008). Note that FDSS did not consider the landmark heuristic used in LAMA.

Training Two variants of FDSS were considered by its authors. FDSS-1 was configured considering all possible combinations of greedy best-first search and the single-heuristic algorithm for weighted-A* resulting in a total number of 38 configurations, which are described in Table A.4 (see Annex A on page 95). However, FDSS-2 was designed using the different combinations of greedy best-first search with a single heuristic, yielding eight different combinations, as it can be seen in Table A.5 (see Annex A on page 95).

Both variants of FDSS were configured using a very large number of planning tasks: 1116 training instances from all the past IPCs. Instead, we have generated two sequential portfolios applying GOP only over a subset composed of the 270 planning tasks from the sequential satisficing track of the IPC 2008. This training benchmark, denoted as SATISFICING IPC DOMAINS 2008, is detailed in Table 2.1 (see Section 2.1.5.1 on page 15). For the sake of fairness, GOP-1 and GOP-2 have been derived from the same set of candidate planners considered in the design of FDSS-1 and FDSS-2 respectively, with GOP-1 and GOP-2 selecting a different set of candidate planners. The time required to execute each candidate planner with each training instance was 213 computation days, since the time limit to solve each training instance is 30 minutes. All the candidate planners were run with the iterated search of Fast Downward because GOP, unlike FDSS, does not modify the behavior of the portfolio once the first solution is found.

The time required to derive GOP-1 by running Algorithm 1 was 92,914.36 seconds — circa 26 hours (less than 45 minutes for the first step and about 25 hours for the second step). The configuration of the generated portfolio is shown in Table 3.5. This portfolio solves 269 problems with a total quality score of 266.631 in its training data set. The time taken to compute GOP-2 was 151.23 seconds. Table 3.5 shows the resulting portfolio, which solves 269 planning tasks with a total score equal to 267.289. Since both variants of GOP have been configured with different sets of candidate planners, the best solution found for each training problem can be different and thus, the overall performance shown here by both variants of GOP differs.

Component Planners			Allotted Time (s)		
Search	Evaluation	Heuristics	GOP-1	GOP-2	GOP-1L08
Greedy best-first	Eager	FF	317	413	52
Weighted-A* w=3	Lazy	FF	115	0	449
Greedy best-first	Eager	FF, CG	74	0	38
Greedy best-first	Eager	ADD, FF, CG	16	0	16
Greedy best-first	Eager	FF, CG, CEA	27	0	56
Greedy best-first	Eager	FF, CEA	8	0	8
Greedy best-first	Eager	ADD, FF	2	0	2
Greedy best-first	Eager	CG, CEA	28	0	28
Greedy best-first	Eager	ADD, CG	0	0	42
Greedy best-first	Lazy	FF	1	349	1
Greedy best-first	Eager	CEA	1	172	1
Greedy best-first	Eager	ADD, CEA	15	0	15
Greedy best-first	Lazy	FF, CG, CEA	1	0	1
Weighted-A* w=3	Eager	FF	21	0	73
Greedy best-first	Eager	ADD	6	23	0
Greedy best-first	Lazy	ADD, FF, CG	746	0	0
Weighted-A* w=3	Lazy	CEA	12	0	2
Weighted-A* w=3	Eager	CEA	0	0	13
Weighted-A* w=3	Lazy	ADD	4	0	10
Weighted-A* w=3	Eager	ADD	0	0	170
Weighted-A* w=3	Eager	CG	2	0	1
Greedy best-first	Eager	CG	3	108	4
Greedy best-first	Lazy	ADD, CG	0	0	1
Weighted-A* w=3	Lazy	CG	400	0	382
Greedy best-first	Lazy	CG	1	366	1
Greedy best-first	Lazy	CEA	0	358	0
Greedy best-first	Lazy	ADD	0	11	0
LAMA-2008			0	0	434

Table 3.5: Configuration of GOP-1, GOP-2 and GOP-1L08 derived with the set of candidate planners considered for the design of FDSS-1, FDSS-2 and FDSS-1 adding lama-2008, respectively. Each component planner is defined by a search algorithm, an evaluation method and a set of heuristics.

LAMA-2011 uses a combination of landmarks count and FF heuristics that performs very well. However, FDSS considered neither the landmarks count heuristic nor the LAMA planner. Thus, we have configured an additional portfolio adding LAMA-2008 (Richter and Westphal, 2008) instead of LAMA-2011 to the set of candidate planners considered for the design of FDSS-1, and then running GOP over all training instances to configure the new portfolio denoted as GOP-1L08, which is shown in Table 3.5. The time taken to compute this portfolio was 21,225.75 seconds, less than 6 hours (31 minutes for the first step and 5.4 hours for the second step).

Test, new domains GOP-1 and GOP-2 have been compared with FDSS-1 and FDSS-2 on the sequential satisficing track of the IPC 2011. All domains considered in the training data (IPC 2008) are included in the IPC 2011. Therefore, these domains were discarded resulting in a new test benchmark denoted as NEW IPC DOMAINS 2011 (see Table 2.1 on page 15). We have excluded GOP-1L08 in this evaluation because LAMA-2008 was not considered in the design of the FDSS portfolios.

Figure 3.6 shows the performance of both variants of FDSS and GOP over all the new domains. As reference, it also shows the performance of both GOP portfolios trained on the test domains (OSS GOP-1 and OSS GOP-2). The performance shown in Figure 3.6 has been computed taking into account only GOP, FDSS and OSS GOP portfolios. Our results indicate that GOP-1 performs better than both variants of FDSS.

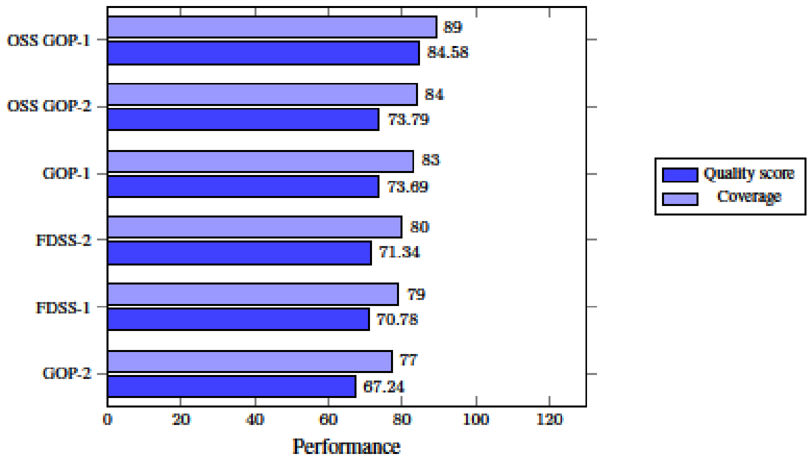


Figure 3.6: Assessment of GOP against FDSS on satisficing planning. The figure shows the performance of GOP, FDSS and OSS GOP portfolios for all planning tasks defined in the new domains introduced in the IPC 2011 sequential satisficing track.

To obtain an overall view of the performance of the resulting portfolios if they would have entered the IPC 2011, we have compared their performance with all the other entrants over all the new domains. It was found that LAMA-2011, the SBS, performs better than both variants of GOP. While GOP-1 solves 83 problems with a total quality score of 69.272, LAMA-2011 solves 92 problems with a total score equal to 82.051 —i. e., almost thirteen points above. Also, GOP-1 is far from the VBS (since it solves 108 planning tasks) as well as GOP-1 trained on the test set, which solves 89 planning tasks with a total quality score of 78.968.

Finally, GOP-1L08 was executed over the test set, since we have already compared GOP with FDSS. GOP-1L08 achieves a performance very close to the winner of the IPC 2011, LAMA-2011, since GOP-1L08 solves 90 problems with a total score of 79.301, while LAMA-2011 solves 92 problems with a total quality score of 82.291 —less than three points more.

Test, all domains The performance of the GOP portfolios has been assessed over the IPC 2011 discarding all domains included in the IPC-2008. However, all participant planners for the IPC-2011 had available all domains of the IPC-2008. Indeed, FDSS considered all domains selected in the IPC-2008 to configure its portfolios. Hence, we re-run the IPC 2011 for the sequential satisficing track with GOP-1L08, GOP-1 and GOP-2 as participant planners. The performance of the best planners (including the VBS and the OSS portfolio for the entire IPC 2011) are shown in Figure 3.7. The results clearly indicate that GOP-1L08 performs better than the SBS, LAMA-2011, in terms of quality score. GOP-1L08 solved less problems than LAMA-2011 because all the GOP portfolios were configured with the aim of maximizing quality score (while minimizing the overall running time). If the GOP portfolio had been configured to maximize coverage, it is expected that the resulting portfolio would have solved more than 246 instances. On the other hand, GOP-1 and GOP-2 perform better than the corresponding variants of FDSS.

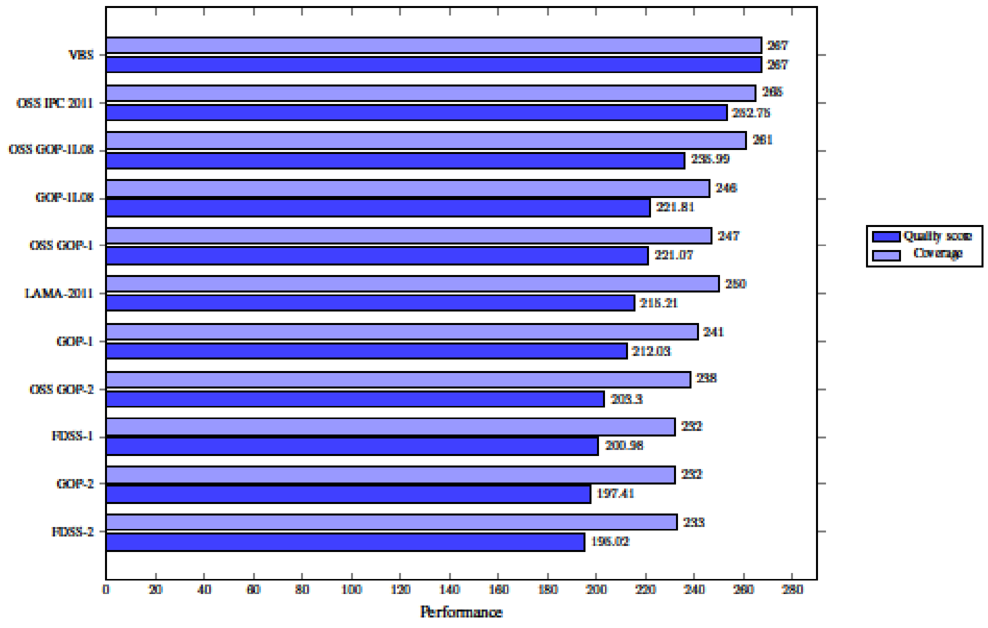


Figure 3.7: Performance of best participant planners in the IPC 2011 sequential satisficing track including GOP-1L08, GOP-1, GOP-2, LAMA-2011, VBS and OSS portfolios.

Comparison against Seipp et al. The contribution of Seipp *et al.* (Seipp et al., 2012) can be split into two parts: first, it was shown that using planners learnt for each domain can lead to good results; second, it was empirically found, among a wide number of learning methods, that distributing the overall allotted time uniformly among all planners produced the best test results despite that it did not achieve a remarkable training score. While we are not dealing with the first part of their contribution (where do planners come from) we tested their second contribution. So, we assume planners are given beforehand and compare our approach with portfolios generated by distributing the allotted time uniformly among a set of candidate planners.

Training We have generated two sequential portfolios. The first one (denoted as GOP-UNIFORM-1) results from applying the uniform method only to the component planners of GOP-1L08, the best portfolio automatically generated by GOP in the preceding subsection, which is shown in Table 3.5. The second one, denoted as GOP-UNIFORM-2, was configured applying the uniform method to all candidate planners.

Test Figure 3.8 shows the performance of both variants of GOP with the uniform method and the GOP-1L08 portfolio for the new domains and for the entire sequential satisficing track of the IPC 2011. The performance has been computed taking into account all participant planners in the IPC 2011. As reference, Figure 3.8 also shows the performance of the SBS and the VBS for this competition. These results show that GOP outperforms both variants of the uniform method for the given selection of candidate planners.

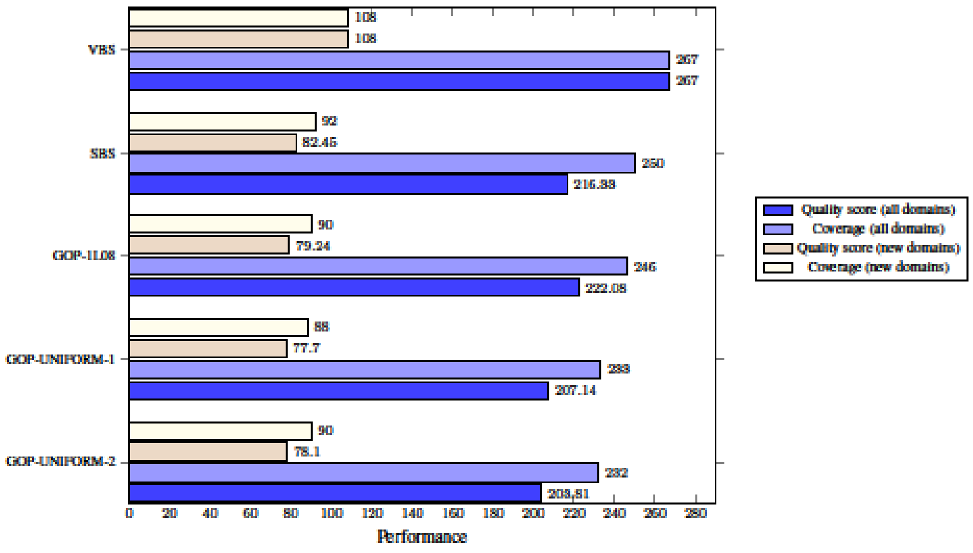


Figure 3.8: Assessment of GOP against the work by Seipp *et al.* on satisficing planning. The figure shows the performance of SBS, VBS, GOP-1L08 and both GOP-UNIFORM portfolios for the new domains and for all planning tasks from IPC 2011 sequential satisficing track.

IPC 2014 Similarly to the planning experiments, we have generated and submitted two sequential portfolios to the IPC 2014: MIPLAN and DPMPLAN. Both portfolios have been configured by repeating the experiments performed for optimal planning but using different training data sets. In the design of both portfolios, we have used all the planning tasks defined for the IPC 2011. Also, all the participant planners in the IPC 2011 were considered to derive them. However, MIPLAN removed the participant portfolios and added their component solvers instead whereas DPMPLAN did not. For more detailed information, please refer to (Núñez et al., 2014b).

Since the temporary files were not removed either in the satisficing track, our portfolios again achieved a score equal to zero in several planning domains. Thus, we have executed the competition for the five best planners according to the official results (among which MIPLAN is included) and DPMPLAN. We have only executed the five best planners because to the best of our knowledge, this problem only affected the results of our portfolios and the IBACoP portfolios, which won the competition. Also, we have considered the same data used to execute the original competition.

Table 3.6 describes the official competition results and the results obtained by re-running the competition. These results indicate that MIPLAN, the fourth best planner according to the competition results, should have been the runner up. MIPLAN obtained an official score equal to zero in the Transport domain while it solves all the planning tasks defined for that domain in our experiments. On the other hand, DPMPLAN, the $10th$ classified in the competition, obtains an overall score better than MERCURY, JASPER and FD-UNIFORM.

Analyzing the total score on each planning domain, we observed a large difference in performance between MIPLAN and IBACoP in the Floortile domain. Specifically, IBACoP solved 19 out of the 20 planning tasks with a total score equal to 18.91 whereas MIPLAN obtained a score equal to 4.23 as a result of solving five instances. We also realized that LPG-TD, a planner that was not considered in the design of MIPLAN, was the component planner used by IBACoP to solve all the instances which were not solved by MIPLAN. It is an example that reflects the importance of the inputs when automatically designing portfolios.

Participant planner	IPC Official Results	IPC Re-execution Results
IBACoP 2	166.21/198	197.07/227
IBACoP 1	162.73/196	205.27/239
MERCURY	153.04/172	160.39/176
MIPLAN	150.00/168	194.40/219
JASPER	144.89/173	163.08/185
FD-UNIFORM	143.25/172	163.87/191
DPMPLAN	125.50/147	168.63/199

Table 3.6: Results of the IPC 2014 sequential satisficing track. For each competition results, the table shows the total quality score and the number of problems solved by each participant planner.

3.3.3.3 Learning Track

In the learning track, planners focus on extracting domain dependent knowledge, which will be exploited in the test phase. GOP can be applied to the learning track since that knowledge is automatically extracted by planners in a prior offline training phase. Thus, we have generated and submitted MIPLAN to the learning track of the IPC 2014. It is able to automatically generate a portfolio configuration of domain-independent planners for a specific input domain (learning phase) and runs a specific static sequential portfolio for each input instance (test phase).

Figure 3.9 shows the results of the overall best quality award for the IPC 2014. As it can be seen, MIPLAN is the winner. It achieves the best overall quality and also the best coverage.

3.3.3.4 SAT

We generated and submitted one sequential SAT portfolio (termed MIPSAT) to the open track of the SAT Competition 2013. Our approach derives sequential portfolios, so the resulting portfolios are not optimized to fully exploit the multiple-core facilities of the competition as the winner did. Nevertheless, GOP does not prevent using solvers from the input set that run subsolvers in parallel, as it happened in fact in the portfolio configured for SAT 2013.

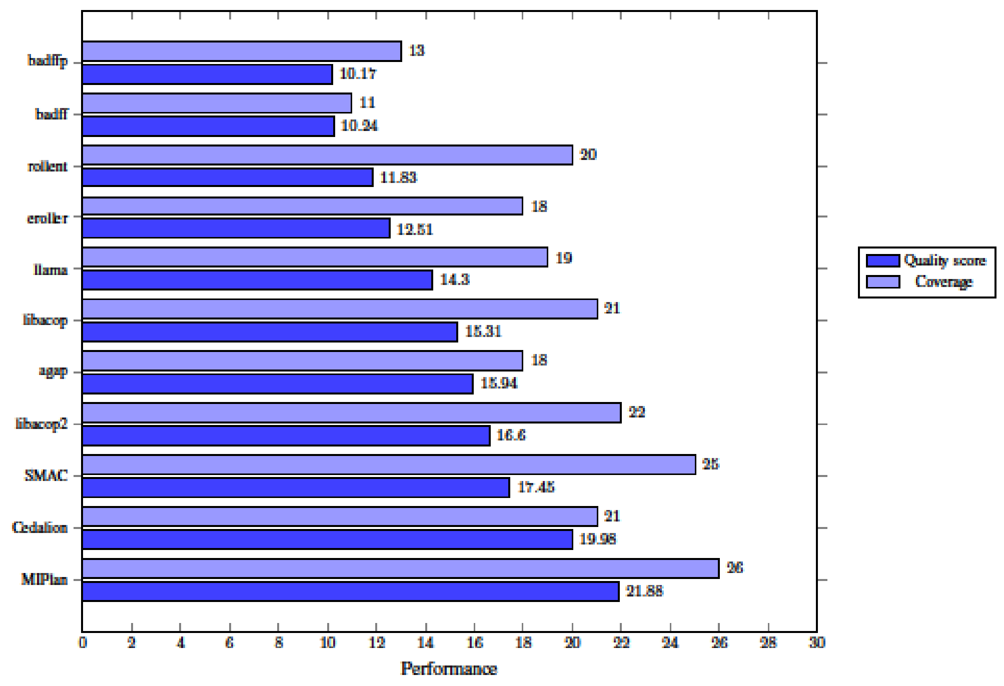


Figure 3.9: Results of the overall best quality award for the IPC 2014 learning track.

The input data for the MIP model was generated using the results of the SAT Competition 2011. We considered 1200 instances (see benchmark denoted as FULL SC 2011 in Table 2.1, page 15) and 54 participant solvers (including parallel solvers and portfolios in the set of candidate solvers) from that competition. Given that not all the candidate solvers were executed with all the training instances, those executions were considered as if the candidate solvers did not solve the training instances.

The time required to derive MIPSAT running Algorithm 1 was 3104.8 seconds. The execution sequence of portfolios generated by GOP is arbitrary. However, ties are broken in the SAT Competition 2013 in ascending order of the average wall-clock time. The resulting configuration is shown in Figure 3.10.

The results of the awarded solvers show that MIPSAT, the second best solver, solved 231 instances, only three instances less than the winner and 45 instances more than the third solver. As reference, the OSS portfolio and the VBS solved 234 and 253 instances respectively. Note that MIPSAT is a static sequential portfolio while the winner of this track (CSHC PAR8) is an 8-core parallel and dynamic portfolio. It always runs three different sequential dynamic portfolios on one core each and a fixed set of solvers on the remaining cores. Moreover, most participants in the open track were manually configured portfolios, which highlights the power of techniques that automatically generate portfolios.

Additionally, we have also generated and submitted sequential portfolios for several *core solvers* and sequential tracks of the SAT Competition 2013. These tracks only allow participants to use solvers that employ at most two different SAT solving engines for all runs and at any time in one track. Therefore, we have added an extra constraint to the MIP model defined by GOP so that the configuration of the generated portfolios is composed of at most two candidate solvers. For more

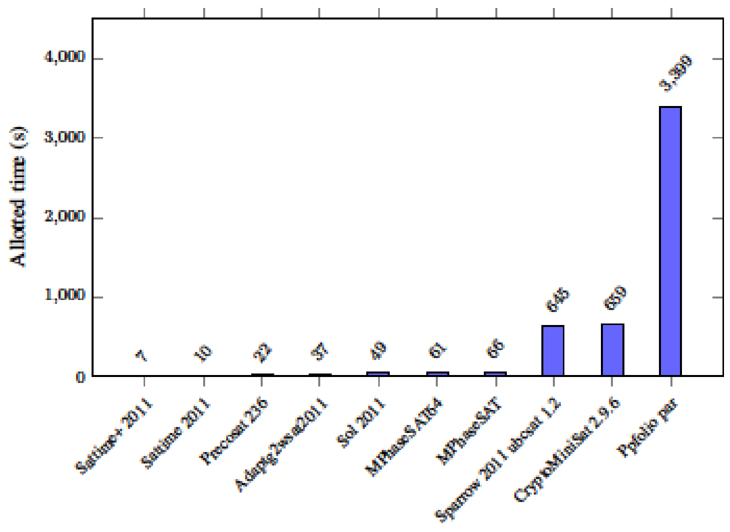


Figure 3.10: Sequential portfolio submitted to the open track of the SAT Competition 2013.

detailed information, please refer to (Núñez et al., 2013).

For each track in which we have participated, we submitted two versions of the same portfolio, termed MIPSAT 1 and MIPSAT 2. The first version is exactly the portfolio derived by GOP using data from the SAT Competition 2011, while MIPSAT 2 is the same portfolio but using the newest version of the selected solvers.

Table 3.7 describes the results obtained by MIPSAT on all the tracks in which it participated. As it can be seen, MIPSAT also won the silver medal in the Random SAT+UNSAT track. Moreover, these results show the advance in the SAT solvers, since *new* individual solvers performed better than portfolios configured using data from the previous competition (SAT Competition 2011).

Track	Ranking	Coverage
Application SAT	9/31	105/119
Application SAT+UNSAT	9/29	201/231
Hard-Combinatorial SAT	26/40	97/124
Hard-Combinatorial SAT+UNSAT	15/35	187/208
Random SAT	9/25	76/99
Random SAT+UNSAT	2/14	151/179

Table 3.7: Results of the MIPSAT portfolio on several *core solvers* and sequential tracks of the SAT Competition 2013. For each track, the table shows the ranking of the best MIPSAT version and the number of participants, and the number of instances solved by the best MIPSAT version and by the winner of the track.

3.3.4 GOP and Other Optimal Approaches

In this Section, we compare GOP against other optimal approaches for configuring sequential portfolios. As mentioned in the Related Work, the work by Streeter *et al.* does not constrain the total available time in the portfolio and it uses the preemptive mode to interleave the execution of the component solvers. That work focuses on a different problem to the one addressed in this Chapter and thus it has not been considered in this set of experiments. Moreover, according to its authors, CPHYDRA only works with a low number of solvers (5 in their experiments). Hence, GOP is only compared with ASPEED (Hoos et al., 2015) and the MIP formulation introduced by 3S (Kadioglu et al., 2011).

Despite the fact that ASPEED, the MIP task proposed by 3S and GOP solve the problem of deriving the OSS portfolio, there are some differences among these techniques which make a direct and fair comparison not possible. On the one hand, we have assessed the generation of the optimal portfolio by ASPEED (without the step for sorting the component solvers of the resulting portfolio) against our MIP task using $w_1 = 1$ and $w_2 = 0$. On the other hand, we have compared the MIP task defined by 3S with Algorithm 1, since both approaches compute the OSS portfolio that also minimizes the overall running time.

The experiment of MIPSAT (see Section 3.3.3.4) was repeated using the aforementioned approaches with the aim of assessing runtime. The first step of ASPEED was still running after 259,200 seconds (3 computation days) while GOP derived the portfolio configuration in 1036.15 seconds. On the other hand, the MIP model defined by 3S is not available online.[6] Thus, we decided to develop the fixed solver schedule approach defined by 3S (without column generation) using the modeling language ZIMPL, the same language used in GOP.

3S only considers the discrete time values t where each candidate solver just solves an instance in the training data (Malitsky et al., 2012a). Thus, the formulation of the problem proposed by 3S requires to generate a different MIP model for each given input data (candidate solvers and training instances). It allows us to minimize the number of binary variables in the resulting model. However, the solutions found by 3S are not optimal since it uses column generation, as opposed to the solutions generated by GOP, which are optimal.

We have generated a MIP model using the formulation of 3S and the training data set considered in the design of MIPSAT. SCIP, the MIP solver used to solve that MIP task took 205.71 seconds while the configuration of MIPSAT computed with GOP took 3104.8 seconds. Thus, the 3S approach is more efficient than GOP. However, the performance of the formulation proposed by 3S is strongly dependent on the input data. It defines one binary variable x_{st} per pair of candidate solver s and candidate time value t for the time allotted to the execution of the candidate solver s. 3S only considers time values t where solvers solve an instance. Therefore, there can be up to 5000 binary variables for each candidate solver, since the time limit in SAT is equal to 5000 seconds and 3S considers discrete-time.

The 3S model generated to compute the configuration of MIPSAT defines (on average) 138.04 binary variables x_{st} for each candidate solver. In total, there were 8378 binary variables in that model while the MIP model proposed by GOP defined 62,400 binary variables and 1252 continuous variables. Thus, the difference in performance between both models is mainly due to the number of binary variables.

The 3S MIP task defined for configuring MIPSAT, which considers 1200 training instances and 52 candidate solvers, was solved in 205.71 seconds. However, the authors of 3S claim that *"the main problem with the formulation is the sheer number of variables. For our most up-to-date benchmark with 37 solvers and more than 5000 training instances, solving the problem is impractical"* (Kadioglu et al., 2011). Hence, since the number of variables depends on the number of solvers and the 3S MIP task generated in this experiment was solved quickly, we hypothesize that the number of candidate time values t (where solvers solve an instance) in our training data set

[6]We tried to obtain their code through different ways without any success. The authors did not directly provide it either.

is too small and in general, this value is much higher. Therefore, we have performed an additional experiment with the aim of analyzing the correlation between the number of candidate time values t for each candidate solver and the time required to solve the MIP task.

This additional experiment consists of generating models using the formulation proposed by 3S and the input data defined in the design of MIPSAT but considering more candidate time values t (in the discrete range $[1, 5000]$) per candidate solver. So, we can analyze the impact of the diversity of the candidate time values per solver on the time required to solve the MIP task.

Figure 3.11 shows the time required to solve all the generated models and the original 3S model (in the axis labeled as $t(s)$). As it can be seen, the diversity of the candidate time values for each candidate solver has a strong impact on the efficiency of the model. The 3S task defined with at least only 400 variables x_{st} for each solver (in total, 22,103 binary variables) was solved in 4264.77 seconds while the task proposed by GOP (whose model defines 63,652 variables) took 3104.8 seconds. The number of variables defined by GOP does not depend on the diversity of the candidate time values, since it models the time allotted to each candidate solver as continuous variables.

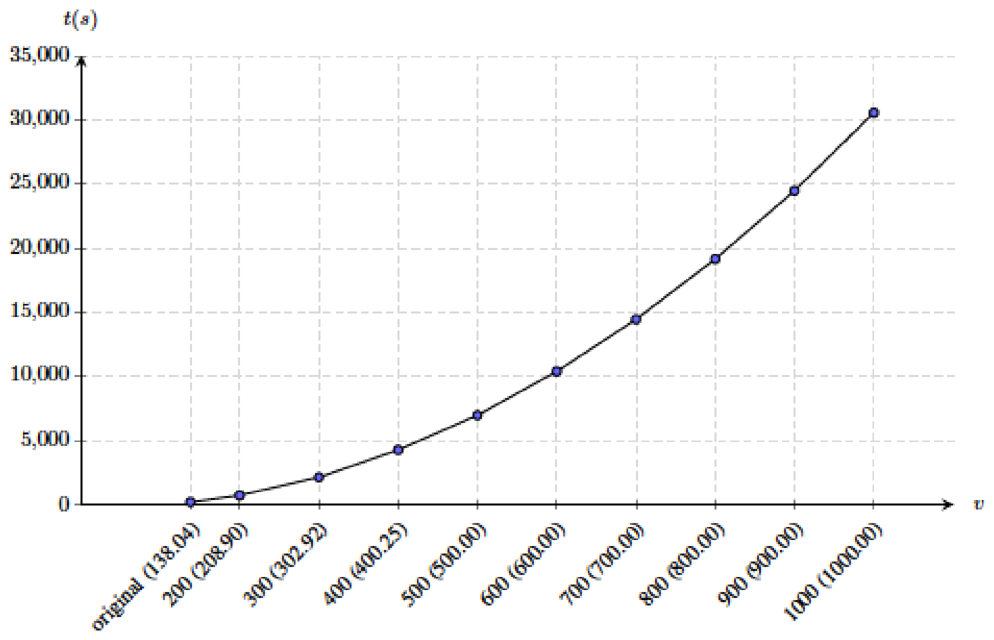


Figure 3.11: Results of the experiment designed to analyze the correlation between the number of candidate time values t for each candidate solver s and the time required to solve the 3S MIP task. The x-axis, marked as v, indicates the minimum number of variables x_{st} defined for each candidate solver and in brackets, the average of the total number of variables x_{st}.

3.3.5 Analysis of the Quality of the Solutions Achieved Over Time

This Section aims to analyze the quality of the solutions found by GOP over time. It always found the optimal solution for a given training data. However, the MIP solver finds several solutions until it ensures that the best solution found is the optimal one. Also, in some cases, the MIP solver can take several computation days to solve the MIP task. Therefore, we have analyzed the quality of the solutions found and the instant at which the optimal solution is achieved.

This analysis considers the experiment performed to compute the OSS portfolio for the sequential satisficing track of the IPC 2011 (see Section 3.3.1.2) because it is one of the hardest problems solved in this Chapter. Specifically, the analysis focuses on the first MIP task (first step) of Algorithm 1 for this experiment since the MIP solver only found the optimal solution for the second task. Indeed, the MIP solver usually only finds the optimal solution for the second task of Algorithm 1.

Figure 3.12 shows with points the solutions found by the MIP solver for the aforementioned task and the instant at which the solver ends. As it can be seen, the time required to solve the task was around 5.5 hours. However, the optimal solution was found after 2042 seconds (less than 35 minutes). Also, the quality of the third solution (which took 3.7 seconds) is equal to 245.3861, while the quality of the optimal solution is 252.75. Therefore, the MIP solver finds high quality solutions very quickly and it takes most of the time to ensure that the best solution found is the optimal one.

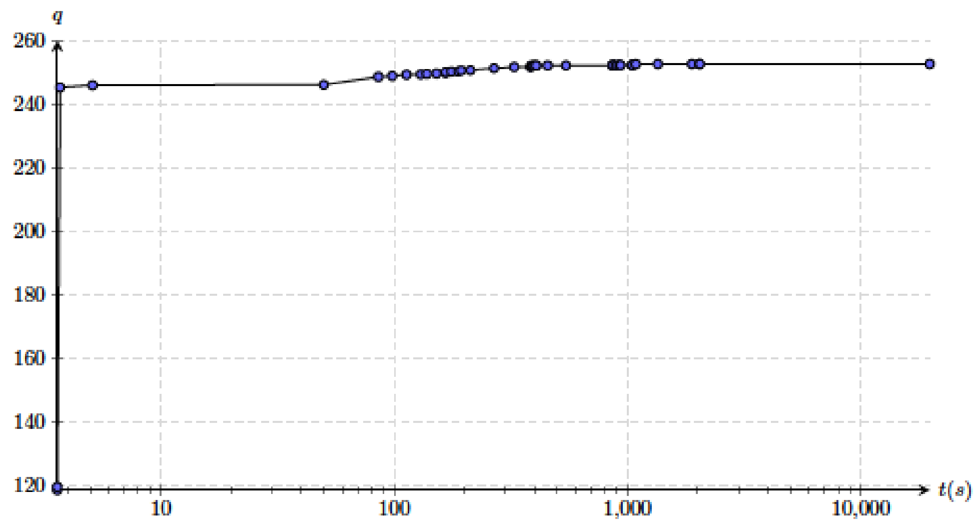


Figure 3.12: Analysis of the quality of the solutions achieved by GOP over time. The figure shows the quality of the solutions found by the MIP solver to compute the OSS portfolio for the sequential satisficing track of the IPC 2011. The x-axis shows the time required to find all the solutions while the y-axis indicates the quality score of the generated portfolios in the training benchmark.

3.4 Summary

In this Chapter we have introduced GOP, a new technique that automatically builds OSS portfolios for a particular training benchmark, a collection of candidate solvers and a performance criteria. GOP defines a MIP model to compute the optimal portfolio configuration. This MIP task for optimal planning and SAT can be seen as a variation of the Knapsack problem, where the reward for including an object in the bag is the number of problems solved by a solver within its allotted time. This allotted time represents the cost (weight) of including the object in the knapsack. The goal is to maximize the reward without exceeding the maximum cost (weight) that we can carry in the bag (the total time available for solving each planning task). On the other hand, the MIP task for satisficing planning can be seen as the Knapsack problem where utilities change over time, since satisficing planners can generate more than one solution.

GOP has been used to address several issues in the automated design of portfolios such as how to derive an upper bound on the solvers performance for a given set of problem solving tasks, among others. The main contributions presented in this Chapter can be summarized as follows:

1. GOP, a theoretically-grounded method that models the automated generation of portfolios using MIP tasks was introduced. We empirically showed that these MIP tasks can be solved in a reasonable time and the resulting portfolios generalize very well to unseen instances.

2. The best linear combination of participant solvers was computed for the International Planning Competition 2011 and SAT Competition 2013.

3. The utility of training instances was empirically analyzed when designing portfolios for Automated Planning and SAT.

4. The generalization capability of the portfolios automatically derived with GOP was assessed in Automated Planning and SAT.

5. GOP, the proposed method to compute OSS portfolios was compared and evaluated against previous optimal approaches from the portfolio literature.

6. The quality of the solutions found by GOP over time was analyzed.

3.5 Publications

In this Section we show all the published works related to this Chapter:

- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2012a). "How Good is the Performance of the Best Portfolio in IPC-2011?" In: *Proceedings of the ICAPS-12 Workshop on International Planning Competition*

- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2012b). "Performance Analysis of Planning Portfolios". In: *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS, Niagara Falls, Ontario, Canada, July 19-21, 2012.* AAAI Press, pp. 65–71

- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2013). "MIPSat". In: *In Proceedings of SAT Competition 2013, Solver and Benchmark Descriptions*, pp. 59–60

- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2014b). "MIPlan and DPMPlan". In: *Planner description, Deterministic track, International Planning Competition 2014*

- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2014a). "MIPlan". In: *Planner description, Learning track, International Planning Competition 2014*

- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2015a). "Automatic Construction of Optimal Static Sequential Portfolios for AI Planning and Beyond". In: *Artificial Intelligence Journal* 226, pp. 75–101

Chapter 4

Ordering Component Solvers in Sequential Portfolios

Several works in portfolios have shown their ability to outperform single-algorithm approaches in some tasks (e. g. SAT or Automated Planning). However, the order in which the component solvers of a sequential portfolio are executed is a relevant issue that has not been analyzed in depth yet. Most successful portfolio approaches for Automated Planning and SAT only focus on maximizing performance (measured as total quality score or coverage) for a fixed time limit. We hypothesize that the order of the component solvers affects the performance of the portfolio over time. Thus, a sequential portfolio should be sorted if its performance over time is relevant. As a consequence, in SAT and optimal planning, the average time required to solve problems can be significantly reduced, whereas in satisficing planning, lower-solution costs can be found more quickly, while preserving coverage or quality score.

An example of the interest in this particular problem can be found in the real world. For instance, during the Hurricane Sandy (2012) there were lots of blackouts in New York City. Thus, several electrical failures (generators, electrical lines, etc.) had to be repaired to restore the electricity as quickly as possible. This situation combines two problems, a logistic problem (parts that are needed to fix the electrical components) and a scheduling problem (Power Restoration Problem) (Hentenryck et al., 2011). The last one is very similar to the problem described in this Chapter. The electrical components that must be repaired are problems to be solved by solvers. The reward (power restored) for repairing each particular component is the number of problems solved (or the quality score of the solutions found) by a solver within its allotted time. The allotted time represents the time required (cost) to fix each component. Finally, the goal is to maximize the power flow (by repairing electrical components) in the network as quickly as possible (coverage or quality score over time).

In this Chapter, we propose to sort the component solvers in a sequential portfolio to improve its performance over time. We empirically show that the ordering used in the portfolio affects its performance over time and that performance can be improved by using a good ordering strategy.

This Chapter is organized as follows. First, Section 4.1 formally defines the problem of ordering component solvers in a sequential portfolio. Sections 4.2 and 4.3 describe the optimal and greedy approaches proposed. Section 4.4 reports the experimental results. Finally, Section 4.5 concludes with a summary and Section 4.6 shows the publications related to this Chapter.

4.1 Formal Description

In problem solving, the performance of a solver s is measured over a set of instances I (s can be either a solver or a portfolio). Every solver s is executed over every instance $i \in I$ to obtain the set R_{st} of solutions. This set contains every solution found by s within a given time bound t. We consider time as discrete with a discretization of one second. Each element of R_{st} stores the cost of the corresponding solution and a timestamp with the time required to find it. In case of solving problems optimally or SAT instances, solvers generate at most one solution per instance. However, in satisficing planning, solvers can generate multiple solutions. Therefore, the performance of s over time is measured by evaluating a specific metric over time. The metrics considered in this Thesis are coverage and total quality score (see Section 2.1.5.3, page 15). In the following, $P(s, I, t)$ denotes generically either coverage, $C(s, I, t)$ or total quality score, $Q(s, I, t)$. Recall that every solver is executed t seconds on each instance to compute a specific metric.

The definition of the sequential portfolio does not consider any notion of *ordering* among solvers in a portfolio (see Definition 2.1 on page 5). Indeed, the performance of the portfolio in time $T \geq \sum_{i=1}^{n} t_i$ is the same for any ordering, where t_i is the time allotted to the execution of each of the n component solvers. Hence, the notion of ordering in a sequential portfolio should be introduced.

Definition 4.1 (Ordering of a Sequential Portfolio). *An ordering τ of a sequential portfolio ρ is a full permutation over the solvers in ρ that defines the execution sequence, $\tau \doteq \{s_1, s_2, \ldots, s_n\}$.*

The component solvers of the sequential portfolios considered in this Thesis are not allowed to share any information among them. Thus, they can not take advantage of their position by using information from previous executions (e.g., a cost upper bound in satisficing planning). Since the notion of ordering has already been presented, the sorted sequential portfolio can be formally defined now as shown below.

Definition 4.2 (Sorted Sequential Portfolio). *Let ρ_τ denote the sorted sequential portfolio of solvers in ρ whose execution ordering is given by τ.*

In order to analyze the performance of a sorted sequential portfolio, we need to analyze separately the contribution of each solver to the overall performance.

Definition 4.3 (Partial Ordering of a Sequential Portfolio). *A partial ordering τ_k is a partial permutation over the first k solvers in the full permutation τ : $\tau_k \doteq \{s_1, s_2, ..., s_k\}$.*

Let ρ_{τ_k} denote the sorted sequential portfolio of solvers in ρ that considers only the first k solvers according to the ordering in τ_k.

Therefore, the first solver, s_1, completes its execution after t_1 seconds, the second solver will finish after $(t_1 + t_2)$ seconds and, in general, the j-th solver s_j will complete its execution after $\sum_{l=1}^{j} t_l$ seconds. Now, we can define how to measure the performance of the resulting portfolio over time as shown below.

Definition 4.4 (Performance of a Sorted Sequential Portfolio over Time). *The performance of a sorted sequence of solvers ρ_τ for a given problem set I over time t, $P(\rho_\tau, I, t)$ is defined as the sum of the best performance over all solvers in ρ executed in the order specified by τ for every instance in I in time less than or equal to t.*

In case that $t < \sum_{i=1}^{n} t_i$ not all component solvers will be considered to compute $P(\rho_\tau, I, t)$. In this case, only the solutions found by those solvers in ρ_τ that could be run before t are considered. The timestamp of each considered solution is equal to the time required to find it by the corresponding solver s_k plus $\sum_{i=1}^{k-1} t_i$. Next, we define the performance of a component solver, s_i, that occupies the i-th position in a permutation τ. It is computed as the increase in performance of the ordered portfolio by adding s_i to the portfolio.

Definition 4.5 (Performance of a Component Solver in a Sorted Sequential Portfolio). *Let $P_{s_i}(\rho_\tau, I)$ denote the performance of a component solver s_i in a partial permutation τ_i (that considers only the first i solvers in τ) wrt the benchmark I:*

$$P_{s_i}(\rho_\tau, I) = P(\rho_{\tau_i}, I, t_{s_i}) - P(\rho_{\tau_{i-1}}, I, t_{s_{i-1}})$$

where τ_i denotes the partial permutation of all solvers in τ until s_i; t_{s_i} is the sum of the allotted times of all solvers in ρ_{τ_i}; and, s_{i-1} denotes the previous solver of s_i in the partial permutation.

The performance of a solver s_i is defined as a function of the ordered portfolio ρ_τ since different solvers in ρ or different orderings τ would yield different performances —i. e., the performance of a solver depends on the previous solvers. As an example, consider a sequential portfolio ρ for optimal planning which consists of two solvers s_1 and s_2 which are allocated 4 and 7 seconds respectively. Let us assume that the performance of these solvers with respect to a set I of 20 planning tasks is:

- s_1 solves instances 11 to 20. Hence, $P(s_1, I, 4) = 10$.
- s_2 solves tasks 1 to 18, resulting in $P(s_2, I, 7) = 18$.

Assume further that both solvers solve the aforementioned instances in one second each. Therefore, $P(s_1, I, 1) = 10$ and $P(s_2, I, 1) = 18$. Figure 4.1 shows the performance over time of the two possible orderings for the given portfolio, $\tau_1 : (s_1, s_2)$ (red solid line) and $\tau_2 : (s_2, s_1)$ (blue dashed line). $P(s_1, I, 4)$ is equal to 10 since s_1 solved 10 instances within its time span. However, the performance of s_1 in each portfolio is different. $P_{s_1}(\rho_{\tau_1}, I)$ is equal to 10 because s_1 is the first solver to be executed. However, the performance $P_{s_1}(\rho_{\tau_2}, I)$ is equal to two since instances 11–18 have already been solved by the previous solver s_2. As shown in Figure 4.1, the performance of a sequential portfolio ρ at time $T = \sum_i t_i$, $P(\rho, I, T)$, is the same for every permutation τ. However, the performance of these orderings over time differs. Figure 4.1 also exemplifies a case where the portfolio achieving its maximum performance sooner is not the one with the best overall performance and, indeed, the first portfolio (red solid line) inscribes a smaller area than the second one.

The key observation is that the performance of ordered portfolios ρ_τ over time can be seen as *bivariate density functions*, $f_{\rho_\tau}(x, t)$. They are defined as the probability that the sorted portfolio ρ_τ reaches a performance P equal to x in t seconds: $f_{\rho_\tau}(x, t) = Prob(P = x, T = t)$. Accordingly, we define the probability function as follows:

Definition 4.6 (Probability Function of the Performance of a Sorted Sequential Portfolio). *Let $Prob(P \leq x, T = t)$ denote the probability that a portfolio ρ_τ reaches a performance equal to x or less in time t:*

$$Prob(P \leq x, T = t) = \sum_{x} f_{\rho_\tau}(x, t)$$

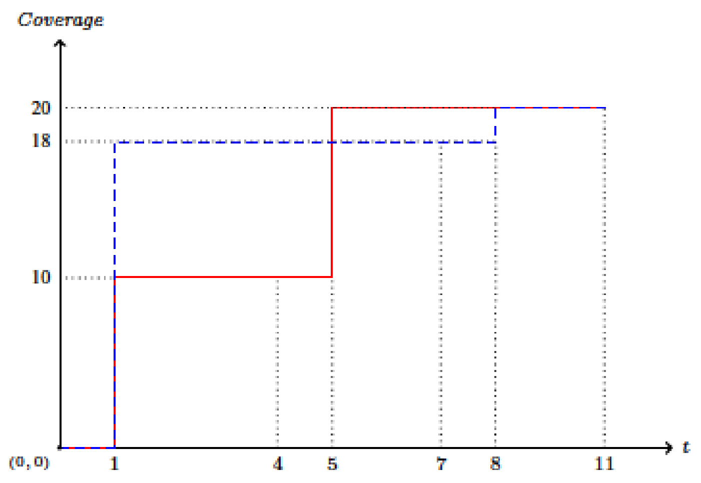


Figure 4.1: Performance of two different orderings of the same portfolio with respect to coverage.

This observation leads to propose the area inscribed by this probability function as the optimization criteria to compare different permutations τ of the same portfolio ρ. Thus, we define the optimization task as follows.

Definition 4.7 (Portfolio Ordering Task). *Given a collection of n component solvers of a sequential portfolio ρ, the portfolio ordering task consists of finding the permutation τ^* of solvers $s \in \rho$ for a given benchmark I such that it maximizes $F_{\rho_\tau}(x, t)$:*

$$F_{\rho_\tau}(x,t) = Prob(P \leq x, T \leq t) = \textstyle\sum_t Prob(P \leq x, T = t)$$

As a result, this task will sort the component solvers of the input portfolio ρ with the aim of maximizing the area inscribed by the probability function shown in Definition 4.6 of the resulting ordering τ (see Definition 4.1).

4.2 Optimal Approach

We first use heuristic search with an admissible heuristic function to find the optimal ordering with respect to a given benchmark. Specifically, we propose Depth First Branch and Bound (DFBnB). It requires two parameters: a sequential portfolio ρ and the set R_{si}, which will be used to compute the area inscribed by the probability function of every combination of the component solvers.

To find the optimal ordering τ^*, DFBnB starts with the empty permutation τ_0. Each node m contains the current partial permutation τ_m and the set A of solvers that are not yet in ρ_{τ_m} (initially, $A = \{s_i \mid s_i \in \rho\}$ i. e., all solvers in ρ). The successors of each node are generated by adding a solver $s \in A$ to the current permutation (and thus removing it from A). Each node defines a partial permutation of the component solvers in ρ, while each leaf node defines a full permutation of all solvers in ρ.

DFBnB uses $f(m) = g(m) + h(m)$. The g-value is the area inscribed by $F_{\rho_{\tau_m}}$ after $T_{\rho_{\tau_m}}$ seconds, where $T_{\rho_{\tau_m}}$ is equal to the sum of the time spans of every solver in ρ_{τ_m}:

$$g(m) = F_{\rho_{\tau_m}}(P(\rho_{\tau_m}, I, T_{\rho_{\tau_m}}), T_{\rho_{\tau_m}})$$

Suppose that DFBnB is initially given the portfolio $\rho = \{\langle s_1, 540\rangle, \langle s_2, 630\rangle, \langle s_3, 630\rangle\}$. Assume also that the DFBnB search is in a state m, where only the solver s_3 has been selected, so that $\rho_{\tau_m} : \{\langle s_3, 630\rangle\}$ and $T_{\rho_{\tau_m}} = 630$. Figure 4.2 shows the area inscribed by the probability function $F_{\rho_{\tau_m}}$ in the interval $[0, 630]$ (blue line patterned area), where points denoted with uppercase letters demarcate the areas inscribed by the probability function. Thus, the area inscribed by $F_{\rho_{\tau_m}}$ is computed as the sum of the rectangular areas $area(T_0A_0A_1T_1)$, $area(T_1A_2A_3T_2)$ and $area(T_2A_4A_5T_3)$.

In relation to $h(m)$, we have defined an admissible heuristic termed SQUARE that optimistically estimates the area inscribed by the probability function F_A. It just assumes that the order of the solvers contained in A is not relevant so that the portfolio will reach the performance $P(\rho, I, T)$ with a 100% probability, one second after the first solver in A starts its execution. $h(m)$ is computed as follows:

$$\begin{aligned} h(m) &= Prob(P \leq P(\rho, I, T), T = T_{\rho_{\tau_m}}) + F_A(P(\rho, I, T), T_A) \\ &= Prob(P \leq P(\rho, I, T), T = T_{\rho_{\tau_m}}) + T_A - 1 \end{aligned}$$

where T_A is equal to the sum of the time spans of every solver in A. The area inscribed by F_A is composed of two rectangular areas. The first one is defined in the interval $[T_{\rho_{\tau_m}}, T_{\rho_{\tau_m}} + 1]$, the first second of the execution of the first solver in A. This area is represented by the first term in $h(m)$. It is computed as the probability of reaching a performance equal to x or less in time $T_{\rho_{\tau_m}}$ multiplied by the time interval (one second). The second area is defined by the probability function F_A in the interval $[T_{\rho_{\tau_m}} + 1, T_{\rho_{\tau_m}} + T_A]$.

Following the example in Figure 4.2, $A = \{s_1, s_2\}$ and $T_A = 540 + 630 = 1170$. Since these solvers have not yet been included in ρ_{τ_m}, the area inscribed by F_A (yellow solid area) is computed using the SQUARE heuristic. The estimated yellow solid area is equal to the sum of the rectangular areas $area(T_3A_5B_0T_4)$ (first term of the heuristic function), $area(T_4BCT_5)$ (solver s_1) and $area(T_5CDT_6)$ (solver s_2). The rectangular areas $area(T_4BCT_5)$ and $area(T_5CDT_6)$ are equal to $T_A - 1 = 1169$. Hence:

$$\begin{aligned} f(m) = \;& area(T_0A_0A_1T_1) + area(T_1A_2A_3T_2) \\ &+ area(T_2A_4A_5T_3) + area(T_3A_5B_0T_4) + 1169 \end{aligned}$$

This technique yields optimal orderings for a specific set of instances. However, it can suffer from overfitting when evaluating its performance over a different benchmark.

4.3 Greedy Approach

The time required to find the optimal solution increases dramatically with the number of component solvers, since there are $n!$ different orderings. Thus, we propose an alternative approach based on greedy search to quickly find a suboptimal solution with good quality.

We assume that the performance $P_{s_i}(\rho_\tau, I)$ can be approximated with a straight line in the interval $[t_{s_{i-1}}, t_{s_{i-1}} + t_i]$, where $t_{s_{i-1}}$ is the sum of the allotted times of all solvers in $\rho_{\tau_{i-1}}$ and t_i is the execution time of s_i. The slope of this line is computed as the performance $P_{s_i}(\rho_\tau, I)$ divided by t_i. The slope has been selected as a heuristic (to be denoted as SLOPE) because it is a conservative approximation of the growth in performance of a component solver in the portfolio. Also, it considers

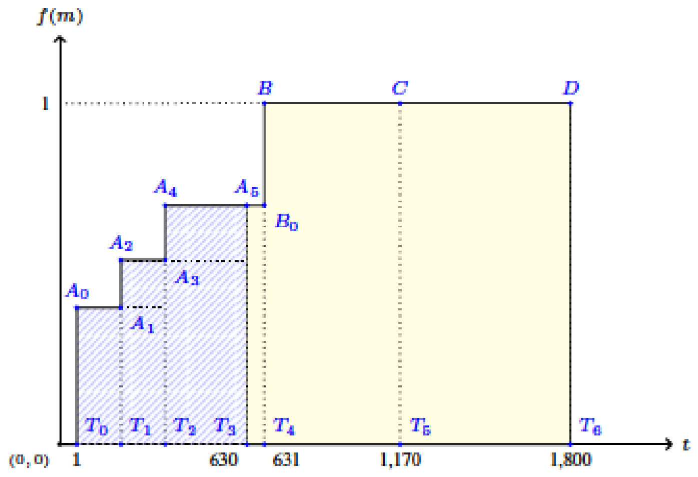


Figure 4.2: Example of the computation of $f(m)$ for computing the optimal ordering of a given portfolio. The blue area is the g-value while the area coloured in yellow represents the heuristic value.

the performance of each component solver with respect to the performance achieved by the previous solvers in the term $P_{s_i}(\rho_\tau, I)$ —see Definition 4.5.

We propose to use hill-climbing in the space of partial permutations of the input portfolio ρ. It takes the same parameters as DFBnB described previously. The search is initialized with the empty permutation τ_0, and the set $A = \{s_i \mid s_i \in \rho\}$. At each step, it selects the solver $s_i \in A$ which has the largest ratio $P_{s_i}(\rho_\tau, I)/t_i$. Then, the selected solver is added to the current permutation and removed from A. Finally, the algorithm returns the ordered portfolio ρ_τ.

Figure 4.3 shows the performance over time of a sequential portfolio sorted by the greedy approach. Specifically, the portfolio used in this example is the OSS portfolio for the sequential optimization track of the IPC 2011 (for details see Section 3.3.1.1, page 43). The green and red lines show the slope of each component solver. As it can be seen, the slope of each solver is lower than the slope of the previous solvers in the portfolio. Also, the slope of CPT-4 is almost a vertical line since its allotted time is equal to 1 second and its performance on the training benchmark (according to its allotted time) is equal to 31 (instances solved).

4.4 Empirical Evaluation

This Section compares the proposed algorithms (SLOPE and DFBnB) with other ordering strategies and reports their performance on Automated Planning and SAT. Inspired by the ordering criteria used by state-of-the-art portfolios in Automated Planning, the following algorithms were defined to compare with our solutions[1]. All the ordering algorithms resolved tie-breaking by using the order in which solvers were initially specified.

Shorter Time Spans (STS): inspired by PBP, this algorithm sorts the component solvers of a given portfolio in increasing order of the allotted time to run each solver.

[1]We also tried to use the ordering criteria defined by BUS against our approach, but the source code is not available.

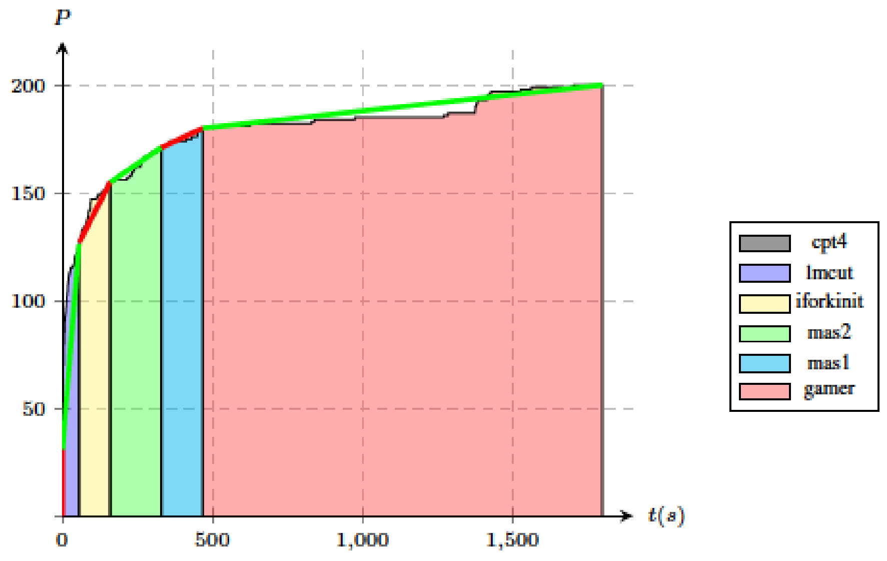


Figure 4.3: Example of a sequential portfolio sorted by the greedy approach.

Memory Failures (MF): inspired by FDSS, it uses the number of times that each component solver exceeds the available memory limit (and does not solve the task) to sort the given portfolio in decreasing order.

Decreasing Coverage (DC): inspired by FDSS, it uses the number of problems solved by each component solver to sort the input portfolio in decreasing order.

Random: sorts the solvers of the input portfolio randomly. This algorithm generates five random orderings and reports the average score of all generated orderings.

Confidence: uses the confidence provided by the learned models to sort the input portfolio in decreasing order. It is defined by IBACOP 2. Therefore, it only will be applied in the comparisons with dynamic input portfolios.

4.4.1 Static Input Portfolios

We used equation (4.1) to measure the score of the resulting orderings. This equation compares the area inscribed by the probability function of each sorted portfolio with the optimal sorting computed by DFBnB over the evaluation set. Thus, higher scores stand for better anytime behaviors.

$$ordering\_score(\rho_{\mathcal{T}}) = \frac{F_{\rho_{\mathcal{T}}}(P(\rho_{\mathcal{T}}, I, T), T)}{F_{\rho_{\mathcal{T}*}}(P(\rho_{\mathcal{T}^*}, I, T), T)} \tag{4.1}$$

The performance of all the considered sortings has been evaluated with different evaluation test sets: Either the training data set or an entirely new one (test set). The second type of experiments examines the generalization capabilities of all ordering approaches. Also, to avoid introducing any bias in the experimentation, different techniques for generating static sequential portfolios are considered:

GOP portfolios. This approach focused on deriving the optimal static sequential portfolio for a particular metric and a given training set (see Chapter 3 for more details).

Random portfolios. They consist of a random selection of solvers from a pool of candidate solvers (at most half the number of candidate solvers) so that at least one problem is solved in each training planning domain or SAT category. The time allotted to each component solver is also randomly chosen, such that the total time does not exceed T.

Random-uniform portfolios. They are random portfolios where the total available time is uniformly distributed among all the component solvers.

4.4.1.1 Automated Planning

We have performed two sets of experiments. The first one considers all the planning tasks from the IPC 2008 to configure and sort the input portfolio (training set). The new domains defined in the IPC 2011 are then used to assess the performance of the resulting portfolio (test set). In this experiment, we have used the set of candidate planners considered in the design of FDSS (FDSS-1 or FDSS-2, depending on the experiment) to configure the input portfolios. The second set of experiments takes the whole collection of planning tasks from the IPC 2011 as training set and the domains of the IPC 2014 (which were not included in IPC 2011) as test set. The input portfolios have been configured considering all the participants of the IPC 2011 but LPRPGP.[2] For more details about the composition of the benchmarks and candidate planners used in this empirical evaluation, see Table 2.1 and Annex A on pages 15 and 93 respectively.

The size of the candidate and component planners sets are defined in the ranges $[8, 38]$ and $[3, 14]$ respectively. The smaller planner sets were used in optimal planning, since there were few participants in the last IPCs. As a reference, IBACoP 2, the state-of-the-art portfolio, considers 12 candidate planners and five component planners.

The ordering generated by DFBnB for the training sets (IPC 2008 and IPC 2011) will be denoted as DFBnB 2008 and DFBnB 2011 respectively. Also, the ordering computed with DFBnB over the test sets will be used only to compute the test score of the orderings generated using the training set.

Table 4.1 shows the score of the resulting ordered portfolios using GOP portfolios. The training results show that the anytime behavior of the portfolio sorted with our greedy approach (SLOPE) is usually extremely close to the optimal performance. As it can be seen, the test results show that the orderings obtained by SLOPE and STS (using data from the IPC 2008) are the optimal orderings for the test set. Also, the permutation computed with the technique that generates the best ordering for the training data (DFBnB 2008) shows over-fitting as expected; it is worse than the random ordering, and it does not generalize well to unknown domains. Moreover, the orderings generated with MF and DC usually perform worse than the RANDOM ordering. Finally, our greedy approach outperforms the other approaches with a remarkable score (IPC 2011) and generalizes well on the IPC 2014.

Table 4.2 presents the training and test results for the random portfolios. Since we are using random portfolios, we have executed 50 times the training and test phases, each one with a different random portfolio. As it can be seen, the SLOPE random portfolio achieves again a training score extremely close to the score of the best permutation (DFBnB solution) of the input portfolio. The test results for random portfolios show that the SLOPE portfolio outperforms others under the same conditions. Also, all the generated orderings usually perform better than RANDOM in the test set. The differences in test scores between SLOPE and STS are larger in satisficing planning than in optimal

[2]We experienced problems with the CPlex license.

	Training Score				Test Score			
	IPC 2008		IPC 2011		IPC 2011		IPC 2014	
Ordering Algorithm	Optimal	Satisficing	Optimal	Satisficing	Optimal	Satisficing	Optimal	Satisficing
DFBnB 2008	**1.0000**	**1.0000**	-	-	0.9421	0.9381	-	-
DFBnB 2011	-	-	**1.0000**	**1.0000**	-	-	**0.9830**	0.9479
SLOPE PORTFOLIO	0.9944	0.9862	0.9993	0.9977	**1.0000**	**0.9735**	0.9764	0.9656
STS PORTFOLIO	0.9944	0.9642	0.9980	0.9756	**1.0000**	0.9576	0.9824	**0.9786**
RANDOM	0.9869	0.9604	0.8716	0.9545	0.9433	0.9591	0.8601	0.8450
DC PORTFOLIO	0.9985	0.9818	0.8579	0.9439	0.8435	0.9516	0.9031	0.7072
MF PORTFOLIO	0.9733	0.9334	0.6457	0.9472	0.9253	0.9437	0.6227	0.8267

Table 4.1: Training and test results of the sorted portfolios using GOP portfolios on Automated Planning. The best ordering scores are highlighted in bold.

planning, mostly because in satisficing planning the ordering task is harder and there is much more variability in the areas of the portfolios.

	Training Score and Std. Deviation (average)		Test Score and Std. Deviation (average)	
	IPC 2008		IPC 2011	
Ordering Algorithm	Optimal	Satisficing	Optimal	Satisficing
DFBnB 2008	**1.0000 - 0.0000**	**1.0000 - 0.0000**	0.9559 - 0.0448	0.9445 - 0.0530
SLOPE PORTFOLIO	0.9923 - 0.0095	0.9965 - 0.0038	**0.9800 - 0.0245**	**0.9617 - 0.0478**
STS PORTFOLIO	0.9850 - 0.0181	0.9900 - 0.0095	0.9797 - 0.0293	0.9595 - 0.0381
RANDOM	0.9661 - 0.0209	0.9571 - 0.0160	0.9255 - 0.0467	0.8792 - 0.0599
DC PORTFOLIO	0.9897 - 0.0113	0.9623 - 0.0284	0.9437 - 0.0606	0.8713 - 0.1082
MF PORTFOLIO	0.9426 - 0.0375	0.9332 - 0.0434	0.9438 - 0.0486	0.8671 - 0.1095
	IPC 2011		IPC 2014	
DFBnB 2011	**1.0000 - 0.0000**	**1.0000 - 0.0000**	0.9806 - 0.0170	**0.9613 - 0.0450**
SLOPE PORTFOLIO	0.9932 - 0.0109	0.9984 - 0.0030	**0.9829 - 0.0115**	0.9607 - 0.0409
STS PORTFOLIO	0.9794 - 0.0203	0.9752 - 0.0264	0.9774 - 0.0193	0.9394 - 0.0541
RANDOM	0.9143 - 0.0654	0.8394 - 0.0900	0.9485 - 0.0490	0.7895 - 0.1107
DC PORTFOLIO	0.9866 - 0.0154	0.9720 - 0.0246	0.9707 - 0.0180	0.8996 - 0.0858
MF PORTFOLIO	0.9393 - 0.0443	0.7977 - 0.1587	0.9773 - 0.0253	0.6877 - 0.2051

Table 4.2: Training and test results of the sorted portfolios using random portfolios on Automated Planning. The best ordering scores are highlighted in bold.

The training and test results for the random-uniform portfolio are described in Table 4.3. Strikingly, these results show that the SLOPE portfolio achieves a training score extremely close to the score obtained by the optimal sorting despite the fact that the uniform method penalizes the SLOPE heuristic. This method also penalizes the STS algorithm. However, the STS portfolio performs worse than the RANDOM portfolio. As it can be seen, the difference in test score between SLOPE and STS (the two algorithms that are penalized by using the uniform time assignment) is quite large. The scores of SLOPE, DFBnB 2008 and 2011 are very close. However, the time required by our greedy approach is exponentially shorter than the time required by DFBnB. Overall, all ordering algorithms (but DFBnB) sort a given portfolio in less than one second, while DFBnB can take several days to sort a given portfolio (depending on the number of component planners).

	Training Score and Std. Deviation (average)		Test Score and Std. Deviation (average)	
	IPC 2008		IPC 2011	
Ordering Algorithm	Optimal	Satisficing	Optimal	Satisficing
DFBnB 2008	**1.0000 - 0.0000**	**1.0000 - 0.0000**	0.9370 - 0.0302	0.9086 - 0.0550
SLOPE PORTFOLIO	0.9996 - 0.0010	0.9996 - 0.0008	0.9353 - 0.0306	**0.9123 - 0.0567**
STS PORTFOLIO	0.9560 - 0.0228	0.9494 - 0.0180	0.8588 - 0.0497	0.8254 - 0.0704
RANDOM	0.9648 - 0.0216	0.9664 - 0.0098	0.9311 - 0.0271	0.8525 - 0.0433
DC PORTFOLIO	0.9913 - 0.0104	0.9623 - 0.0142	0.9413 - 0.0329	0.8183 - 0.0782
MF PORTFOLIO	0.9442 - 0.0304	0.9442 - 0.0208	**0.9566 - 0.0292**	0.8371 - 0.0800
	IPC 2011		IPC 2014	
DFBnB 2011	**1.0000 - 0.0000**	**1.0000 - 0.0000**	0.9776 - 0.0139	0.8878 - 0.0486
SLOPE PORTFOLIO	0.9992 - 0.0013	0.9995 - 0.0015	**0.9791 - 0.0143**	**0.8919 - 0.0453**
STS PORTFOLIO	0.8941 - 0.0535	0.8429 - 0.0644	0.9261 - 0.0331	0.7897 - 0.0823
RANDOM	0.9411 - 0.0255	0.8745 - 0.0431	0.9487 - 0.0149	0.7982 - 0.0529
DC PORTFOLIO	0.9636 - 0.0131	0.9871 - 0.0112	0.9530 - 0.0144	0.8694 - 0.0437
MF PORTFOLIO	0.9412 - 0.0314	0.9086 - 0.0558	0.9605 - 0.0339	0.7711 - 0.0981

Table 4.3: Training and test results of the sorted portfolios using random-uniform portfolios on Automated Planning. The best ordering scores are highlighted in bold.

4.4.1.2 SAT

The ordering algorithms have also been assessed on SAT. In this experiment, all the instances defined for the SAT Competition 2011 are used to configure and sort the input portfolios (training set). The resulting sorted portfolios are then evaluated on the whole collection of instances from the SAT Competition 2013 (test set). On the other hand, MIPSAT has been used as the input portfolio for ordering GOP portfolios and its component solvers have been considered to generate the random and random-uniform input portfolios (see Figure 3.10 on page 58 for details about the MIPSAT configuration).

Similarly to the planning experiments, the ordering generated by DFBnB for the training set has been denoted as DFBnB 2011. Also, the ordering computed with DFBnB over the test set will be used only to compute the test score of the orderings generated using the training set.

Table 4.4 shows the results of the orderings generated using MIPSAT as the input portfolio. As it can be seen, the orderings computed by SLOPE and DC achieve the best training score (without considering DFBnB 2011). Strikingly, the test results show a large difference in score between both orderings, despite the fact that they achieved the same training score. In general, the test results show low scores except for the DC portfolio. Indeed, the orderings computed by MF and STS perform worse than the RANDOM ordering in the test set. On the other hand, the ordering computed by DFBnB for the training set seems to not suffer from over-fitting in the test set, since it achieves the second best score and performs better than the random ordering.

The training and test results for the random portfolios are described in Table 4.5. As in the planning experiments, we have executed 50 times the training and test phases, each one with a different random portfolio. The training results indicate that the ordering generated with SLOPE achieves a quite remarkable training score. As it can be seen, the test results show that the input portfolio ordered with DFBnB for the training set achieves the best test score, which together with the previous results tends to empirically confirm that DFBnB does not suffer from over-fitting in the SAT experiments. Also, the difference in test score between SLOPE and DFBnB 2011 is not large,

Ordering algorithm	Training score	Test score
DFBnB 2011	**1.0000**	0.8970
SLOPE PORTFOLIO	0.9747	0.8302
STS PORTFOLIO	0.9699	0.8200
RANDOM	0.9635	0.8265
DC PORTFOLIO	0.9747	**0.9952**
MF PORTFOLIO	0.9492	0.7994

Table 4.4: Training and test results of the sorted portfolios using GOP portfolios on SAT. The best ordering scores are highlighted in bold.

unlike the difference in computation time, which is exponentially large. In the SAT experiments, the greedy approach always sorts the input portfolio in less than one second whereas the DFBnB based approach can take more than one hour, depending on the size of the input portfolio. On the other hand, in this experiment, the DC portfolio does not perform well in the test set. Indeed, it performs worse than the STS portfolio, which performed worse than the random ordering for the GOP input portfolio (see Table 4.4).

Ordering algorithm	Training Score and Std. Deviation (average)	Test Score and Std. Deviation (average)
DFBnB 2011	**1.0000 - 0.0000**	**0.9778 - 0.0363**
SLOPE PORTFOLIO	0.9924 - 0.0105	0.9654 - 0.0432
STS PORTFOLIO	0.9742 - 0.0283	0.9301 - 0.0665
RANDOM	0.8803 - 0.0651	0.8544 - 0.0978
DC PORTFOLIO	0.9401 - 0.0702	0.9157 - 0.1295
MF PORTFOLIO	0.8582 - 0.1131	0.7802 - 0.1620

Table 4.5: Training and test results of the sorted portfolios using random portfolios on SAT. The best ordering scores are highlighted in bold.

Table 4.6 presents the results for the random-uniform portfolios. Similarly to the planning results, the training results show that the SLOPE portfolio achieves a score extremely close to the score of the optimal ordering despite the fact that the uniform method penalizes the SLOPE heuristic. The test results show that DFBnB 2011 performs again very well in the test set. Also, STS, the technique which is also penalized by the uniform method, does not perform well in the test set as expected; the difference in test score among STS and SLOPE (or DFBnB 2011) is quite large. Finally, SLOPE achieves again a near-optimal solution in the training set and generalizes very well on the SAT Competition 2013.

4.4.2 Dynamic Input Portfolios

In this experiment, we only focus on Automated Planning because most of the dynamic SAT portfolios only run one solver for most of the available time. Thus, we now apply all the ordering algorithms defined above to IBACOP 2, the winner of the IPC-2014 (satisficing planning). Instead of configuring the same portfolio for all test instances (as the approaches used in the previous experiments), IBACOP 2 generates a different sequential portfolio for each input instance. Therefore,

Ordering algorithm	Training Score and Std. Deviation (average)	Test Score and Std. Deviation (average)
DFBnB 2011	**1.0000 - 0.0000**	0.9821 - 0.0208
SLOPE PORTFOLIO	0.9999 - 0.0001	**0.9830 - 0.0200**
STS PORTFOLIO	0.9318 - 0.0513	0.8904 - 0.0839
RANDOM	0.8938 - 0.0354	0.8467 - 0.0489
DC PORTFOLIO	0.9823 - 0.0181	0.9778 - 0.0458
MF PORTFOLIO	0.8912 - 0.0554	0.8064 - 0.0804

Table 4.6: Training and test results of the sorted portfolios using random-uniform portfolios on SAT. The best ordering scores are highlighted in bold.

the score of each sorted ordering is computed as $\sum_{i \in I} \text{score}(\rho_{i\tau})$, where $\rho_{i\tau}$ is the ordered portfolio computed for each input instance i. The generation of each portfolio is based on learned data using a training set.

All the instances defined in the IPC 2011 have been considered to sort the input portfolios and the domains of the IPC 2014 (which were not included in IPC 2011) to evaluate the resulting orderings (see Table 2.1 on page 15 for details).

As it can be seen in Table 4.7, the SLOPE portfolio achieves again near-optimal solutions in the training set. The training score obtained by the DC portfolio is remarkable while the training score of the STS portfolio is worse than the score achieved by the random ordering (similarly to the random-uniform portfolios). It is mainly due to the uniform method, which is applied by IBACoP 2 to distribute the available time among the component solvers. The CONFIDENCE portfolio does not show training score because it was ordered by its learned models. On the other hand, the test score of the resulting orderings show that SLOPE again outperforms others. However, the test score of all the permutations are close among them.

Ordering algorithm	Training score	Test score
DFBnB 2011	**160.00**	105.93
SLOPE	159.94	**108.95**
STS	144.64	107.32
RANDOM	149.30	107.89
DC	158.31	105.92
MF	152.87	107.10
CONFIDENCE	-	108.37

Table 4.7: Training and test score of the sorted portfolios using IBACoP 2 on Automated Planning. The best ordering scores are highlighted in bold.

4.5 Summary

In this Chapter, we have presented a formal definition for the problem of sorting the component solvers in a sequential portfolio. It focuses on maximizing the area inscribed by the probability function of the sorted portfolio. This open problem has been addressed with the aim of improving the performance of the portfolios over time. The contributions of this Chapter can be summarized as follows:

1. The problem of ordering component solvers in a sequential (static or dynamic) portfolio was formally defined.

2. Two different algorithms to solve the problem were proposed and compared. We empirically showed that our greedy approach produces near-optimal solutions very quickly and that it generalizes much better than an optimal solution with respect to a specific training set which has been observed to suffer from overfitting in the planning experiments.

3. An extensive evaluation was performed with the algorithms introduced here, a random ordering algorithm and others from the literature with data from the last three IPCs (2008, 2011 and 2014) and the SAT Competitions 2011 and 2013.

4.6 Publications

Next we show the published work related to this Chapter:

- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2015b). "Sorting Sequential Portfolios in Automated Planning". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 1638–1644

Chapter 5

Conclusions and Future Work

In this Chapter, we summarize the main contributions of this Thesis, discuss the conclusions and describe possible future work.

5.1 Contributions

The main contributions of this Thesis can be split into two parts, as we describe in detail next.

5.1.1 The Automated Generation of Static Sequential Portfolios

The notion of portfolio applied to problem solving has shown to be a promising avenue of research. We analyzed the state-of-the-art portfolio techniques and introduced GOP, a Mixed-Integer Programming approach in order to theoretically address the automated generation of sequential portfolios. Besides, we showed that GOP is able to compute an upper bound on the performance that is feasible with a linear combination of candidate solvers for a particular training data set. In our view, reaching an overall performance larger than the upper bound automatically defined by GOP in the same pool of instances, under the same conditions, would be a remarkable achievement.

We also performed an empirical analysis to determine the composition of the best training benchmark to configure high-performance portfolios. Our results suggest that not all problems provide the same information, termed *utility* here, and that portfolios configured with small training data sets can perform very well. Hence, we conjecture that the best training benchmark should contain only a small number of instances that a few solvers are able to solve.

In addition, we assessed the performance of GOP against the most successful approaches to automatically configure portfolios in planning. Our results indicated that GOP, a theoretically-grounded method frequently dominates all others under the same conditions. Indeed, we submitted a participant planner termed MIPLAN to the learning track of the IPC 2014. The competition results indicated that MIPLAN, the planning system which uses GOP to generate a sequential portfolio for each given planning domain, outperforms others. It achieved the best overall quality and also the best coverage in the competition. Therefore, MIPLAN won the overall best quality award.

Finally, GOP was evaluated on the SAT Competition 2013. We submitted one sequential portfolio (called MIPSAT) to the open track, which was specifically defined for portfolio approaches. The competition results of this track reported that MIPSAT was the second best solver. It only solved three instances less than the winner. Also, we showed that the best performance achievable with a linear combination of the participant solvers was the same than the performance achieved by the winner. Thus, MIPSAT only solved three instances less than the best portfolio configuration for the open track.

5.1.2 The Automated Process of Ordering the Component Solvers in a Sequential Portfolio

Most state-of-the-art approaches in the automated design of portfolios did not focus on the order in which the component solvers should be executed. Thus, we presented a formal definition of the problem of sorting the component solvers in a sequential portfolio as a function defined over time. In addition, we introduced two algorithms to solve the aforementioned problem. The first one solved the problem optimally for a given data set using DFBnB and an admissible heuristic. The second one was a greedy approach that used the ratio between performance of each solver and execution time.

The results of the extensive empirical evaluation performed on Automated Planning and SAT indicated that the performance of the portfolio over time can significantly vary by using different ordering algorithms. Besides, the DFBnB based approach computed an optimal ordering for the training set but it did not generalize well to unseen instances in Automated Planning. It suffered from over-fitting when evaluated its performance over a test set. Also, the time required to compute the optimal ordering can grow exponentially with the number of solvers of the input portfolio. On the other hand, SLOPE, the greedy approach, computed orderings very fast, and obtained near-optimal solutions when compared against the optimal technique in training. Moreover, it generalized much better than the state-of-the-art ordering techniques. It is important to remark that the good behavior of SLOPE did not depend on the algorithm used for generating the input portfolio, as shown by using randomly generated portfolios (with uniform and non-uniform times).

In view of these results, we conjecture that it is going to be difficult to find a better algorithm in terms of the balance between computation time, generalization power and quality of results.

5.1.3 Publications

List of publications related to this Thesis:

- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2012a). “How Good is the Performance of the Best Portfolio in IPC-2011?” In: *Proceedings of the ICAPS-12 Workshop on International Planning Competition*
- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2012b). “Performance Analysis of Planning Portfolios”. In: *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS, Niagara Falls, Ontario, Canada, July 19-21, 2012.* AAAI Press, pp. 65–71
- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2013). “MIPSat”. In: *In Proceedings of SAT Competition 2013, Solver and Benchmark Descriptions*, pp. 59–60
- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2014b). “MIPlan and DPMPlan”. In: *Planner description, Deterministic track, International Planning Competition 2014*
- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2014a). “MIPlan”. In: *Planner description, Learning track, International Planning Competition 2014*

- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2015a). "Automatic Construction of Optimal Static Sequential Portfolios for AI Planning and Beyond". In: *Artificial Intelligence Journal* 226, pp. 75–101
- Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2015b). "Sorting Sequential Portfolios in Automated Planning". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 1638–1644

5.2 Future Work

In this Thesis, we addressed relevant issues about the automated design of sequential portfolios. As future work, we propose to generalize the MIP model to the parallel case. Also, we would like to study different behaviors in the sequential portfolio once the first solution has been found. We believe that developing techniques based on dynamic portfolios, which could generate several configurations while solving the given input instance, might improve the performance of the static portfolios considered here. Moreover, we propose to analyze the influence of the ordering algorithms on these dynamic portfolios. Additionally, we suggest to extend the MIP formulation proposed by GOP with the aim of solving the portfolio generation task and the ordering task together. Finally, it would be very interesting to continue working on the utility analysis of the training instances. It is a promising avenue of research related to the automated design of portfolios and other tasks.

Bibliography

Carlos Ansótegui, Yuri Malitsky, and Meinolf Sellmann (2014). "MaxSAT by Improved Instance-Specific Algorithm Configuration". In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.* Ed. by Carla E. Brodley and Peter Stone. AAAI Press, pp. 2594–2600 (page 30).

Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney (2009). "A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms". In: *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings.* Ed. by Ian P. Gent. Vol. 5732. Lecture Notes in Computer Science. Springer, pp. 142–157 (page 30).

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer (2002). "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning* 47.2-3, pp. 235–256 (page 22).

Christer Bäckström and Bernhard Nebel (1995). "Complexity Results for SAS+ Planning". In: *Computational Intelligence* 11, pp. 625–656 (page 22).

Adrian Balint, Daniel Diepold, Daniel Gall, Simon Gerber, Gregor Kapler, and Robert Retz (2011). "EDACC - An Advanced Platform for the Experiment Design, Administration and Analysis of Empirical Algorithms". In: *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, pp. 586–599 (page 14).

Debasish Basak, Srimanta Pal, Dipak Ch, and Ra Patranabis (2007). "Support vector regression". In: *Neural Information Processing Letters and Reviews*, pp. 203–224 (page 35).

Anton Belov, Daniel Diepold, Marijn Heule, and Matti Järvisalo, eds. (2014). *Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions.* Vol. B-2014-2. Department of Computer Science Series of Publications B. ISBN 978-951-51-0043-6. University of Helsinki (page 6).

Armin Biere (2011). "Lingeling and Friends the SAT Competition 2011". In: *Technical Report* (page 32).

Armin Biere, Marijn Heule, Hans van Maaren, and Tory Walsh (2009). *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications.* Amsterdam, The Netherlands, The Netherlands: IOS Press (page 12).

Bernd Bischl et al. (2015). “ASlib: A Benchmark Library for Algorithm Selection”. In: *CoRR* abs/1506.02465 (page 33).

Avrim Blum and Merrick L. Furst (1997). “Fast Planning Through Planning Graph Analysis”. In: *Artif. Intell.* 90.1-2, pp. 281–300 (page 17).

Blai Bonet and Hector Geffner (2001). “Planning as Heuristic Search”. In: *Artificial Intelligence* 129.1-2, pp. 5–33 (page 51).

Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer (2005). “Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators”. In: *J. Artif. Intell. Res. (JAIR)* 24, pp. 581–621 (page 19).

Leo Breiman (2001). “Random Forests”. In: *Machine Learning* 45.1, pp. 5–32 (page 25).

G. Briscoe and T. Caelli (1996). *A Compendium of Machine Learning: Symbolic Machine Learning*. Ablex Series in Artificial Intelligence v. 1. Ablex Pub. (page 23).

Christina N. Burt, Nir Lipovetzky, Adrian R. Pearce, and Peter J. Stuckey (2015). “Scheduling with Fixed Maintenance, Shared Resources and Nonlinear Feedrate Constraints: A Mine Planning Case Study”. In: *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, pp. 91–107 (page 10).

Shaowei Cai and Kaile Su (2012). “Configuration Checking with Aspiration in Local Search for SAT”. In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* Ed. by Jörg Hoffmann and Bart Selman. AAAI Press (page 32).

Isabel Cenamor, Tomás de la Rosa, and Fernando Fernández (2012). “Mining IPC-2011 Results”. In: *In ICAPS 2012 Workshop on International Planning Competition* (page 22).

Isabel Cenamor, Tomás de la Rosa, and Fernando Fernández (2013). “Learning Predictive Models to Configure Planning Portfolios”. In: *In ICAPS 2013 Workshop on Planning and Learning* (pages 9, 23).

Isabel Cenamor, Tomás de la Rosa, and Fernando Fernández (2014). “IBaCoP and IBaCoP2 Planner”. In: *Planner description, IPC 2014* (pages 2, 8, 25).

Yair Censor (1977). “Pareto optimality in multiobjective problems”. In: *Applied Mathematics and Optimization* 4.1, pp. 41–59 (page 25).

Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth (2010). “Step-by-step data mining guide”. In: (page 22).

Yixin Chen, Benjamin W. Wah, and Chih-Wei Hsu (2006). “Temporal Planning using Subgoal Partitioning and Resolution in SGPlan”. In: *J. Artif. Intell. Res. (JAIR)* 26, pp. 323–369 (pages 19, 24).

Lukás Chrpa (2010). “Generation of macro-operators via investigation of action dependencies in plans”. In: *Knowledge Eng. Review* 25.3, pp. 281–297 (page 23).

Lukás Chrpa and Thomas Leo McCluskey (2012). “On Exploiting Structures of Classical Planning Problems: Generalizing Entanglements”. In: *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, pp. 240–245 (page 23).

Lukás Chrpa and Mauro Vallati (2014). “AGAP: As Good as Possible”. In: *Planner description, IPC 2014* (page 24).

Lukás Chrpa, Mauro Vallati, and Thomas Leo McCluskey (2014). “MUM: A Technique for Maximising the Utility of Macro-operators by Constrained Generation and Use”. In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014* (page 24).

Lukás Chrpa, Mauro Vallati, Thomas Leo McCluskey, and Diane E. Kitchin (2013). “Generating Macro-Operators by Exploiting Inner Entanglements”. In: *Proceedings of the Tenth Symposium on Abstraction, Reformulation, and Approximation, SARA 2013, 11-12 July 2013, Leavenworth, Washington, USA.* (Page 24).

Andrew Coles and Amanda Smith (2007). “Marvin: A Heuristic Search Planner with Online Macro-Action Learning”. In: *J. Artif. Intell. Res. (JAIR)* 28, pp. 119–156 (page 19).

Stephen A. Cook (1971). “The Complexity of Theorem-Proving Procedures”. In: *STOC*. Ed. by Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman. ACM, pp. 151–158 (page 12).

Alexander Czutro, Sudhakar M. Reddy, Ilia Polian, and Bernd Becker (2014). “SAT-Based Test Pattern Generation with Improved Dynamic Compaction”. In: *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems, Mumbai, India, January 5-9, 2014*, pp. 56–61 (page 12).

Carmel Domshlak, Malte Helmert, Erez Karpas, Emil Keyder, Silvia Richter, Gabriele Röger, Jendrik Seipp, and Matthias Westphal (2011). “BJOLP: The big joint optimal landmarks planner”. In: *In Seventh International Planning Competition (IPC 2011), Deterministic Part*, pp. 91–95 (page 49).

Richard O. Duda, Peter E. Hart, and David G. Stork (2000). *Pattern Classification (2Nd Edition)*. Wiley-Interscience (page 30).

Niklas Eén and Niklas Sörensson (2003). “An Extensible SAT-solver”. In: *SAT*. Ed. by Enrico Giunchiglia and Armando Tacchella. Vol. 2919. Lecture Notes in Computer Science. Springer, pp. 502–518 (page 13).

Chris Fawcett, Malte Helmert, Holger Hoos, Erez Karpas, Gabriele Röger, and Jendrik Seipp (2011). “FD-Autotune: Domain-Specific Configuration using Fast Downward”. In: pp. 13–20 (pages 21, 26).

Chris Fawcett, Mauro Vallati, Frank Hutter, Jörg Hoffmann, Holger Hoos, and Kevin Leyton-Brown (2014). "Improved Features for Runtime Prediction of Domain-Independent Planners". In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014* (page 26).

Alan Fern, Sung Wook Yoon, and Robert Givan (2004). "Learning Domain-Specific Control Knowledge from Random Walks". In: *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, pp. 191–199 (page 8).

Maria Fox and Derek Long (1998). "The Automatic Inference of State Invariants in TIM". In: *J. Artif. Intell. Res. (JAIR)* 9, pp. 367–421 (page 17).

Santiago Franco, Mike Barley, and Pat Riddle (2014). "RIDA: In Situ Selection of Heuristic Subsets". In: *Planner description, Deterministic track, International Planning Competition 2014* (page 51).

Raquel Fuentetaja and Daniel Borrajo (2006). "Improving Control-Knowledge Acquisition for Planning by Active Learning". In: *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, pp. 138–149 (page 8).

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Marius Thomas Schneider, and Stefan Ziller (2011). "A Portfolio Solver for Answer Set Programming: Preliminary Report". In: *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*. Ed. by James P. Delgrande and Wolfgang Faber. Vol. 6645. Lecture Notes in Computer Science. Springer, pp. 352–357 (pages 9, 35).

Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub (2007). "Conflict-Driven Answer Set Solving". In: *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. Ed. by Manuela M. Veloso, p. 386 (page 35).

Alfonso Gerevini, Alessandro Saetti, and Ivan Serina (2003). "Planning Through Stochastic Local Search and Temporal Action Graphs in LPG". In: *J. Artif. Intell. Res. (JAIR)* 20, pp. 239–290 (pages 19, 24).

Alfonso Gerevini, Alessandro Saetti, and Ivan Serina (2006). "An Approach to Temporal Planning and Scheduling in Domains with Predictable Exogenous Events". In: *J. Artif. Intell. Res. (JAIR)* 25, pp. 187–231 (page 19).

Alfonso Gerevini, Alessandro Saetti, and Mauro Vallati (2014). "Planning through Automatic Portfolio Configuration: The PbP Approach". In: *J. Artif. Intell. Res. (JAIR)* 50, pp. 639–696 (pages 7, 19).

Malik Ghallab, Dana Nau, and Paolo Traverso (2004). *Automated planning - theory and practice*. Elsevier (page 10).

Carla P. Gomes and Bart Selman (2001). “Algorithm Portfolios”. In: *Artificial Intelligence* 126.1-2, pp. 43–62 (page 34).

Aarti Gupta, Malay K. Ganai, and Chao Wang (2006). “SAT-Based Verification Methods and Applications in Hardware Verification”. In: *Formal Methods for Hardware Verification, 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2006, Bertinoro, Italy, May 22-27, 2006, Advanced Lectures*, pp. 108–143 (page 12).

Greg Hamerly and Charles Elkan (2003). “Learning the K in K-Means”. In: *In Neural Information Processing Systems*. MIT Press, p. 2003 (page 30).

Frank Van Harmelen, Vladimir Lifschitz, and Bruce Porter (2008). *Handbook of Knowledge Representation (1st edition)*. Elsevier (page 12).

Pearl E. Hart, Nils J. Nilsson, and Bertram Raphael (1968). “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactional System Science and Cybernetics* 4.2, pp. 100–107 (page 43).

Malte Helmert (2004). “A Planning Heuristic Based on Causal Graph Analysis”. In: *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*. AAAI Press, pp. 161–170 (page 51).

Malte Helmert (2006). “The Fast Downward Planning System”. In: *Journal of Artificial Intelligence Research (JAIR)* 26, pp. 191–246 (pages 20, 51).

Malte Helmert (2009). “Concise finite-domain representations for PDDL planning tasks”. In: *Artif. Intell.* 173.5-6, pp. 503–535 (page 22).

Malte Helmert and Carmel Domshlak (2011). “LM-Cut: Optimal Planning with the Landmark-Cut Heuristic”. In: *In Seventh International Planning Competition (IPC 2011), Deterministic Part* (page 49).

Malte Helmert and Hector Geffner (2008). “Unifying the Causal Graph and Additive Heuristics”. In: *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pp. 140–147 (page 51).

Malte Helmert, Gabriele Röger, and Erez Karpas (2011). “Fast Downward Stone Soup: A Baseline for Building Planner Portfolios”. In: *In ICAPS 2011 Workshop on Planning and Learning*, pp. 28–35 (pages 2, 20).

Pascal Van Hentenryck, Carleton Coffrin, and Russell Bent (2011). “Vehicle Routing for the Last Mile of Power System Restoration”. In: *Proceedings of the 17th Power Systems Computation Conference (PSCCÕ11), Stockholm, Sweden* (page 63).

Jörg Hoffmann (2003). “The Metric-FF Planning System: Translating ”Ignoring Delete Lists” to Numeric State Variables”. In: *Journal of Artificial Intelligence Research (JAIR)* 20, pp. 291–341 (pages 19, 24).

Jörg Hoffmann and Bernhard Nebel (2001). "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *Journal of Artificial Intelligence Research (JAIR)* 14, pp. 253–302 (pages 22, 51).

Holger Hoos, Roland Kaminski, Marius Thomas Lindauer, and Torsten Schaub (2015). "aspeed: Solver scheduling via answer set programming". In: *TPLP* 15.1, pp. 117–142 (pages 34, 59).

Holger Hoos, Roland Kaminski, Torsten Schaub, and Marius Thomas Schneider (2012). "Aspeed: ASP-based Solver Scheduling". In: *ICLP (Technical Communications)*. Ed. by Agostino Dovier and V'ıtor Santos Costa. Vol. 17. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 176–187 (page 8).

Holger Hoos, Marius Thomas Lindauer, and Torsten Schaub (2014). "claspfolio 2: Advances in Algorithm Selection for Answer Set Programming". In: *TPLP* 14.4-5, pp. 569–585 (page 33).

Andrei Horbach, Thomas Bartsch, and Dirk Briskorn (2012). "Using a SAT-solver to schedule sports leagues". In: *J. Scheduling* 15.1, pp. 117–125 (page 12).

Adele Howe and Eric Dahlman (2002). "A Critical Assessment of Benchmark Comparison in Planning". In: *Journal of Artificial Intelligence Research (JAIR)* 17, pp. 1–33 (page 1).

Adele Howe, Eric Dahlman, Christoper Hansen, Michael Scheetz, and Anneliese Von Mayrhauser (2000). "Exploiting competitive planner performance". In: *Recent Advances in AI Planning*, pp. 62–72 (page 16).

Richard Howey, Derek Long, and Maria Fox (2004). "VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL". In: *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), 15-17 November 2004, Boca Raton, FL, USA*, pp. 294–301 (page 14).

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown (2011). "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, pp. 507–523 (pages 26, 33).

Frank Hutter, Holger Hoos, Kevin Leyton-Brown, and Thomas Stützle (2009). "ParamILS: An Automatic Algorithm Configuration Framework". In: *J. Artif. Intell. Res. (JAIR)* 36, pp. 267–306 (pages 21, 29).

Frank Hutter, Lin Xu, Holger Hoos, and Kevin Leyton-Brown (2014). "Algorithm runtime prediction: Methods & evaluation". In: *Artif. Intell.* 206, pp. 79–111 (page 31).

Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann (2011). "Algorithm Selection and Scheduling". In: *CP*. Ed. by Jimmy Ho-Man Lee. Vol. 6876. Lecture Notes in Computer Science. Springer, pp. 454–469 (pages 9, 31, 59).

Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney (2010). "ISAC - Instance-Specific Algorithm Configuration". In: *ECAI 2010 - 19th European Conference on Artificial*

Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings. Ed. by Helder Coelho, Rudi Studer, and Michael Wooldridge. Vol. 215. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 751–756 (pages 28, 30).

Henry Kautz and Bart Selman (1998). “Blackbox: A new approach to the application of theorem proving to problem solving”. In: *Working notes of the AIPS98 Workshop on Planning and Combinatorial Search, Pittsburgh, PA* (page 17).

Henry Kautz, Bart Selman, and Joerg Hoffmann (2006). “SatPlan: Planning as Satisfiability”. In: *Abstracts of the 5th International Planning Competition* (pages 17, 24).

Emil Keyder and Hector Geffner (2008). “Heuristics for Planning with Action Costs Revisited”. In: *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pp. 588–592 (page 51).

Ashiqur R. KhudaBukhsh, Lin Xu, Holger Hoos, and Kevin Leyton-Brown (2009). “SATenstein: Automatically Building Local Search SAT Solvers from Components”. In: *IJCAI*. Ed. by Craig Boutilier, pp. 517–524 (page 29).

Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos (1997). “Extending Planning Graphs to an ADL Subset”. In: *Recent Advances in AI Planning, 4th European Conference on Planning, ECP'97, Toulouse, France, September 24-26, 1997, Proceedings*, pp. 273–285 (page 17).

Ron Kohavi (1995). “The Power of Decision Tables”. In: *Machine Learning: ECML-95, 8th European Conference on Machine Learning, Heraclion, Crete, Greece, April 25-27, 1995, Proceedings*, pp. 174–189 (page 25).

Lars Kotthoff (2014). “Algorithm Selection for Combinatorial Search Problems: A Survey”. In: *AI Magazine* 35.3, pp. 48–60 (page 28).

Akshat Kumar, Sudhanshu Shekhar Singh, Pranav Gupta, and Gyana R. Parija (2014). “Near-Optimal Nonmyopic Contact Center Planning Using Dual Decomposition”. In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014* (page 10).

Carlos Linares López, Sergio Jiménez Celorrio, and Angel García Olaya (2015). “The deterministic part of the seventh International Planning Competition”. In: *Artificial Intelligence* 223, pp. 82–119 (page 11).

Marius Thomas Lindauer, Holger Hoos, Frank Hutter, and Torsten Schaub (2015). “AutoFolio: An Automatically Configured Algorithm Selector”. In: *J. Artif. Intell. Res. (JAIR)* 53, pp. 745–778 (page 33).

Nir Lipovetzky and Hector Geffner (2011). “Searching for Plans with Carefully Designed Probes”. In: *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011* (page 24).

Derek Long and Maria Fox (2003). “The 3rd International Planning Competition: Results and Analysis”. In: *J. Artif. Intell. Res. (JAIR)* 20, pp. 1–59 (page 19).

David J. C. MacKay (2003). *Information theory, inference, and learning algorithms*. Cambridge University Press (page 21).

Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann (2012a). “Parallel SAT Solver Selection and Scheduling”. In: *CP*. Ed. by Michela Milano. Vol. 7514. Lecture Notes in Computer Science. Springer, pp. 512–526 (pages 31, 59).

Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann (2012b). “Satisfiability Solver Selector”. In: *In Proceedings of SAT Challenge 2012, Solver and Benchmark Descriptions*, pp. 50–51 (page 31).

Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann (2013a). “Algorithm Portfolios Based on Cost-Sensitive Hierarchical Clustering”. In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. Ed. by Francesca Rossi. IJCAI/AAAI (page 32).

Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann (2013b). “Boosting Sequential Solver Portfolios: Knowledge Sharing and Accuracy Prediction”. In: *Learning and Intelligent Optimization - 7th International Conference, LION 7, Catania, Italy, January 7-11, 2013, Revised Selected Papers*. Ed. by Giuseppe Nicosia and Panos M. Pardalos. Vol. 7997. Lecture Notes in Computer Science. Springer, pp. 153–167 (page 32).

Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann (2013c). “Parallel Lingeling, CCASat, and CSCH-based Portfolios”. In: *In Proceedings of SAT Competition 2013, Solver and Benchmark Descriptions*, pp. 26–27 (pages 1, 32, 45).

Yuri Malitsky and Meinolf Sellmann (2009). “Stochastic Offline Programming”. In: *ICTAI 2009, 21st IEEE International Conference on Tools with Artificial Intelligence, Newark, New Jersey, USA, 2-4 November 2009*. IEEE Computer Society, pp. 784–791 (page 30).

Yuri Malitsky, David Wang, and Erez Karpas (2014). “The AllPACA Planner: All Planners Automatic Choice Algorithm”. In: *Planner description, IPC 2014* (pages 7, 24).

Harry Markowitz (1952). “Portfolio Selection”. In: *The Journal of Finance* 7.1, pp. 77–91 (page 1).

Hootan Nakhost, Martin Müller, Richard Valenzano, and Fan Xie (2011). “Arvand: The art of Random Walks”. In: *IPC2011 Deterministic Track Planner Reports* (page 22).

Muhammad Abdul Hakim Newton, John Levine, Maria Fox, and Derek Long (2007). “Learning Macro-Actions for Arbitrary Planners and Domains”. In: *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, pp. 256–263 (page 19).

Mladen Nikolic, Filip Maric, and Predrag Janicic (2009). “Instance-Based Selection of Policies for SAT Solvers”. In: *Theory and Applications of Satisfiability Testing - SAT 2009, 12th Interna-*

tional Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. Ed. by Oliver Kullmann. Vol. 5584. Lecture Notes in Computer Science. Springer, pp. 326–340 (page 31).

Mladen Nikolic, Filip Maric, and Predrag Janicic (2013). "Simple algorithm portfolio for SAT". In: *Artif. Intell. Rev.* 40.4, pp. 457–465 (pages 28, 30).

Raz Nissim, Jörg Hoffmann, and Malte Helmert (2011). "Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning". In: *IJCAI, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 1983–1990 (pages 43, 49).

Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2012a). "How Good is the Performance of the Best Portfolio in IPC-2011?" In: *Proceedings of the ICAPS-12 Workshop on International Planning Competition* (pages 62, 78).

Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2012b). "Performance Analysis of Planning Portfolios". In: *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS, Niagara Falls, Ontario, Canada, July 19-21, 2012.* AAAI Press, pp. 65–71 (pages 62, 78).

Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2013). "MIPSat". In: *In Proceedings of SAT Competition 2013, Solver and Benchmark Descriptions*, pp. 59–60 (pages 7, 28, 58, 62, 78).

Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2014a). "MIPlan". In: *Planner description, Learning track, International Planning Competition 2014* (pages 42, 62, 78).

Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2014b). "MIPlan and DPMPlan". In: *Planner description, Deterministic track, International Planning Competition 2014* (pages 50, 55, 62, 78).

Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2015a). "Automatic Construction of Optimal Static Sequential Portfolios for AI Planning and Beyond". In: *Artificial Intelligence Journal* 226, pp. 75–101 (pages 8, 62, 79).

Sergio Núñez, Daniel Borrajo, and Carlos Linares López (2015b). "Sorting Sequential Portfolios in Automated Planning". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 1638–1644 (pages 8, 75, 79).

Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan (2008). "Using Case-based Reasoning in an Algorithm Portfolio for Constraint Solving". In: *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science*, pp. 210–216 (pages 8, 34).

J. Scott Penberthy and Daniel S. Weld (1992). "UCPOP: A Sound, Complete, Partial Order Planner for ADL". In: *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, October 25-29, 1992.* Pp. 103–114 (page 17).

I. Pohl (1970). “Heuristic search viewed as path finding in a graph”. In: *Artificial Intelligence* 1.3-4, pp. 193–204 (page 20).

Julie Porteous, Laura Sebastia, and Jörg Hoffmann (2001). “On the Extraction, Ordering, and Usage of Landmarks in Planning”. In: pp. 37–48 (page 22).

J. Ross Quinlan (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann (page 23).

John R. Rice (1976). “The Algorithm Selection Problem”. In: *Advances in Computers* 15, pp. 65–118 (page 27).

Silvia Richter, Malte Helmert, and Matthias Westphal (2008). “Landmarks Revisited”. In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pp. 975–982 (page 22).

Silvia Richter, Jordan Tyler Thayer, and Wheeler Ruml (2010). “The Joy of Forgetting: Faster Anytime Search via Restarting”. In: *ICAPS*. Ed. by Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz. AAAI, pp. 137–144 (page 20).

Silvia Richter and Matthias Westphal (2008). “The LAMA planner. Using Landmark Counting in Heuristic Search”. In: *Planner description, Deterministic track, International Planning Competition 2008* (pages 22, 53).

Silvia Richter and Matthias Westphal (2010). “The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks”. In: *Journal of Artificial Intelligence Research (JAIR)* 39, pp. 127–177 (pages 19, 24, 44).

Jussi Rintanen (2012). “Engineering Efficient Planners with SAT”. In: *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, pp. 684–689 (page 24).

Jussi Rintanen (2014). “Madagascar: Scalable Planning with SAT”. In: *Planner description, Deterministic track, International Planning Competition 2014* (page 17).

Mattia Rizzini, Chris Fawcett, Mauro Vallati, Alfonso Emilio Gerevini, and Holger Hoos (2015). “Portfolio Methods for Optimal Planning: An Empirical Analysis”. In: *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*, pp. 494–501 (pages 8, 9, 26).

Mark Roberts and Adele Howe (2006). “Directing a Portfolio with Learning”. In: *Proceedings of the AAAI 2006 Workshop on Learning for Search*, pp. 126–135 (page 17).

Mark Roberts and Adele Howe (2007). “Learned models of performance for many planners”. In: *Proceedings of the ICAPS-07 Workshop of AI Planning and Learning* (pages 18, 19).

Mark Roberts and Adele Howe (2009). “Learning from Planner Performance”. In: *Artificial Intelligence* 173.5-6, pp. 536–561 (page 18).

Mario Rodríguez-Molins, Miguel Á. Salido, and Federico Barber (2010). “Domain-Dependent Planning Heuristics for Locating Containers in Maritime Terminals”. English. In: Lecture Notes in Computer Science 6096, pp. 742–751 (page 6).

Stuart J. Russell and Peter Norvig (2010). *Artificial Intelligence - A Modern Approach (3. internat. ed.)* Pearson Education (page 9).

Horst Samulowitz, Chandra Reddy, Ashish Sabharwal, and Meinolf Sellmann (2013). “Snappy: A Simple Algorithm Portfolio”. In: *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*. Ed. by Matti Järvisalo and Allen Van Gelder. Vol. 7962. Lecture Notes in Computer Science. Springer, pp. 422–428 (page 2).

Jendrik Seipp, Manuel Braun, Johannes Garimort, and Malte Helmert (2012). “Learning Portfolios of Automatically Tuned Planners”. In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*, pp. 368–372 (pages 7, 20, 26, 54).

Jendrik Seipp and Manuel Braun Johannes Garimort (2014). “Fast Downward Uniform Portfolio”. In: *Planner description, IPC 2014* (page 21).

Jendrik Seipp, Silvan Sievers, Malte Helmert, and Frank Hutter (2015). “Automatic Configuration of Sequential Planning Portfolios”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Pp. 3364–3370 (pages 7, 25).

Herbert A. Simon and Joseph B. Kadane (1975). “Optimal Problem-Solving Search: All-Oor-None Solutions”. In: *Artif. Intell.* 6.3, pp. 235–247 (page 18).

Matthew J. Streeter, Daniel Golovin, and Stephen F. Smith (2007). “Combining Multiple Heuristics Online”. In: *AAAI*. AAAI Press, pp. 1197–1203 (page 28).

Éric D. Taillard (1993). “Benchmarks for basic scheduling problems”. In: *European Journal of Operational Research* 64.2, pp. 278–285 (page 19).

Paolo Traverso and Marco Pistore (2004). “Automated Composition of Semantic Web Services into Executable Processes”. In: *The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings*, pp. 380–394 (page 10).

Richard Anthony Valenzano, Hootan Nakhost, Martin Müller, Jonathan Schaeffer, and Nathan R. Sturtevant (2012). “ArvandHerd: Parallel Planning with a Portfolio”. In: *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, pp. 786–791 (pages 7, 22).

Mauro Vallati, Lukás Chrpa, Marek Grzes, Thomas Leo McCluskey, Mark Roberts, and Scott Sanner (2015). “The 2014 International Planning Competition: Progress and Trends”. In: *AI Magazine* 36.3, pp. 90–98 (page 12).

Mauro Vallati, Lukás Chrpa, and Diane E. Kitchin (2014). "ASAP: An Automatic Algorithm Selection Approach for Planning". In: *International Journal on Artificial Intelligence Tools* 23.6 (pages 9, 23).

Mauro Vallati, Chris Fawcett, Alfonso Gerevini, Holger Hoos, and Alessandro Saetti (2013). "Automatic Generation of Efficient Domain-Optimized Planners from Generic Parametrized Planners". In: *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013, Leavenworth, Washington, USA, July 11-13, 2013.* (Page 19).

Manuela M. Veloso, Jaime G. Carbonell, M. Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe (1995). "Integrating planning and learning: the PRODIGY architecture". In: *J. Exp. Theor. Artif. Intell.* 7.1, pp. 81–120 (page 17).

Vincent Vidal (2004). "A Lookahead Strategy for Heuristic Search Planning". In: *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, pp. 150–160 (page 19).

Daniel S. Weld, Corin R. Anderson, and David E. Smith (1998). "Extending Graphplan to Handle Uncertainty & Sensing Actions". In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.* Pp. 897–904 (page 17).

Lin Xu, Holger Hoos, and Kevin Leyton-Brown (2010). "Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection". In: *Proceedings of the Twenty-Fourth AAI Conference on Artificial Inteligence*. AAAI, pp. 210–216 (pages 7, 29).

Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown (2008). "SATzilla: Portfolio-based Algorithm Selection for SAT". In: *Journal of Artificial Intelligence Research (JAIR)* 32, pp. 565–606 (pages 2, 29).

Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown (2009). "Satzilla2009: An Automatic Algorithm Portfolio for SAT". In: *Solver description, SAT competition 2009* (page 29).

Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown (2012a). "Evaluating Component Solver Contributions to Portfolio-Based Algorithm Selectors". In: *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, pp. 228–241 (pages 16, 29).

Lin Xu, Frank Hutter, Jonathan Shen, Holger Hoos, and Kevin Leyton-Brown (2012b). "SATzilla2012: Improved Algorithm Selection Based on Cost-sensitive Classification Models". In: *Solver description, SAT Challenge 2012* (page 29).

Appendix A

Candidate Solvers

In this Annex we show the candidate solvers used in all experiments. Table A.1 shows the planners considered to compute the OSS portfolio for the sequential optimization track of the IPC 2011 in Section 3.3.1.1, page 43. The set of candidate planners used in Section 3.3.3.1 (see page 48) to assess the performance of the GOP portfolios in optimal planning is shown in Table A.2.

Planner	Authors	Source
Blind	Silvia Richter *et al.*	FDSS-2 planner
BJOLP	Erez Karpas *et al.*	IPC 2011
CPT4	Vincent Vidal	IPC 2011
FD Autotune	Chris Fawcett *et al.*	IPC 2011
Fork Init	Michael Katz *et al.*	IPC 2011
Gamer	Peter Kissmann *et al.*	IPC 2011
IFork Init	Michael Katz *et al.*	IPC 2011
LM-cut	Malte Helmert *et al.*	IPC 2011
LMFork	Michael Katz *et al.*	IPC 2011
M&S-bisim 1	Raz Nissim *et al.*	FDSS-1 planner
M&S-bisim 2	Raz Nissim *et al.*	FDSS-1 planner
Selective Max	Erez Karpas *et al.*	IPC 2011

Table A.1: Optimal planners considered from the IPC 2011.

The OSS portfolio for the IPC 2011 sequential satisficing track has been derived in Section 3.3.1.2 (see page 44) with the set of candidate planners shown in Table A.3. The assessment of GOP to configure sequential portfolios in satisficing planning has been described in Section 3.3.3.2, which starts on page 51. The performance of the portfolios automatically derived with GOP has been compared against FDSS using the sets of candidate planners shown in Tables A.4 and A.5, where each planner is defined by a search algorithm, an evaluation method and a set of heuristics.

Finally, Table A.6 shows the set of candidate solvers used to compute the OSS portfolio for the open track of the SAT Competition 2013 in Section 3.3.1.3, page 44.

Planner	Authors	Source
Blind	Silvia Richter *et al.*	FDSS-2 planner
BJOLP	Erez Karpas *et al.*	IPC 2011
h^1 landmarks	Erez Karpas *et al.*	FDSS-1 planner
h^{max} landmarks	Malte Helmert *et al.*	FDSS-1 planner
LM-cut	Malte Helmert *et al.*	IPC 2011
M&S-bisim 1	Raz Nissim *et al.*	FDSS-1 planner
M&S-bisim 2	Raz Nissim *et al.*	FDSS-1 planner
M&S-LFPA 10000	Malte Helmert *et al.*	FDSS-1 planner
M&S-LFPA 50000	Malte Helmert *et al.*	FDSS-1 planner
M&S-LFPA 100000	Malte Helmert *et al.*	FDSS-1 planner
RHW landmarks	Erez Karpas *et al.*	FDSS-1 planner

Table A.2: Optimal planners considered by Fast-Downward Stone Soup.

Planner	Authors	Source
ACOPlan	Marco Baioletti *et al.*	IPC 2011
ACOPlan2	Marco Baioletti *et al.*	IPC 2011
Arvand	Hootan Nakhost *et al.*	IPC 2011
BRT	Vidal Alcázar *et al.*	IPC 2011
CBP	Raquel Fuentetaja	IPC 2011
CBP2	Raquel Fuentetaja	IPC 2011
Roamer	Qiang Lu *et al.*	IPC 2011
CPT4	Vincent Vidal	IPC 2011
DAE-YAHSP	Johann Dréo *et al.*	IPC 2011
FD Autotune 1	Chris Fawcett *et al.*	IPC 2011
FD Autotune 2	Chris Fawcett *et al.*	IPC 2011
FDSS 1	Malte Helmert *et al.*	IPC 2011
FDSS 2	Malte Helmert *et al.*	IPC 2011
Fork Uniform	Michael Katz *et al.*	IPC 2011
LAMA 2008	Silvia Richter *et al.*	IPC 2011
LAMA 2011	Silvia Richter *et al.*	IPC 2011
Lamar	Alan Olsen *et al.*	IPC 2011
LPRPG-P	Amanda Coles *et al.*	IPC 2011
Madagascar	Jussi Rintanen	IPC 2011
Madagascar-p	Jussi Rintanen	IPC 2011
POPF2	Amanda Coles *et al.*	IPC 2011
Probe	Nir Lipovetzky *et al.*	IPC 2011
Randward	Alan Olsen *et al.*	IPC 2011
SATPLANLM-C	Dunbo Cai *et al.*	IPC 2011
Sharaabi	Bharat Ranjan	IPC 2011
YAHSP2	Vincent Vidal	IPC 2011
YAHSP2-MT	Vincent Vidal	IPC 2011

Table A.3: Satisficing planners considered from the IPC 2011.

Search	Planner Evaluation	Heuristics	Source
Greedy best-first	Eager	FF	FD IPC 2011
Weighted-A* w=3	Lazy	FF	FD IPC 2011
Greedy best-first	Eager	FF, CG	FD IPC 2011
Greedy best-first	Eager	ADD, FF, CG	FD IPC 2011
Greedy best-first	Eager	FF, CG, CEA	FD IPC 2011
Greedy best-first	Eager	FF, CEA	FD IPC 2011
Greedy best-first	Eager	ADD, FF, CG, CEA	FD IPC 2011
Greedy best-first	Eager	ADD, FF	FD IPC 2011
Greedy best-first	Eager	ADD, CG, CEA	FD IPC 2011
Greedy best-first	Eager	ADD, FF, CEA	FD IPC 2011
Greedy best-first	Eager	CG, CEA	FD IPC 2011
Greedy best-first	Eager	ADD, CG	FD IPC 2011
Greedy best-first	Lazy	FF	FD IPC 2011
Greedy best-first	Eager	CEA	FD IPC 2011
Greedy best-first	Eager	ADD, CEA	FD IPC 2011
Greedy best-first	Lazy	FF, CG, CEA	FD IPC 2011
Weighted-A* w=3	Eager	FF	FD IPC 2011
Greedy best-first	Eager	ADD	FD IPC 2011
Greedy best-first	Lazy	FF, CEA	FD IPC 2011
Greedy best-first	Lazy	FF, CG	FD IPC 2011
Greedy best-first	Lazy	ADD, FF, CG, CEA	FD IPC 2011
Greedy best-first	Lazy	ADD, FF, CEA	FD IPC 2011
Greedy best-first	Lazy	ADD, FF, CG	FD IPC 2011
Greedy best-first	Lazy	ADD, FF	FD IPC 2011
Weighted-A* w=3	Lazy	CEA	FD IPC 2011
Weighted-A* w=3	Eager	CEA	FD IPC 2011
Greedy best-first	Lazy	CG, CEA	FD IPC 2011
Greedy best-first	Lazy	ADD, CEA	FD IPC 2011
Greedy best-first	Lazy	ADD, CG, CEA	FD IPC 2011
Greedy best-first	Lazy	ADD, CG	FD IPC 2011
Weighted-A* w=3	Lazy	ADD	FD IPC 2011
Weighted-A* w=3	Eager	ADD	FD IPC 2011
Greedy best-first	Lazy	CEA	FD IPC 2011
Weighted-A* w=3	Eager	CG	FD IPC 2011
Greedy best-first	Lazy	ADD	FD IPC 2011
Greedy best-first	Eager	CG	FD IPC 2011
Weighted-A* w=3	Lazy	CG	FD IPC 2011
Greedy best-first	Lazy	CG	FD IPC 2011

Table A.4: Satisficing planners considered by FDSS-1.

Search	Planner Evaluation	Heuristics	Source
Greedy best-first	Eager	FF	FD IPC 2011
Greedy best-first	Lazy	FF	FD IPC 2011
Greedy best-first	Eager	CEA	FD IPC 2011
Greedy best-first	Eager	ADD	FD IPC 2011
Greedy best-first	Lazy	CEA	FD IPC 2011
Greedy best-first	Lazy	ADD	FD IPC 2011
Greedy best-first	Eager	CG	FD IPC 2011
Greedy best-first	Lazy	CG	FD IPC 2011

Table A.5: Satisficing planners considered by FDSS-2.

Planner	Authors	Source
CSHC par8	Yuri Malitsky *et al.*	SAT 2013
MIPSat	Sergio Núñez *et al.*	SAT 2013
GlucoRed + March r531	Siert Wieringa	SAT 2013
interact_open 1.0	Jingchao Chen	SAT 2013
Glucans strict	Xiaojuan Xu *et al.*	SAT 2013
Solver43 a	Valeriy Balabanov	SAT 2013
Solver43 b	Valeriy Balabanov	SAT 2013
forl nodrup	Mate Soos	SAT 2013
GlueMiniSat 2.2.7j	Hidetomo Nabeshima *et al.*	SAT 2013
MINIPURE 1.0.1	Hsiao-Lun Wang	SAT 2013

Table A.6: SAT solvers considered from the SAT Competition 2013.