University of Basel

(Near)-optimal policies for Probabilistic IPC 2018 domains

Master's Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Thomas Keller

Brikena Çelaj
brikena.celaj@unibas.ch
17-063-462

30.05.2020

Abstract

The International Planning Competition (IPC) is a competition of state-of-the-art planning systems. The evaluation of these planning systems is done by measuring them with different problems. It focuses on the challenges of AI planning by analyzing classical, probabilistic and temporal planning and by presenting new problems for future research. Some of the probabilistic domains introduced in IPC 2018 are Academic Advising, Chromatic Dice, Cooperative Recon, Manufacturer, Push Your Luck, Red-finned Blue-eyes, etc.

This thesis aims to solve (near)-optimally two probabilistic IPC 2018 domains, Academic Advising and Chromatic Dice. We use different techniques to solve these two domains. In Academic Advising, we use a relevance analysis to remove irrelevant actions and state variables from the planning task. We then convert the problem from probabilistic to classical planning, which helped us solve it efficiently. In Chromatic Dice, we implement backtracking search to solve the smaller instances optimally. More complex instances are partitioned into several smaller planning tasks, and a near-optimal policy is derived as a combination of the optimal solutions to the small instances.

The motivation for finding (near)-optimal policies is related to the IPC score, which measures the quality of the planners. By providing the optimal upper bound of the domains, we contribute to the stabilization of the IPC score evaluation metric for these domains.

Acknowledgments

Writing this thesis would not have been possible without the major support of many remarkable people in all aspects of my life.

I would like to start by thanking Prof. Dr. Malte Helmert and Dr. Thomas Keller. Prof. Dr. Malte Helmert, for the opportunity to write this thesis on the AI research group, which was both productive and entertaining. Dr. Thomas Keller, for being the most supportive and helpful supervisor. In the last years that I have known him, he has always amazed me with his expertise and dedication during the lectures. His enthusiasm for his research and his patience has motivated me to work with him. As a supervisor, he has not only given me constructive feedback and crucial insights into the thesis, but he has also taught me general knowledge related to his research and scientific writing.

Furthermore, I want to thank my friends, Manvi Bhatia, Niluka Piyasinghe and Augusto Blaas Corrêa for helpful discussions, suggestions and corrections throughout all the studies. Especially, I want to thank Drilon Vukaj, for continuously supporting me in both professional and mental aspects.

Finally, I want to thank my family for always being by my side no matter what. To my parents Xhelal Çelaj and Dashurije Berisha, for always motivating me and giving me the possibility to follow my dreams. To my brother Genc, for encouraging me to be persistent all throughout my studies. Last, but not least, I want to thank my uncle Sali Çelaj and his wonderful family for being such an inspiration and for bringing to life my immense desire to study abroad.

Contents

Chapter 1

Introduction

Two parts of planning are classical planning and probabilistic planning. In the former, the actions are deterministic while in the latter the actions are stochastic and exogenous events are considered.

The International Planning Competition (IPC) is a competition of different tracks in several parts of planning. It focuses on the challenges of AI planning by analyzing state-of-the-art planning systems in different problems. The IPC 2018 tracks classical, probabilistic and temporal planning. Some probabilistic domains introduced in IPC 2018 are Academic Advising, Chromatic Dice, Cooperative Recon, Manufacturer, Push Your Luck, Red-finned Blue-eyes, etc. In this paper, we will focus on the probabilistic part of the competition and compute the optimal or near-optimal policies for two domains, Academic Advising and Chromatic Dice [1].

The IPC score measures the quality of the planner systems that take place in the IPC competition. A planner evaluation, based on the IPC score without having an optimal upper bound, is very flawed[1]. The ranking of the planners is done based on the planner's performance in each instance. Each new participant can change the ranking of the current participants. As a consequence, the winner might change. Since optimal upper bounds are required to stabilize the evaluation metric, we need to find these optimal state values for the initial state.

This thesis aims to find a (near)-optimal policy for two probabilistic IPC 2018 domains. In this way, we contribute to the stabilization of the IPC score evaluation metric for these domains. The first part studies the Academic Advising domain. We use a relevance analysis which has a large impact on reducing the complexity of the problem. Here, the agent does not have to look all the available actions and state variables of the planning task. We then present a new solution for this domain, where we switch from probabilistic planning to deterministic planning by converting the probabilities in expected costs. Using Fast Downward, we are able to come up with lower bounds in cost of some instances. The second problem is the Chromatic Dice domain where we present a (near)-optimal solution. This is done by analyzing and representing the state space compactly. To solve the second domain optimally, we use a Backtracking method. If we compare our implementation with the best planners that already are there, we expect to see that the best planners are very far from optimal. That being said, our solution may not be optimal either, but we assume it is closer. Chapter 2 contains the background knowledge that is needed for this thesis. Chapter 3 and Chapter 4 describe and analyse two objectives, Academic Advising and Chromatic Dice domain, introduced in IPC 2018.

Chapter 2

Background

In this section, we are going to give a brief introduction to planning, where specifically we explain two parts of planning, classical and probabilistic planning.

2.1 Classical Planning

Planning is a problem of finding a path that leads to a goal state from a given initial state using a sequence of actions.

Definition 2.1.1. *(Proposition Planning Task)* A Propositional Planning Task *is a 4-tuple* $\Pi = \langle V, I, O, \gamma \rangle$ *where:*

- *V is a finite set of proposition variables,*
- *I is an initial state, an assignment of all $v \in V$,*
- *O is a finite set of operators as defined below and*
- *γ is a partial variable assignment over V which is called the goal.*

Instead of writing sets of partial variable assignment, we will replace it with literals, i.e. we write $(a, \neg b)$ instead of $\{a \rightarrow true, b \rightarrow false\}$.

Definition 2.1.2. *(Operator) An* operator *O over state variables V is an object with three properties:*

- *a precondition pre(o), a formula over V,*
- *an effect eff(o), a partial variable assignment over V and*
- *a cost cost(o)* $\in \mathbb{R}_0^+$

Operators are also called actions and they are written as triples of precondition, effect and cost $\langle \mathrm{pre(o)}, \mathrm{eff(o)}, \mathrm{cost(o)} \rangle$, or in abbreviated version $\langle \mathrm{pre(o)}, \mathrm{eff(o)} \rangle$ where the cost is irrelevant.

An Operator o is applicable in state s iff $s \models \mathrm{pre(o)}$. We write $A(s)$ for the set $\{o \in O\}$ that is applicable.

Definition 2.1.3. *(Resulting State) The resulting state* $s[o] = s'$ *of applying operator* o *in state* s *is the state* s' *where for all* $v \in V$

$$s'(v) = \begin{cases} d, & \textit{if } v \rightarrow d \in \textit{eff(o)} \\ s(v), & \textit{otherwise} \end{cases}$$

A typical example of planning is the blocks world domain. This domain is about blocks positioned on a table or on top of another block, and each block can be moved by a robot. However, if there is a block under another, the robot cannot move it, and only one block can fit on top of another. The goal is to put blocks in a certain position. For example, one goal could be putting blocks A and B on the table and block C on top of B (see Figure 2-1)[8]. An instance of a blocks world problem would be as shown in Figure 2-1, which illustrates the initial state and the goal state of the instance. There is a set of variables V that contains variables such as on(A,B), on(B,C), clear(B), clear(C). The variables, e.g. on(A,B), mean that the block B is on top of A, and the variables, e.g. clear(B), tell us there is no block on top of B. Some operators would be, for instance, move(B, A, C) which means move block A from block B to block C.

A Propositional Planning Task can be mapped into a Transition System.

Definition 2.1.4. *(Transition System) A* Transition System *is a 6-tuple* $\mathcal{T} = \langle S,\ A,\ c,\ T,\ s_0,\ S_* \rangle$ *where*

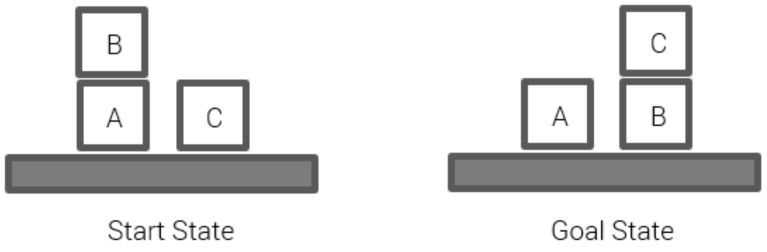

Figure 2-1: Blocks world problem

- *S is a finite set of states,*
- *A is a finite set of labels,*
- *$c : A \rightarrow \mathbb{R}_0^+$ is a cost function,*
- *$T \subseteq S \times A \times S$ is the relation of the transition,*
- *$s_0 \in S$ is the initial state and*
- *$S_* \subseteq S$ is the set of goal states.*

Definition 2.1.5. *(Transition System Induced by a Propositional Planning Task) The* propositional Planning Task $\Pi = \langle V, I, O, \gamma \rangle$ *induces the Transition System* $\mathcal{T}(\Pi) = \langle S, A, c, T, s_0, S_* \rangle$ *where*

- *S is a set of all valuations of V ($S = 2^V$),*
- *$A = O$,*
- *$c(o) = cost(o)$ for all operators $o \in O$,*
- *$T = \{\langle s, o, s' \rangle | s \in S,\ o \in A(s),\ s' = s[o]\}$,*
- *$s_0 = I$ and*
- *$S_* = \{s \in S | \gamma \subseteq S\}$.*

A sequence of operators that forms a solution of $\mathcal{T}(\Pi)$ is called a plan of Π.

2.2 Probabilistic Planning

Different decisions are made every day in our lives which we try to achieve better performance with. Those decisions are mostly made under uncertainty.

In probabilistic planning, the agent cannot execute any plan as in classical planning because the actions are non-deterministic. As a consequence of non-determinism, applying an action can lead to different outcomes. Therefore, it is important to find a more general solution than a plan. In probabilistic planning, this solution is called a policy. A policy should be able to perform cycles and to branch over the outcomes.

Definition 2.2.1. *(Probabilistic Planning Task) A* Probabilistic Planning Task *is a 4-tuple* $\Pi = \langle V, I, O, H \rangle$ *where*

- V *is a finite set of propositional variables,*
- I *is an initial state,*
- O *is a finite set of probabilistic operators as defined below and*
- H *is the horizon.*

Definition 2.2.2. *(Operator) A* probabilistic operator *is an object with three properties:*

- *a precondition pre(o), a formula over* V*,*
- *a probabilistic effect eff(o) as defined below and*
- *a reward, where reward(o)* $\in \mathbb{R}$

Definition 2.2.3. *(Probabilistic effect) A* probabilistic effect *is a set of tuples* $\{\langle e_1, p_1 \rangle$, ..., $\langle e_n, p_n \rangle\}$ *where* $p_i \in (0, 1]$ *for* $1 \leq i \leq n$, $\sum_{i=1}^{n} p_i$ *and* e_i *is a partial variable assignment over* V *for* $1 \leq i \leq n$.

Let us take an example and explain the difference between the classical and probabilistic operator. While classical operator uses a deterministic effect, the probabilistic operator uses a probabilistic effect.

Example 2.2.4. *(Classical and Probabilistic operators) One example of an operator is* $o = \langle \top, \{a, b\}, 1 \rangle$. *As a precondition we have* $\top$, *the cost is 1 and the effect is* a, b. *However, since in Probabilistic Planning Task the effect is a distribution, it differs from the one above. We show the differences between the effects below:*

Classical Planning effect:

eff(o) $= \{a, b\}$

Probabilistic Planning effect :

$$\textit{eff(o)} = \{\langle\{a, b\}, 0, 2\rangle, \langle\{\neg a, c, \neg d\}, 0, 5\rangle, \langle\{\neg b, d\}, 0, 3\rangle\}$$

Probabilistic Planning problems are usually modelled as Markov Decision Processes (MDP). In the sequential decision, first, the agent accepts the state and then it chooses which action to perform next. Execution of action sends the agent to a new state of the environment. As a result, the agent obtains a specific reward. The goal is to choose actions sequentially in a way that the overall system gives us an optimal solution by maximizing the expected reward or minimizing the expected cost. This kind of sequential decision model is the Markov decision process model. There are different types of MDPs, e.g. Stochastic Shortest Path (SSP), Finite-horizon MDP (FH-MDPs) and Discounted Reward MDP (DR-MDPs). SSPs are considered as classical planning with a probabilistic transition function. FH-MDPs are acyclic MDPs with a finite number of steps (finite horizon). DR-MDPs discount the reward over an infinite number of steps (have an infinite horizon)[7]. We use finite-horizon MDPs to define a theoretical overview of our problems, therefore we define the MDPs in the FH-MDPs setting.

Definition 2.2.5. *(Markov Decision Process) A* Markov decision Process *is a tuple* $\mathcal{T} = \langle S, A, R, T, s_0, h \rangle$, *where*

- S *is a finite set of states,*
- A *is a finite set of (transition) labels,*
- $R : A \to \mathbb{R}$ *is the reward function,*

- *$T : S \times A \times S \to [0, 1]$ is the transition function,*
- *$s_0 \in S$ is the initial state and*
- *$h \in \mathbb{N}$ is a finite horizon.*

For all $s \in S$ and $a \in A(s)$, it is required that $\sum_{s' \in S} T(s, a, s') = 1$.

Definition 2.2.6. *(Applicable actions) The set of* applicable actions*, written as $A(s)$, is a set of labels for which $\sum_{s' \in S} T(s, a, s') \neq 0$.*

In a system there are a finite number of states that are denoted by S and the current state of the agent is denoted by $s \in S$. Let $A = \bigcup_{s \in S} A(s)$ be the set of all actions and $A(s)$ be a set of actions that can be taken in state s. From the state s, the agent can choose action $a \in A(s)$. If we make this choice, as a result, the agent will receive a reward $R(s, a)$, and the next state will be s' which will be sampled according to a transition function. The transition function, in this case, is $T(s, a, s')$.

Example 2.2.7. *Figure 1 shows an example of MDP. Actions are represented with circles, and states with rectangles. Each action has a reward denoted in the incoming edge. There are one or more outgoing edges for each action, which are connected to the outcomes and have probabilities that correspond to each of them. The MDP of this example is $M = \langle S, A, T, R, s_0, h \rangle$, where*

- *$S = \{s_0, s_1, s_2\}$ is a set of states,*
- *$A = \{a_1, a_2, a_3, a_4, a_5\}$ is a set of actions,*
- *s_0 is the initial state,*
- *$h = 2$ and*
- *transition and reward function are given as bellow*

- $T(s_0, a_1, s_1) = \frac{1}{2}$	$R(a_1) = 2$
- $T(s_0, a_1, s_2) = \frac{1}{2}$	

- $T(s_0, a_2, s_2) = 1$	$R(a_2) = 0.5$
- $T(s_2, a_3, s_1) = \frac{2}{3}$	$R(a_3) = 1$
- $T(s_2, a_3, s_0) = \frac{1}{3}$	
- $T(s_2, a_4, s_1) = 1$	$R(a_4) = 1$
- $T(s_1, a_5, s_0) = 1$	$R(a_5) = 0.5$

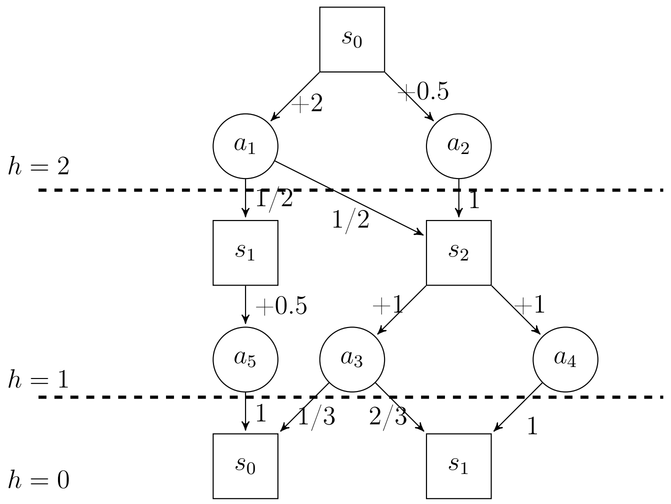

Figure 2-2: An MDP example

Let us now describe how we can map a Probabilistic Planning Task into an MDP.

Definition 2.2.8. *(MDP induced by a Probabilistic Planning Task) The Probabilistic Planning Task* $\Pi = \langle V,\ I,\ O,\ H\rangle$ *induces the MDP defined as* $\mathcal{T}(\Pi) = \langle S,\ A,\ R,\ T,\ s_0,\ h\rangle$*, where*

- $S = 2^V$,
- $A = O$,
- $R(o) = reward(o)$ *for all* $o \in O$,
- $T(s, o, s') = P$, *where* $P = \prod_{i=1}^{n} p_i$ *for all* $\langle e, p_i\rangle \in$ *eff(o) with* $s[e] = s'$,
- $s_0 = I$ *and*
- $h = H$

A policy for an MDP is defined as below:

Definition 2.2.9. *Let $\mathcal{T} = \langle S,\ A,\ R,\ T,\ s_0,\ h\rangle$ be an MDP. A policy for $\mathcal{T}$ is a mapping $\pi : S \times \{1, 2, ..., h\} \rightarrow A(s) \cup \{\perp\}$*

In the following definitions, we explain what a value function and Bellman optimality equation are.

Definition 2.2.10. *(MDP Value function) Let $\mathcal{T} = \langle S, A, R, T, s_0, h\rangle$ be an MDP. The state-value $V_\pi(s, d)$ of a state $s \in S$ with d steps to go is given as*

$$V_\pi(s, d) := \begin{cases} 0, & \text{if } d = 0 \\ Q_\pi(s, d, \pi(s)), & \text{otherwise} \end{cases}$$

where $Q_\pi(s, d, a)$ is the action value and is given as

$$Q_\pi(s, d, a) := R(s, a) + \sum_{s' \in S} [T(s, a, s') \cdot V_\pi(s', d - 1)]$$

Definition 2.2.11. *(MDP Bellman Optimality equation) Let $\mathcal{T}$ be an MDP. The Bellman optimality equation for a state $s \in S$ with d steps to go is defined as*

$$V_*(s, d) := \begin{cases} 0, & \text{if } d = 0 \\ \max_{a \in A(s)} Q_*(s, d, a), & \text{otherwise} \end{cases}$$

where $Q_*(s, d, a)$ is defined as

$$Q_*(s, d, a) := R(s, a) + \sum_{s' \in S} [T(s, a, s') \cdot V_*(s', d - 1)]$$

Let π^* be a policy that is greedy w.r.t. Q_*,

$$i.e.\ \pi^*(s, d) = max_{a \in A} Q_*(s, d, a)$$

The policy π^* is optimal if $\pi^*(s, d) \in argmax_{a \in A(s)} Q_*(s, d, a)$ for all states where $s \in S$, $d \in \{1, 2, ..., h\}$ and the expected reward is $V_*(s, d)$.

A policy π is optimal if for all policies π', the reward of π is the largest. The optimal policy is described by a Bellman optimality equation. The expected reward of the policy is described by state-value of the policy.

Chapter 3

Academic Advising

Academic Advising is a domain where a student during the studies chooses which courses to take to graduate from their program as fast as possible.

3.1 Academic Advising Domain

Definition 3.1.1. *An* Academic Advising instance *is a 9-tuple* $A = \langle C, C^*, P, cost, cost^+, h, \sigma, penalty, \mathbb{P}\rangle$, *where*

- C *is a set of* courses*,*
- $C^* \subseteq C$ *is the set of* program requirement courses*,*
- $P : C \to 2^C$ *is the course* prerequisites function*,*
- $cost \in \mathbb{N}_0$ *is the* cost *for taking a course for the* first time*,*
- $cost^+ \in \mathbb{N}_0$ *is the* cost *for taking a course except for the first time,*
- $h \in \mathbb{N}$ *is the* horizon*,*
- $\sigma \in \mathbb{N}$ *is* concurrency*, the number of courses per semester,*
- $penalty \in \mathbb{N}$ *is the* program incomplete penalty *and*
- $\mathbb{P} : C \to [0, 1]$ *is the* prior probability*.*

Each instance of the domain has a specific number of courses C, where some of them are considered program requirement courses C^*. To graduate from the program, the student has to complete all the required courses. Taking and retaking a course have a cost which is specified in the domain. Prior probabilities of passing a course are increased by passing the prerequisites of the course. Based on a given concurrency, multiple courses can be taken per semester. We add a penalty each time we choose an action while the program is not completed.

In the initial state, none of the courses are passed or taken. To pass a course, we have to take one of the applicable operators that the state has, including the *noop* operator. We describe a state of probabilistic planning task over three variables: $passed_c$, which denotes if the course c is passed or not; $taken_c$, which denotes if the course c has already been taken before or not yet; complete, which is assigned as true only if the program is completed (all required courses are passed).

Let $\psi \subseteq C$. Then we define:

$$P(\psi) = \bigcup_{c \in \psi} P(c) \tag{3.1}$$

$P(\psi)$ represents the union of the preconditions of a set of courses ψ,

$$Probability_{c,C_P} = \mathbb{P} + [(1 - \mathbb{P}(c)) \cdot \frac{|C_P|}{1 + |P(c)|}] \tag{3.2}$$

$Probability_{c,C_P}$ is the probability of passing a course c, given that the prerequisite courses $C_P \subseteq P(c)$ are passed,

$$Prob_{\psi,\psi',C_P} = \prod_{c \in \psi} Probability_{c,C_P} \cdot \prod_{c \in \psi'} (1 - Probability_{c,C_P}) \tag{3.3}$$

$Prob_{\psi,\psi',C_P}$ is the probability that for a set of courses $\psi \cup \psi'$, the courses in ψ are passed and the courses in ψ are failed if the courses $C_P \subseteq P(\psi \cup \psi')$ have already been passed.

$$\hat{\psi} = \psi^0 \cup \psi^1 \tag{3.4}$$

We denote the union of ψ^0 and ψ^1 with $\hat{\psi}$.

An Academic Advising instance $A = \langle C, C^*, P, cost, cost^+, h, \sigma, penalty, \mathbb{P} \rangle$ induces a Probabilistic Planning Task $\Pi = \langle V,\, I,\, O,\, H \rangle$, where

- $V = \{passed_c | c \in C\} \cup \{taken_c | c \in C\} \cup \{complete\}$
- $I = \{\neg passed_c | c \in C\} \cup \{\neg taken_c | c \in C\} \cup \{\neg complete\}$
- $H = h$
- $O = \bigcup_{\psi^0, \psi^1 \in 2^C, \psi^0 \cap \psi^1 = \emptyset, 1 \leq |\hat{\psi}| \leq \sigma} O_{\hat{\psi}} \cup \{noop, noop^+\}$ where

$$O_{\psi^0,\psi^1} = \{o_{\psi^0,\psi^1,C_P} | C_P \subseteq P(\hat{\psi})\} \cup \{\hat{o}_{\psi^0,\psi^1,C_P} | C_P \subseteq P(\hat{\psi})\}$$

We give below the precondition, effect and the reward for each of the operators. We divide the operators into four types: $o_{\hat{\psi},C_P}$, $\hat{o}_{\hat{\psi},C_P}$, *noop* and *noop+*. The hat is used when we reach a goal, while the operators without the hat are applicable to other states.

- o_{ψ^0,ψ^1,C_P}

$$\text{pre}(o_{\psi^0,\psi^1,C_P}) = \bigwedge_{c \in \hat{\psi}} \neg passed_c \wedge \bigwedge_{c \in \psi^0} \neg taken_c \wedge \bigwedge_{c \in \psi^1} taken_c \wedge$$
$$\bigwedge_{c \in C_P} passed_c \wedge \bigwedge_{c \in P(\hat{\psi}) \setminus C_P} \neg passed_c \wedge \exists_{c \in C^*, c \notin \hat{\psi}} \neg passed_c,$$
$$\text{eff}(o_{\psi^0,\psi^1,C_P}) = \bigcup_{\psi' \in 2^{\hat{\psi}}} \Big\langle \bigcup_{c \in \psi'} \{passed_c\} \cup \bigcup_{c \in \hat{\psi}} \{taken_c\}, Prob_{\psi', \hat{\psi} \setminus \psi', C_P} \Big\rangle,$$
$$\text{reward}(o_{\psi^0,\psi^1,C_P}) = -|\psi^0| \cdot cost - |\psi^1| \cdot cost^+ - penalty$$

- $\hat{o}_{\psi^0,\psi^1,C_P}$

$$\text{pre}(\hat{o}_{\psi^0,\psi^1,C_P}) = \bigwedge_{c \in \psi} \neg passed_c \wedge \bigwedge_{c \in \psi} \neg taken_c \wedge \bigwedge_{c \in C_P} passed_c \wedge$$
$$\forall_{c \notin \psi \in C^*} passed_c,$$
$$\text{eff}(o_{\psi^0,\psi^1,C_P}) = \bigcup_{\psi' \in 2^{\psi}} \Big\langle \bigcup_{(c \in \psi'} \{passed_c\} \cup \bigcup_{c \in \psi} \{taken_c\} \cup \{complete\}, Prob_{\psi', \psi \setminus \psi', C_P} \Big\rangle,$$
$$\text{reward}(o_{\psi^0,\psi^1,C_P}) = -|\psi^0| \cdot cost - |\psi^1| \cdot cost^+ - penalty$$

- *noop*

$$\begin{aligned}
&\text{pre(noop)} = \neg \textit{complete},\\
&\text{eff(noop)} = \langle \emptyset, 1 \rangle,\\
&\text{reward(noop)} = -\textit{penalty}
\end{aligned}$$

- *noop*$^+$

$$\begin{aligned}
&\text{pre}(\textit{noop}^+) = \textit{complete},\\
&\text{eff}(\textit{noop}^+) = \langle \emptyset, 1 \rangle,\\
&\text{reward}(\textit{noop}^+) = 0
\end{aligned}$$

3.2 Relevance Analysis

An Academic Advising problem is hard. As shown in the 4^{th} column of Table 3.1, the number of variables in the smallest instance is 31. This means that this instance has more than a trillion states due to the exponential relationship. So far, no solver can solve such a huge state space. Therefore, the first step on finding the optimal solution is simplifying the problem.

Let us take the first instance of Academic Advising 2018 domain as an example. There are 15 courses in total, where five of them are in program requirement, $C^* = \{C0001, C0002, C0202, C0101, C0300\}$. The domain can be interpreted in a graph in the following way: With double circles, we represent the courses that have to be taken to complete the degree, while with single circles we represent the other ones. The arrows that connect a course with another tell us which course is a prerequisite of which one. For example, the course C0001 and C0002 have no prerequisites, the course C0202 has as a prerequisite the course C0002 and so on. Figure 3-1 shows the relationship between all the courses of this instance within a graph.

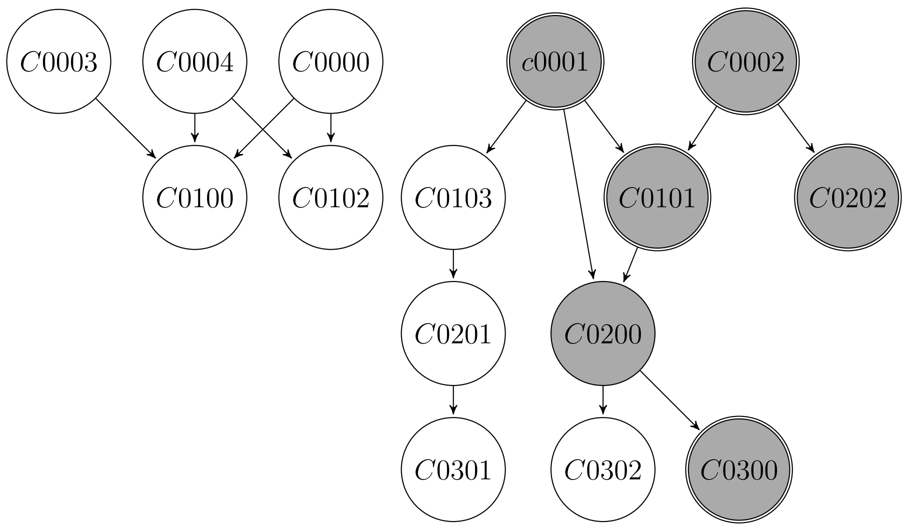

Figure 3-1: The graph of the first instance and the pruned version of it colored in grey.

A Directed Acyclic Graph (DAG) is a graph that consists of vertices and edges. Each directed edge points from one vertex to another. The path formed by these edges has no cycles. As an observation, we can state that all Academic Advising instances of 2014 and 2018 are represented by a DAG.

Definition 3.2.1. *Let the* program requirement DAG *of A be the graph $D = \langle V, E \rangle$, where $V = C$; and where for all $c_1, c_2 \in C, \langle c_1, c_2 \rangle \in E$ iff $c_2 \in P(c_1)$.*

From the above example, we can observe that passing or failing some of the courses like $C0000$, $C0102$, etc., affect anything. This is a very important observation and we can conclude that all leaf nodes that represent the non-mandatory courses can be pruned because we do not take them into consideration even under the optimal policy.

Definition 3.2.2. *Let A be an Academic Advising instance with a set of courses C and let D be the program requirement DAG of A. We call a course $c \in C$* relevant *for A iff there is a path $\langle c_1, ...c_n \rangle$ in D such that $c_1 = c$ and $c_n \in C^*$*

Our pruning algorithm prune nodes by looking at the leaves. In every iteration we prune a leaf that is not in program requirements until there are no more such leaves.

Algorithm 1: The pruning algorithm

```
function PRUNE(G)
$G = \langle V, E \rangle$
vertices := G.getVertices()
queue := $\emptyset$
foreach $v \in vertices$ where $v.isLeaf()$ and not $v.isRequired()$ do
    queue.insert(v)
end
while $queue \neq \emptyset$ do
    $E' = \{(v,u) \mid v \in V\}$
    $E \leftarrow E \setminus E'$
    $V \leftarrow V \setminus \{u\}$
    foreach $(v,u) \in E'$ do
        if v.isLeaf() and not v.isRequired() then
            queue.insert(v)
        end
    end
end
return graph
```

After we prune the graph, we end up with a simplified graph which contains only relevant courses. We show the result in Figure 3-1 colored in grey.

Theorem 1. *Let $A = \langle C, C^*, P, cost, cost^+, h, \sigma, penalty, \mathbb{P} \rangle$ be an Academic Advising instance, $\mathcal{T} = \langle S, A, R, T, s_0, H \rangle$ be the MDP induced by A, $A' = \langle C', C^*, P', cost, cost^+, h, \sigma, penalty, \mathbb{P} \rangle$ an Academic Advising instance where $C' \subseteq C$ is the set of relevant courses for A and $P' : C' \rightarrow 2^{C'}$ where $P'(c) = P(c)\ \forall c \in C'$ and $\mathcal{T}' = \langle S', A', R', T', s'_0, H' \rangle$ the MDP induced by A'. Then*

$$V_*(s_0, H) = V_*(s'_0, H')$$

Proof. For every policy in the original task, π for $\mathcal{T}$, there is a policy in pruned task,

π' for $\mathcal{T}'$, that is at least as good as π.

$$V^{\pi'}(s'_0, H') \leq V^{\pi}(s_0, H)$$

We know that if $\pi(s) = o_{\{c_1,c_2\},C_P} \Rightarrow \pi'(\hat{s}) = o_{\{c_1,c_2\}\cap C',C_P}$. If $\{c_1, c_2\} \cap C' = \emptyset$ then we need to apply *noop* or *noop*$^+$ (depending on which one is applicable in that state). From this, we derive that $\pi'(s) = noop$ if $\pi(s) = a$ and $a \notin A'$. Since the cost of *noop* is always less than taking a course, the above in-equation holds.

Assume that there is an optimal policy π^* in $\mathcal{T}$, then there is also an optimal policy π'^* in $\mathcal{T}$. This holds because throwing any operator change none of the probabilities of achieving a state in the future. $\square$

We provide a table where we see the difference between the two instances, A and A' (Table 3.1).

Table 3.1: Comparison of instance A and instance A' in IPC 2018 domain.

Instance	|C|	$\|C'\|$	|V|	$\|V'\|$	Reduction of |V| in %
01	15	6	31	13	$55,06\%$
02	15	10	31	21	$32,26\%$
03	24	5	49	11	$77,55\%$
04	27	6	55	13	$76,36\%$
05	25	14	51	29	$43,14\%$
06	30	17	61	35	$42,62\%$
07	40	25	81	51	$37,04\%$
08	31	17	63	35	$44,44\%$
09	44	20	89	41	$53,93\%$
10	62	29	125	59	$52,80\%$
11	58	26	117	53	$54,70\%$
12	64	29	129	59	$54,26\%$
13	59	33	119	67	$43,70\%$
14	93	43	187	87	$53,48\%$
15	116	46	233	93	$60,09\%$
16	96	49	193	99	$48,70\%$
17	150	67	301	135	$55,15\%$
18	233	85	467	171	$63,38\%$
19	217	83	435	167	$61,61\%$
20	278	76	557	153	$72,53\%$

The IPC 2014 introduced the Academic Advising domain as well. Table 3.2 shows the result of the relevance analysis in its instances. The average shrinking of IPC 2014 domain is 24.3%. Compared to Academic Advising domain of IPC 2014, our relevance analysis is around double times more efficient in the Academic Advising domain of IPC 2018.

Table 3.2: Comparison of instance B and instance B' in IPC 2014 domain.

Instance	|C|	$\lvert C'\rvert$	|V|	$\lvert V'\rvert$	Reduction of |V| in %
01	10	5	21	11	$47,62\%$
02	10	10	21	21	$0,00\%$
03	15	5	31	11	$64,52\%$
04	15	14	31	29	$6,45\%$
05	20	17	41	35	$14,63\%$
06	20	14	41	29	$29,27\%$
07	25	17	51	35	$31,37\%$
08	25	20	51	41	$19,61\%$
09	30	24	61	49	$19,67\%$
10	30	27	61	55	$9,84\%$

3.3 Mapping to Classical Planning

After pruning, if we use blind search, we could solve some of the 20 instances, but not all of them. This is because the state space is still large. Therefore, we present a new solution to our problem, which is mapping from Probabilistic Planning to Classical Planning Task.

We start this part by assuming that the horizon is an arbitrarily large number or infinite and there is no concurrency ($\sigma = 1$). To map the Probabilistic Planning Task to Classical Planning Task, there are three points that we have to consider:

1. We can see that all the rewards are non-positive, so instead of talking about non-positive rewards, we can talk about (non-negative) costs.

$$cost(o) = reward(o) \cdot (-1) \tag{3.5}$$

2. All the operators have a negative reward, except $noop^+$. If we can never apply $noop^+$, the expected reward would be $-\infty$ if $H = \infty$. The only way to get a

finite reward is to reach a state where *noop*$^+$ is applicable. Since the *noop*$^+$ operator is applicable only on the last course where we finish the program, it always makes sense to try to finish the program. Once we reach that point, we are sure that our policy is better. Therefore, an optimal policy aims to reach a state s where $complete \in S$.

3. In the third part, we have to map probabilistic operators, which have probabilistic effects, into classical ones.

   If passing a course is an action that optimal policy wants to do, then we apply the specific operator until we achieve the desired effect. Consequently, we can calculate how often we have to take a course in expectation to pass it. We can combine the operators of the task into a single operator by adapting the cost accordingly. The cost of an operator o would be the cost of taking the course for the first time and $\frac{1}{Prob_{c,C_P}} - 1$ multiplied by the cost of retaking the same course.

Let s, a be such that $\sum_{s'} T(s,a,s') = 1$. The mapping can happen only when $\sigma = 1$ because only in this case we have the property $T(s,a,s') = 1 - T(s,a,s)$ for all $s' \neq s$.

Given an Academic Advising instance A, we create a Classical Planning Task as follows:

- $o_{\{c\},C_P}$ for all $c \in C$ maps into

$$\langle \neg passed_c \wedge \exists_{c' \in C^*, c' \neq c} \neg passed_{c'}, \{passed_c\}, cost(o) + (\frac{1}{Prob_{c,C_P}} - 1) \cdot cost^+(o) \rangle$$

To better understand how this transformation occurs, let us take a simple example. Let A, C and D be courses, where D is the required course. The edges, same as above, tell us about the prerequisites for each course. This is illustrated in Figure 3-2.

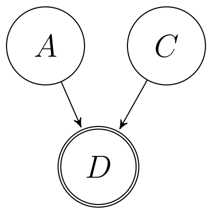

Figure 3-2: Academic Advising domain in probabilistic version

When this problem is converted to a classical domain, the graph is changed by adding new nodes and new edges, as is shown in Figure 3-3.

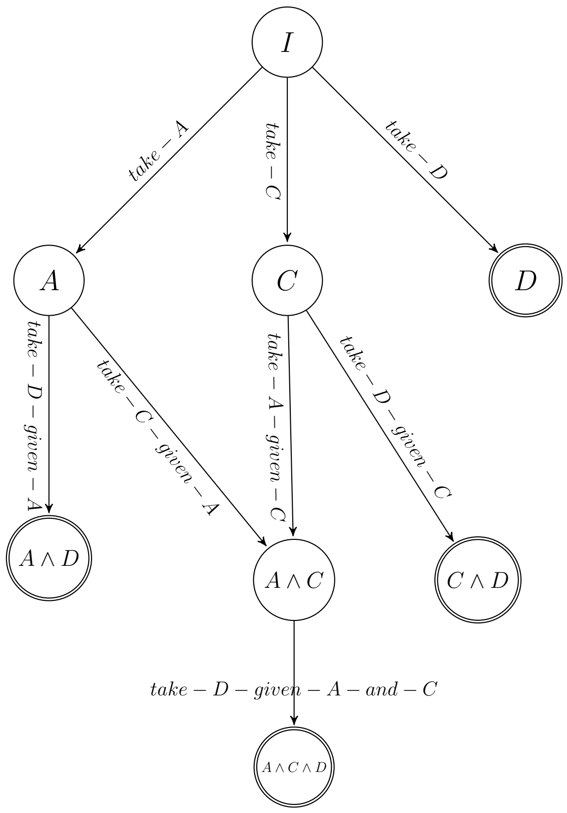

Figure 3-3: Academic Advising domain converted into a classical domain

Based on the above three points and on the subsequent discussion, we can state the following theorem:

Theorem 2. *For all Academic Advising instances A, where $\sigma = 1$ and $h = \infty$, and π, an optimal plan for the induced classical Planning Task $\mathcal{T}$, we have*

$$V_*(s_0, H) = -cost(\pi)$$

For instances where these two constraints are fulfilled, the reward of these instances corresponds to the true expected reward. There are instances where the concurrency can be larger than one. For the cases where $\sigma > 1$, we can no longer compile them into Classical Planning. However, we can still use it to compute the upper bound reward.

Theorem 3. *For all Academic Advising instances A, where $\sigma > 1$ and $h = \infty$, and π, an optimal plan for the induced Classical Planning Task $\mathcal{T}$, we have*

$$V_*(s_0, H) \geq -\frac{cost(\pi)}{\sigma}$$

Proof. (Proof sketch) Let us assume that we always perform as many actions as the concurrency and we take the courses where all the prerequisites are already passed. If for one course per semester we get a reward of x, then with two courses per semester, we expect a reward of $\frac{x}{2}$. This is an optimistic assumption because it assumes that we can perfectly distribute the course load into each semester (which is not always the case). On the other side, since we apply operators in parallel, there could be a case where we have to take a course and its prerequisites at the same time. Hence, the result can only be larger than the true reward, and therefore, we consider it as an admissible heuristic for the initial state. □

If $h \neq \infty$, things get harder and we no more have an admissible heuristic.

Theorem 4. *Given that the condition $h = \infty$ does not hold, our result is not optimal.*

Proof. (Proof by counterexample) Let $A = \langle C, C^*, P, cost, cost^+, h, \sigma, penalty, \mathbb{P}\rangle$ be an Academic Advising instance, where

- $C = \{A, B\} \wedge C^* = \{B\}$,
- $h = 3 \wedge \sigma = 1$,
- $cost = cost^+ = 1$,
- $penalty = -5$,
- $P(A) = \{\} \wedge P(B) = \{A\}$
- $\mathbb{P}(A) = 0,2 \wedge \mathbb{P}(B) = 0,8$

The horizon in this example is small, which means that the given constraint is violated. The graph for this instance is depicted below.

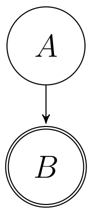

Figure 3-4: Academic Advising instance

Let us assume that the optimal path initially takes course A, but is unlucky in its first attempt and does not pass ($h = 2$). In the second attempt, it passed it ($h = 1$) and then; it takes and passes the course B($h = 0$). In this scenario, we reach the goal at the last step. However, it does not make sense to try to reach the goal if we reach it at the last step because we can never apply *noop*$^+$, and we can never benefit from this action. Thus, reaching the goal instead of just applying *noop* operator will cost us more, which is not what we want and it is not the optimal solution. $\square$

The reason we could ignore the horizon at the beginning is that it is irrelevant during the mapping process. We can consider it by just comparing the expected reward to the *noop* policy. Between our results and *noop*, we choose as our optimal policy the better one.

Let us take an example. Assume we have 10 courses to complete the program. We pass each of them with a probability of 1, and it costs us 1. The number of courses that we can take per semester is 1, the penalty is 5, and the reward is -6. To calculate the reward of *noop* policy, we multiply the penalty with the horizon. Our total reward would be -50, instead of -60 as the *noop* policy would give us because we benefit from the fact that we have completed the program. Taking *noop* would only make sense if we never finish the program and in this case, taking a new course would give us additional cost, which is worse than just playing *noop* (the case where $h = 13$ in the Table 3.3). We present the results for both policies on the table below:

Table 3.3: Comparison of 1-deterministic course and noop policy

Horizon	noop	1-deterministic course
10	-60	-50
11	-60	-55
12	-60	-60
13	-60	-65

3.4 Results

Considering our problem as deterministic makes it much simpler than solving it as a probabilistic problem. To solve a classical domain, we used Fast Downward planner[4]. To do so, we generated Planning Domain Definition Language (PDDL) files and searched using A* algorithm and landmark cut heuristic (LM-cut)[2] [5].

For cases where $\sigma = 1$ and $h = \infty$, the result gives us the true expected rewards. For cases where $\sigma > 1$ and $h = \infty$, the result gives us an admissible heuristic for the initial state. If $h \neq \infty$, we no longer have an admissible heuristic, but we assume that the result we get is a near-optimal solution. Here, simulating the execution would be a good solution. In Table 3.4 are shown the results for 20 instances.

Table 3.5 compares our results with other results of the planners who took part in IPC 2018.

Table 3.4: The results of the instances

Instance	Reward	Concurrency	Horizon
01	25	1	20
02	15	2	20
03	20	1	20
04	21.87	1	20
05	26.63	2	20
06	55	1	30
07	40.98	2	30
08	30.41	2	30
09	25	1	30
10	42	2	30
11	34.09	3	40
12	36.51	2	40
13	42.57	2	40
14	44.24	3	40
15	53.09	2	40
16	52.79	3	50
17	41.8	4	50
18	44.74	3	50
19	45.59	4	50
20	35.35	5	50

Table 3.5: Comparison of our results with other planners

Instance	A2CPlan	Conformant-SOGBOFA-B	Conformant-SOGBOFA-F	Imitation Net	Prost 2011	Prost 2014	Prost-DD-1	Prost-DD-2	Random Bandit	Our results
01	-64.0	-48.46	-48.4	-100.0	-53.66	-46.86	-47.13	-46.33	-82.4	-25
02	-76.53	-68.4	-63.13	-100.0	-57.33	-56.73	-49.93	-43.66	-100.0	-15
03	-100.0	-35.06	-35.2	-100.0	-46.13	-45.6	-37.8	-35.2	-100.0	-20
04	-72.72	-78.92	-79.18	-100.0	-69.01	-71.56	-39.48	-43.02	-76.46	-21.87
05	-133.45	-100.0	-100.0	-100.0	-100.0	-102.3	-90.12	-99.6	-100.0	-26.63
06	-150.0	-102.2	-82.86	-150.0	-136.8	-134.2	-83.46	-84.33	-150.0	-55
07	-198.89	-150.0	-150.0	-150.0	-150.0	-157.68	-188.96	-190.17	-150.0	-40.98
08	-229.45	-150.0	-150.0	-150.0	-150.0	-164.56	-182.84	-179.65	-150.0	-30.41
09	-150.0	-97.93	-66.53	-150.0	-145.2	-148.26	-86.33	-77.0	-150.0	-25
10	-200.2	-150.0	-150.0	-150.0	-200.37	-150.0	-200.24	-199.7		-42
11	-350.45	-200.0	-200.0	-200.0					-200.0	-34.09
12	-204.94	-200.0	-200.0	-200.0	-200.0	-204.09	-215.2	-213.38		-36.51
13	-230.04	-200.0	-200.0	-200.0	-200.0	-207.17	-282.48	-281.41		-42.57
14	-491.26	-200.0	-200.0	-200.0						-44.24
15	-240.46	-200.0	-200.0	-200.0					-200.0	-53.09
16	-467.93	-250.0	-250.0							-52.79
17		-250.0	-250.0							-41.8
18		-250.0	-250.0							-44.74
19		-250.0	-250.0							-45.59
20		-250.0	-250.0							-35.35

Chapter 4

Chromatic Dice

Yahtzee is a game where five dice are rolled with the opportunity to choose some dice and re-roll them up to two times. The displayed values could fit into a set of categories. The planner has to choose a category from this set for the given values of the dice. Each category has a corresponding reward. In the end, the gathered reward determines the planner's performance for the chosen category. Additionally, it determines if the planner will receive a bonus for its performance[9].

Chromatic Dice is an MDP variant of Yahtzee. However, here we use two-dimensional dice represented by values and colors. The values range from 1 to 6, and there are 5 different colors. The values and the colors of the dice are independent of each other, i.e. all combinations of values and colors are possible. Yahtzee is considered a special case of Chromatic Dice with a single colored dice. They are also different in the bonuses and the set of categories as described below. Depending on the instance, the number of categories varies.

The categories of the Chromatic Dice are divided into four sections. In each description of these categories below, we provide the abbreviations used to denote them.

The Upper value Section:

- **Ones**(C_1) scores one point for each of the dice that shows a value of one,
- **Twos**(C_2), **Threes**(C_3), **Fours**(C_4), **Fives**(C_5) **and Sixes**(C_6) are accordingly.

The middle color section:

- **Reds**(C_r) scores the total value of dice that have red color,
- **Greens**(C_g), **Blues**(C_b), **Yellows**(C_y) **and Purples**(C_p) are accordingly.

The Lower value Section:

- **Two pairs**(C_{2p}) scores the total value shown on all dice if there exist two set of different pairs, zero otherwise. Two dice are paired when they have the same value,
- **Three of a kind**(C_{3oak}) scores the total value shown on all dice if three of them have the same value, zero otherwise,
- **Four of a kind**(C_{4oak}) scores the total value shown on all dice if four of them have the same value, zero otherwise,
- **Full house**(C_{fh}) scores 25 points if three dice share the same value and the other two share same value,
- **Small straight**(C_{ss}) scores 30 points if four dice have consecutive values,
- **Large straight**(C_{ls}) scores 40 points if five dice have consecutive values,
- **Chance**(C_c) scores the total value of all dice with no restriction and
- **Five of a kind**(C_{5oak}) scores 50 points if all dice show the same value.

The Lower color Section:

- **Three of a color**(C_{3oac}) scores the total value shown on all dice if three of the dice have the same color, zero otherwise,
- **Four of a color**(C_{4oac}) scores the total value shown on all dice if four of the dice have the same color, zero otherwise,
- **Color full house**(C_{cfh}) scores 20 points if three of the dice show one color and the other two show one color,

- **Flush(C_f)** scores the total value shown on all dice if all dice show the same color,
- **Rainbow(C_{rb})** scores 35 points if all the dice have different colors.

Besides these rules, there are bonus points that are awarded in certain circumstances. In Yahtzee, the bonus is based on the points we gather in the upper section, specifically, if we over than 63 points. One way to get this number is if, for each category in the upper section, at least three dice show the correct value. For instance, in category C_1, after the dice are rolled, we have at least three of them showing the value of one. The same is true for other categories (C_2, C_3, C_4, C_5 and C_6).
In Chromatic Dice domain, the bonuses are described differently. It provides two bonuses: one for the upper value section and the other for the middle color section. For both bonuses we have 6 different levels $\mathscr{L} = \{l_1, l_2, l_3, l_4, l_5, l_6\}$, depending on what is scored in each round of the upper and middle categories. Each level has a probabilistic component; so there is a stochastic chance that we receive the bonus. Initially, we are on the fourth level for both bonuses. We remain at the same level if, for each of the upper and middle section categories, we score with three dice showing the correct values. If less than three dice show the correct value, the bonus level is decreased, and if more than three dice show the correct value, the bonus level is increased. In the end, based on the level we are on, we get the bonus with a specific probability.

4.1 Chromatic Dice Domain

Let $\mathscr{C} = \{C_1,\ C_2,\ C_3,\ C_4,\ C_5,\ C_6,\ C_r,\ C_b,\ C_g,\ C_y,\ C_p,\ C_{4oak},\ C_{3oak},\ C_{5oak},\ C_c,\ C_{ss},\ C_{ls},\ C_{2p},\ C_{fh},\ C_{3oac},\ C_{4oac},\ C_f,\ C_{cfh},\ C_{rb}\}$ be the set of all possible categories, $l_i \in L$ be a specific level, $\mathscr{V} = \{1, 2, 3, 4, 5, 6\}$, $\mathscr{R} = \{green, red, blue, yellow, purple\}$ and $\mathscr{D} = \{d_1, d_2, ..., d_k\}$.

Definition 4.1.1. *A Chromatic Dice instance is a 9-tuple* $W = \langle C,\ P_v,\ P_c,\ B_v,\ B_c,\ P_{B_v},\ P_{B_c}\rangle$, *where*

- $C \subseteq \mathscr{C}$ *is a set of Categories,*

- P_v *is a probability distribution over* $\mathscr{V}$ *s.t.* $P_v(v) > 0, \forall v \in \mathscr{V}$*,*

- P_c *is a probability distribution over* $\mathscr{R}$*,*

- $B_v : \mathscr{L} \rightarrow \mathbb{N}_0$ *is the value bonus,*

- $B_c : \mathscr{L} \rightarrow \mathbb{N}_0$ *is the color bonus,*

- $P_{B_v} : l_i \rightarrow [0,1]$ *is the probability of winning the value bonus for level* $l_i \in \{l_1, l_2, l_3, l_4, l_5, l_6\}$ *and*

- $P_{B_c} : l_i \rightarrow [0,1]$*, where* $l_i \in \{l_1, l_2, l_3, l_4, l_5, l_6\}$*, is the probability of winning color bonus.*

Let $\mathscr{R}_w$ be the set of all colors of the instance, $\mathscr{R}_w = \{c \in \mathscr{R} | P_c(c) > 0\}$.

A Chromatic Dice instance $W = \langle C, P_v, P_c, B_v, B_c, P_{B_v}, P_{B_c} \rangle$ induces a Probabilistic Planning Task $\Pi = \langle V, I, O, H \rangle$, where

- $\mathbf{V} = \{taken_c | c \in C\} \cup \bigcup_{n \in \{1,2,3,4\}} \{phase_n\} \cup \bigcup_{v_i \in \mathscr{V}} \bigcup_{c_j \in \mathscr{R}_w} \bigcup_{d_k \in \mathscr{D}} \{d_k\_is\_v_i\_c_j\} \cup \{receive\_value\_bonus\} \cup \{receive\_color\_bonus\}$

- $\mathbf{I} = \{\neg taken_c | c \in C\} \cup \{phase_1\} \bigcup_{n \in \{2,3,4\}} \{\neg phase_n\} \cup \{\neg receive\_color\_bonus, \neg receive\_value\_bonus\} \bigcup_{v_i \in \mathscr{V}} \bigcup_{c_j \in \mathscr{R}_w} \bigcup_{d_k \in \mathscr{D}} \{\neg d_k\_is\_v_i\_c_j\}$

- $\mathbf{H} = 4 \cdot |C| + 2$

- $\mathbf{O} = \{roll^1\} \cup \{roll^n_{\mathscr{D}'} | n \in \{2,3\}, \mathscr{D}' \in 2^{\mathscr{D}}\} \cup \{assign^c_{\{d_1\_is\_v_i\_c_j, \ldots, d_5\_is\_v_i\_c_j\}} | \forall c \in C,\ v_i \in \mathscr{V}$ and $c_j \in \mathscr{R}_w\} \cup \{receive\_bonus_k | k \in \{0,1,2,3\}\}$

We have divided the operators into three main types: *roll*, *assign* and *receive_bonus*. The *roll* operators are applicable only on roll phases where the dice are rolled(denoted below as $phase_1$, $phase_2$ and $phase_3$). The reward in each of the *roll* operators is 0. The *assign* operators are applicable for the assigning phase of the domain where a category is assigned(denoted as $phase_4$). The reward of the *assign* operator is equal to the score that is received when the shown dice is assigned into a category. After each three roll operators, an assign operator is always applied as long as there is an open category. If all categories are taken, then the *receive_bonus* operator is applied. By applying this operator, depending on the bonus levels we are on, it is decided if we receive the bonus or not. The reward of this operator could be 0 ($receive\_bonus_0$), the score of the value bonus ($receive\_bonus_1$), the score of the color bonus ($receive\_bonus_2$) or the score of both bonuses ($receive\_bonus_3$). Each round of the domain consists of three *roll* operators and an *assign* operator except for the last round which contains the *receive_bonus* operator as well. Initially, we start the round by applying the $roll^1$ operator.

Below, we give the precondition, the effect and the reward for each of the operators.

- $roll^1$:

$$\mathbf{pre}(roll^1) = phase_1$$

$$\mathbf{eff}(roll^1) = \bigcup_{\langle v_1,..,v_5\rangle \in \mathscr{V}^5} \bigcup_{\langle c_1,...c_5\rangle \in \mathscr{R}_w^5} \Big\langle \bigcup_{k \in \{1,...,5\}} \Big\{d_k\_is\_v_i\_c_j\Big\} \cup \{\neg phase_1 \cup phase_2\}, \prod_{i=1}^{k} P_v(v_i) \cdot \prod_{i=1}^{k} P_c(c_i)\Big\rangle$$

$$\mathbf{reward}(roll^1_{\mathscr{D}_5}) = 0$$

- $roll^n_{\mathscr{D}'}$ for $n \in \{2,3\}$ and $\mathscr{D}' \subseteq 2^{\mathscr{D}}$:

$$\mathbf{pre}(roll^n_{\mathscr{D}'}) = phase_n$$

$$\mathbf{eff}(roll^n_{\mathscr{D}'}) = \bigcup_{\langle v_1,..,v_{|\mathscr{D}'|}\rangle \in \mathscr{V}^{|\mathscr{D}'|}} \bigcup_{\langle c_1,...c_{|\mathscr{D}'|}\rangle \in \mathscr{R}_w^{|\mathscr{D}'|}} \Big\langle \bigcup_{k \in \{1,...,|\mathscr{D}|\}} \Big\{d_k\_is\_v_i\_c_j\Big\} \cup \{\neg phase_n \cup phase_{n+1}\}, \prod_{i=1}^{k} P_v(v_i) \cdot \prod_{i=1}^{k} P_c(c_i)\Big\rangle$$

$$\mathbf{reward}(roll^n_{\mathscr{D}'}) = 0$$

- $assign^{c}_{\{d_1\_is\_v_i\_c_j,...,d_5\_is\_v_i\_c_j\}} \forall v_i \in \mathscr{V} \wedge c_j \in \mathscr{R}_w$:

$$\mathbf{pre}(assign^{c}_{\{d_1\_is\_v_i\_c_j,...,d_5\_is\_v_i\_c_j\}}) = phase_4 \wedge \neg taken_c \wedge \bigwedge_{\langle v_1,..,v_k\rangle \in \mathscr{V}^k} \bigwedge_{\langle c_1,...c_k\rangle \in \mathscr{R}_w^k} \bigwedge_{i \in \{1,...,5\}} \{d_i\_is\_v_i\_c_i\}$$

$$\mathbf{eff}(assign^{c}_{\{d_1\_is\_v_i\_c_j,...,d_5\_is\_v_i\_c_j\}}) = taken_c \cup \neg phase_4 \cup phase_1 \cup \langle receive\_value\_bonus, P_{B_v}(l)\rangle \cup \langle receive\_color\_bonus, P_{B_c}(l)\rangle$$

$$\mathbf{reward}(assign^{c}_{\{d_1\_is\_v_i\_c_j,...,d_5\_is\_v_i\_c_j\}}) = score^{c}_{\{d_1\_is\_v_i\_c_j,...,d_5\_is\_v_i\_c_j\}}$$

The score function calculates the score for a category c when dice $d_1\_is\_v_i\_c_j$, ..., $d_5\_is\_v_i\_c_j$ are rolled. The way this calculation is carried out for each category is given (informally) in the Chromatic Dice description at the beginning of the chapter. For example:

For $c = C_1$:

$$score^{C_1}_{\{d_1\_is\_v_i\_c_j,...,d_5\_is\_v_i\_c_j\}} = \sum_{\langle c_1,...c_k\rangle \in \mathscr{R}_w^k \wedge k \in \{1,...,5\}} [d_k\_is\_1\_c_j] \cdot 1$$

- $receive\_bonus_0$:

$$\mathbf{pre}(receive\_bonus_0) = \{taken_c | c \in C\}$$
$$\mathbf{reward}(receive\_bonus_0) = 0$$

- $receive\_bonus_1$:

$$\mathbf{pre}(receive\_bonus_1) = \{taken_c | c \in C\} \cup receive\_value\_bonus$$
$$\mathbf{reward}(receive\_bonus_1) = B_v$$

- $receive\_bonus_2$:

$$\textbf{pre}(receive\_bonus_2) = \{taken_c | c \in C\} \cup receive\_color\_bonus$$
$$\textbf{reward}(receive\_bonus_2) = B_c$$

- $receive\_bonus_3$:

$$\textbf{pre}(receive\_bonus_3) = \{taken_c | c \in C\} \cup receive\_value\_bonus \cup$$
$$receive\_color\_bonus$$
$$\textbf{reward}(receive\_bonus_3) = B_v + B_c$$

Throughout this chapter, we will explain the complexity of the Chromatic Dice domain while comparing it with the Yahtzee domain [3]. In this paper, the author has described a graph-theoretic and combinatoric technique which are used to compute an optimal strategy for Yahtzee. In the subsequent section, we will look into the Chromatic dice structure compared to Yahtzee.

4.2 Chromatic Dice Structure

Illustrated in Figure 4-1 is the general architecture of the Chromatic Dice domain. It mainly consists of multiple rounds limited by a horizon. Each round contains two parts: a *Micro-step* and a *Macro-step*. A *Micro-step* consists of three roll phases (denoted as $phase_1$, $phase_2$ and $phase_3$) where the dice are rolled and re-rolled. A *Macro-step* consists of an assign phase (denoted as $phase_4$), where based on the values of the dice, a category is assigned. The *Macro-step* is always performed after the completion of the *Micro-step*. The table in the corner maintains the information of the categories assigned in previous rounds and the information on bonuses.

A state consists of all information of levels of both bonuses and the assigned categories as depicted in the table of Figure 4-1, and all information of the dice. As long as we perform *Micro-steps*, the categories and the levels of the bonuses do not change, and as long as we perform *Macro-steps*, the dice do not change.

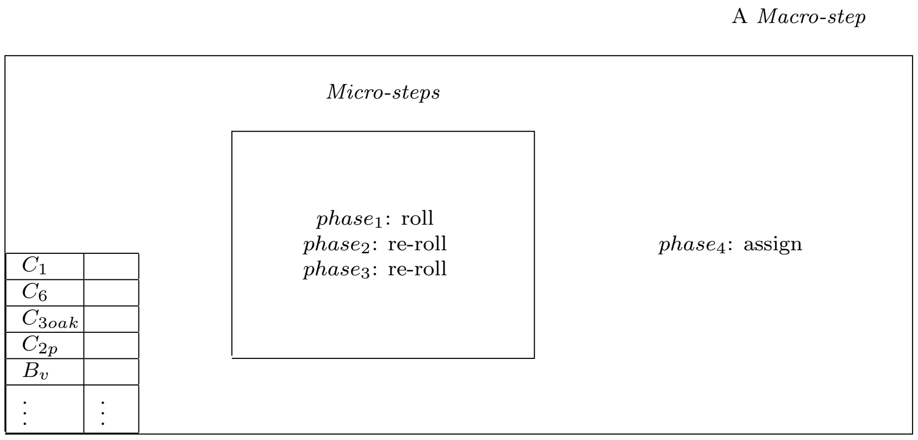

Figure 4-1: Chromatic Dice architecture

Figure 4-2 and Figure 4-3 show the relationship between the *Micro-steps* and the *Macro-steps*. Figure 4-2 represents combinations and connections between the states during the *Micro-step* process. The application of the *roll* operator (roll or re-roll) is denoted by an edge. The probability of reaching the state from the initial state by applying the *roll*[1] operator is denoted with p equal to $\frac{1}{\binom{n+k-1}{k}}$, where n is the number of combinations of the dice ($values \cdot colors$) and k is the number of dice. By applying the *roll* operators, we change the dice, but the assigned categories and levels of bonuses remain unaffected.

Figure 4-3 represents the *Macro-steps* and the connections between them. The whole *Micro-step* shown in Figure 4-2 is denoted by a rectangle node in Figure 4-3, which for simplicity, contains only the last rolled results. The application of the *assign* operator is denoted by an edge. By applying the *assign* operator, the rolled dice always remains the same, while the assigned category and the levels of the bonuses change. The result of a *Micro-step* is an input for a *Macro-step* and vice versa.

In the following sub-sections, we explain the *Macro-step* and *Micro-step* in more detail. In the *Macro-step* sub-section, first, we described the naive state space representation and how we can represent the states more compactly. We can simplify the state space by looking into the *Macro-step* and *Micro-step* separately. By using the compact representation, we reduce the total number of Macro-steps. Afterwards, we

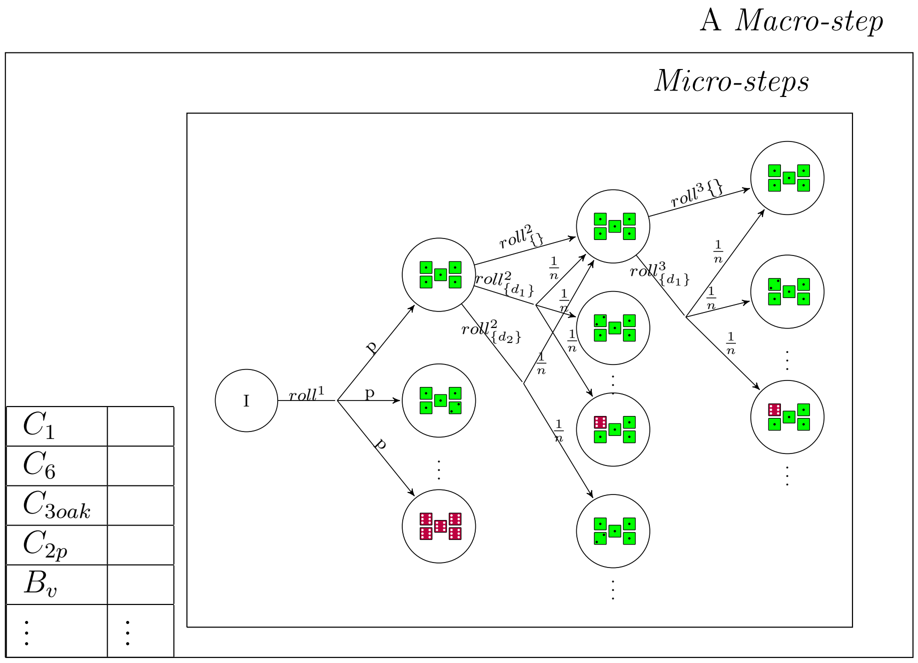

Figure 4-2: *Micro-step*

describe the *Macro-step* structure and how they are interconnected. In the *Micro-step* sub-section, we first describe, in details, the *Micro-step* structure. Then, we explain briefly how we decrease the branching factor by shrinking the edges within a *Micro-step* instead of just using the naive strategy. By shrinking the edges, we shrink the applicable operators which help us to improve the performance of the optimal solution.

4.2.1 Macro-step

In a very naive implementation, we could represent states by considering all information depicted in the scorecard. In a below table we compare the number of possibilities to fill out the scorecard in Chromatic Dice and Yahtzee (in reference to the James Glenn paper[3]). There are seven possibilities for each of the upper section categories: the category is empty, or it contains a result that corresponds to zero, one, two, three, four or five dice of the right value. In the lower section, there are 28 possibilities for three of a kind, four of a kind and chance that include: zero, unused or a number of points between 5 to 30; and there are three possibilities for each of

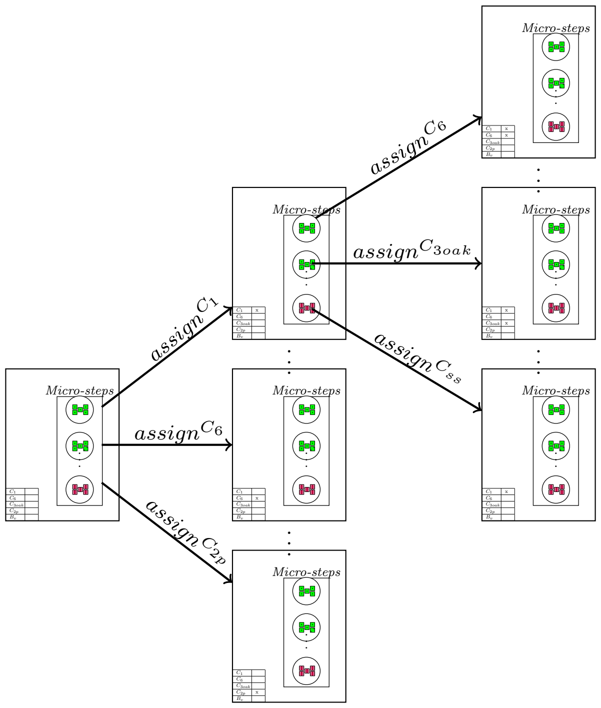

Figure 4-3: *Macro-steps*

the other categories that include: zero, unused or the constant score of the category. In total, there are $7^6 \cdot 28^3 \cdot 3^4 = 209,193,098,688$ different possibilities to fill out the scorecard. This is enormous and expensive in memory and time. Chromatic Dice has an even more complex and larger state space than Yahtzee. It has even more possibilities because of the increased number of categories (taking into consideration instances where $C = \mathscr{C}$). In Yahtzee, there are approximately 10^{11}*Macro-steps*, while in Chromatic Dice there are approximately 10^{22} (100 billion times larger than the number of *Macro-steps* in Yahtzee). In Table 4.1 we represent the differences.

However, we can have a smaller state space and still act optimally because some information is irrelevant for the computation of an optimal policy. The best compact way of representing the states is to keep track of which category has already been taken on previous rounds. This is important because Chromatic Dice depends on the history of the previous rounds.

The game aims to increase the total number of scored points by choosing the most suitable category in each round. Because of that, what would also affect the result of the game are the bonus points. For that reason, the optimal strategy is also sensitive

Table 4.1: Comparison of all possibilities between Yahtzee and Chromatic Dice

Section	Categories	Possibilities of Yahtzee	Possibilities of Chromatic Dice
Upper value section	C_1, C_2, C_3, C_4, C_5, C_6	7^6	7^6
Upper color section	C_r, C_b, C_g, C_y, C_p	-	7^5
Lower value section	C_{3oak}, C_{4oak}, C_c, C_{2p} [1]	28^3	28^4
	C_{5oak}, C_{ss}, C_{ls}, C_{fh}	3^4	3^4
Lower color section	C_{3oac}, C_{4oac}, C_f	-	28^3
	C_{fch}, C_{rb}	-	3^2
Total		$\approx 10^{11}$	$\approx 10^{22}$

to the bonus levels of values and colors.

So, the optimal strategy depends on two key information of the states: 1) which category is still available (is the category taken or not) and 2) which is the bonus level of the upper and middle section. There are $2^{|C|}$ possibilities for the former and 36 for latter. In total, we have $2^{|C|} \cdot 36$ *Macro-steps*, which are represented as boxes in Figure 4-3. For harder instances of Chromatic Dice, where $C = \mathscr{C}$, we have $2^{24} \cdot 36 \approx 2^{29}$ of these boxes. In Yahtzee, there are 13 categories and 64 possibilities for the upper bonus. Hence, there are 2^{19} *Macro-steps*. Hence, we can see that Chromatic Dice is 1 thousand times larger. This strategy is adopted for the implementation described in section 4.3

A *Macro-step* is a process where we map each possible combination of dice faces (the *Micro-step* results) to an open category. The decision of assigning into a specific category comes as a result of the total score. In each round, we choose the category that maximizes the total score of the domain. For a *Macro-step*, it does not matter how we get the input. An optimal strategy for the *Macro-step* can be found separately from the *Micro-step*. We can compute the perfect assignment without having to think about the dice rolling.

In Figure 4-4 is illustrated the last *Macro-step* of the Chromatic Dice domain,

[1]The category C_{2p} belongs only to the Chromatic Dice domain.

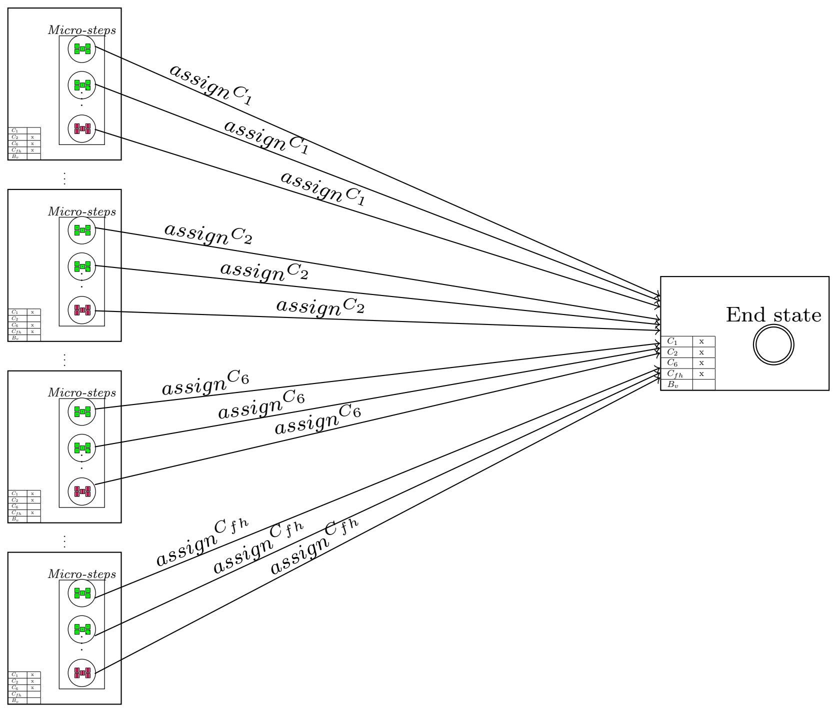

Figure 4-4: A fragment of a single *Macro-step*.

where we are left with only one open category. For simplicity, we will consider only a fragment of it. By applying an *assign* operator, a specific category is taken, and the bonus level is updated for the next round.

4.2.2 Micro-step

Micro-step is the process where we take the steps inside one of the boxes of Figure 4-3. An input of a *Micro-step* is a *Macro-step* result.

Shrinking the Micro-step State Space Naively, the total number of states in a single *Micro-step* is n^k, where n is the number of combinations of the dice($values \cdot colors$) and k is the number of dice. For Yahtzee(n=6) there are $6^5 = 7776$ states. In a Chromatic Dice instance, where $|\mathscr{R}| = 5$, there are $n = 30$ combinations of dice, and there are $30^5 = 24300000$ states.

However, it does not matter how the dice values are distributed among the dice within a roll. As long as the shown values of rolls are the same, although, in a different order, we consider those rolls equivalent. Due to this property, the number of states that we need to distinguish is lower than in the naive state representation as described and given in the next paragraphs.

The *Micro-step* is divided into two parts: The first part represents the outcome after each roll and the second part represents the choice of the agent for which dice to keep. Rolling and re-rolling the dice happens three times in a single round (first part) while the process of choosing which dice to keep happens twice in a single round (second part). After every roll there are $\binom{n+k-1}{k}$ possible outcomes, out of which an agent has $\sum_{k=0}^{5}\binom{n+k-1}{k}$ possibilities for which dice to keep; where n is the number of combinations of the dice ($values \cdot colors$), and k is the number of dice.

Since we have instances of different complexity, n varies between 6 and 30. For instances where $|\mathscr{R}|= 1$ (n=6), the total number of the states within a *Micro-step* is equivalent to Yahtzee. In Yahtzee, there are $\binom{10}{5} = 252$ possible outcomes when rolling 5 dice with 6 sides, and there are $\sum_{k=0}^{5}\binom{n+k-1}{k} = 462$ choices of which dice to keep. In total, there are $1 + 3 \cdot 252 + 2 \cdot 462 = 1681$ states (including the initial state). For complex instances of Chromatic Dice, where$|\mathscr{R}|= 5$ (n=30), we have $\binom{34}{5} = 278256$ possible outcomes when rolling 5 dice with 30 sides and $\sum_{k=0}^{5}\binom{n+k-1}{k} = 324632$ ways of choosing which dice to keep. Therefore, there are $1+3\cdot 278256+2\cdot 324532 = 1484033$ states in total for each *Micro-step*.

Shrinking the Micro-step edges Edges have a key role in finding the optimal path due to the impact on the expected values of the state. In the second part of the *Micro-step*, each state has at most than $2^5 = 32$ edges (5 dices with two possible decisions; keep or re-roll). However, many states have a lower number of edges. Let us explain this with a concrete example. Let us have a result $(4, red)(4, red)(4, red)(5, blue)(5, blue)$ on the first roll. The decision on how many dice to keep and how many dice to re-roll is divided into two groups, the group of $(4, red)$ and $(5, blue)$. Three choices can be made on $(5, blue)$ (keep none, keep one dice or keep both). Similarly, for $(4, red)$, we have four different choices. In this case, we have 12 different possibilities of edges. The first table shows the occurrences and outcomes for all patterns, while Table 4.2 shows the number of occurrences and the number of different choices of which dice to keep for all patterns. So, the total number of edges in the first process is $\sum_{i=0}^{6} Occurrence_i \cdot Keep_i$ for both two steps. The same calculation is done for the second part based on the Table 4.3 values, and we have the same results.

Table 4.2: Number of occurrences and possibilities of choosing which dice to keep, with respect to dice faces n, and k number of dice.

	Pattern	Occurrences	Keeps
1	abcde	$\binom{n}{k}$	32
2	aabcd	$\binom{n}{1}\binom{n-1}{3}$	24
3	aabbc	$\binom{n}{2}\binom{n-2}{1}$	18
4	aaabc	$\binom{n}{1}\binom{n-1}{2}$	16
5	aaabb	$\binom{n}{1}\binom{n-1}{1}$	12
6	aaaab	$\binom{n}{1}\binom{n-1}{1}$	10
7	aaaaa	$\binom{n}{1}$	6

Table 4.3: Number of occurrences and outcomes while re-rolling the dice, with respect to dice faces n, and k number of dice.

Dice kept	Occurrences	Keeps
0	$\binom{n+k-6}{0}$	$\binom{n+k-1}{5}$
1	$\binom{n+k-5}{1}$	$\binom{n+k-2}{4}$
2	$\binom{n+k-4}{2}$	$\binom{n+k-3}{3}$
3	$\binom{n+k-3}{3}$	$\binom{n+k-4}{2}$
4	$\binom{n+k-2}{4}$	$\binom{n+k-5}{1}$
5	$\binom{n+k-1}{5}$	$\binom{n+k-6}{0}$

The number of combinations (n) of the first five instances of Chromatic Dice is equivalent to Yahtzee. Therefore, Yahtzee is included in the table (for $n = 6$). For these cases, we have the same results as in the tables given by James Glenn in his paper[3]. Let us compare Yahtzee and the hardest instance of Chromatic Dice ($n = 30$), so we can see the difference between them in the number of edges. In Yahtzee, for the first part (which contains re-roll actions), referring to the formula,

$\sum_{i=0}^{6} Occurrence_i \cdot Keep_i$, we have 4368 edges. For the second part (dice kept after roll or re-roll), we have 4368 edges. While in other Chromatic Dice instances (where n=30), we have 7624512 for the first part and 7624512 for the second. The edges of the first and the second part are the same because edges in are a mirror of the edges out.

For each entry state of the *Micro-step*, we have $\binom{n+k-1}{k}$ edges, and on each state of the last phase of a round, we have at most edges as the number of categories. In Yahtzee, there are 252 edges for the entry state and 13 edges for each category in the assign phase, so there are at most $252+4\cdot4368+13\cdot252 = 21000$ edges. In total, there are 2^{19} *Micro-steps*, so we have around 11 billion edges. On the other hand, in the hardest Chromatic Dice instance, there are 278256 entry edges and 24 edges for each category in the assign phase, so for a single *Micro-step*, we have 37454448 edges. In total, for 2^{29} *Micro-steps*, we have around 200 trillion edges. Compared to the naive representation of the states, the compact version has a smaller number of edges, even though the number is still large and finding an optimal solution is challenging.

4.3 Implementation strategies

In this master thesis, the implementation of the Chromatic Dice is done using two approaches. First, we implement the Chromatic Dice optimal solution. After that, we implement the heuristic method to find the near-optimal solution. We do this because the state space of the domain is enormous and for most instances, finding an optimal solution is inconceivable. Hence, we need a near-optimal solution to compute it with a computer.

To derive an optimal policy for this domain, we use the backtracking method by starting from the last round and calculating backwards the expected values for all states. We choose this method so we can find an optimal solution of two parts (*Micro-step* and *Macro-step*) independent of each other.

Backtracking method We start at the point where all categories are taken. We consider all level combinations of both bonuses and add the rewards of the bonuses to each of the corresponding states. Then we go back for one step and calculate the *Macro-step* of the round, where we consider all possibilities with one free category. We calculate the expected value by maximizing the score over the free categories plus the expected value of the next state (in this case, the corresponding rewards of bonuses). The optimal solution of a *Macro-step* can be found in isolation from the *Micro-step*. The next step is the last re-roll in the *Micro-step*. Here, the optimal solution of the *Micro-step* can also be found in isolation from *Macro-step* because we already have the results of the *Macro-step*. Therefore, the expected value of ending up with the specific roll is given by the weighted average of the expectation of the states that we could be in. In the same way, we go backward to the initial state.

4.4 Results of the optimal strategy

Using the optimal strategy, we could solve instances that had a low number of colors or a low number of free categories, which make the instances less complex. Therefore, finding an optimal solution was possible within a reasonable time. Below is the table with the optimal results.

Table 4.4: Optimal solutions

Instances	|Macro-steps|	Expected Score
01	$2^6 \cdot 36 = 2304$	72.51
02	$2^{10} \cdot 36 = 36864$	160.03
03	$2^{11} \cdot 36 = 73728$	216.88
04	$2^{14} \cdot 36 = 589824$	279.42
05	$2^8 \cdot 36 = 9216$	154.40
08	$2^9 \cdot 36 = 18432$	147.17
11	$2^{10} \cdot 36 = 36864$	155.48

4.5 Near-optimal strategies

Since the optimal solution has a large number of states, finding an optimal solution is possible only in the instances where the number of free categories is small. Therefore, for other harder instances, we came up with a near-optimal solution. We introduce a heuristic strategy which follows the optimal solution strategy.

In this strategy, we divide the categories into any number of disjoint groups. In this way, we consider an instance as n independent sub-instances, and we execute them separately. Here, we also have disjoint actions of the MDP in a way that all the actions responsible for the reward are only relevant to one of the sub-instances.

This idea is similar to the cost partitioning in classical planning, where it has been shown that it is possible to sum up the estimates of heuristics admissibly under some constraints [6]. Similar result can be obtained for probabilistic planning.

Definition 4.5.1. *(Reward partitioning) Let Π be a Probabilistic Planning Task with operators O.*
A reward partitioning *for Π is a tuple $\langle reward_1, ..., reward_n \rangle$ where*

- *$reward_i : O \to \mathbb{R}_0^+$ for $1 \leq i \leq n$ and*
- *$\sum_{i=1}^{n} reward_i(o) \geq reward(o)$ for all $o \in O$*

The reward partitioning *induces a tuple $\langle \Pi_1, ..., \Pi_n \rangle$ of probabilistic planning tasks, where each Π_i is identical to Π except that the reward of each operator o is $reward_i(o)$.*

Theorem 5. *Let Π be a probabilistic planning task, $\langle reward_1, ..., reward_n \rangle$ be a reward partitioning and $\langle \Pi_1, ..., \Pi_n \rangle$ be a tuple of induced tasks.*
Then the sum of the solution rewards of the induced tasks is an admissible expected reward, i.e., $\sum_{i=1}^{n} V_{\Pi_i}^{\pi_i^*} \geq V_{\Pi}^*$

Proof. Since in the FH-settings we do not have cycles, in equation system we can replace every $V^*(s)$ of states $s \neq s_0$ with rewards. Consequently, $V^*(s_0)$ depends only on the weighted rewards.

Let $O = \{o_1, ..., o_m\}$. Then

$$\begin{aligned}
\sum_{i=1}^{n} V_{\Pi_i}^{\pi_i^*}(s_0) &\geq \sum_{i=1}^{n} V_{\Pi_i}^{*}(s_0) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{m} \omega_j \cdot reward_i(o_j) \\
&= \sum_{j=1}^{m} \omega_j \sum_{i=1}^{n} reward_i(o_j) \\
&\geq \sum_{j=1}^{m} \omega_j \cdot reward(o_j) \\
&= V_{\Pi}^{*}(s_0)
\end{aligned}$$

□

The Definition 4.5.1 holds for all FH-planning tasks. In our strategy, we propose to use the zero-one reward partitioning. We partition the categories among the sub-instances in such a way that each category yield the reward in only one of the sub-instances, while in all others the reward is 0.

Although we divide the open categories mutual exclusively into sub-instances, the horizon is still the same as in the original instance, and the size of the MDP also remains the same. Hence, the admissible computation is almost as hard as the optimal solution since the resulting of the sub-instances are still too large and hard to compute. Therefore, we compute a near-optimal policy which does not guarantee the upper bound solution, but we assume that it is a near-optimal strategy. This strategy shrinks the horizon as much as possible by throwing out all the categories that do not yield the reward. In this case, we have smaller sub-instances, and the computation is possible. Executing the resulting policy would give us a lower bound on the rewards.

4.6 Results of near-optimal strategy

All instances that we could not solve optimally are solved near-optimally using the heuristic strategy. We divided the instances into 2 or 3 sub-instances depending on their complexity; one sub-instance that contains all lower section categories, and one or two sub-instances that contain the upper and middle section categories. Table 4.5 shows the results.

Table 4.5: Heuristic results

Instances	|Sub-instances|	Expected Score
06	3	389 .95
07	3	496.29
09	3	395.49
10	3	489 .76
12	3	480.70
13	3	225.70
14	3	297.05
15	3	500.50
16	2	406.43
17	2	409.32
18	2	381.33
19	2	401.63
20	2	430.44

Table 4.6 compares our results with other results of the planners who took part in IPC 2018.

Table 4.6: Comparison of our results with other planners

Instance	A2CPlan	Conformant-SOGBOFA-B	Conformant-SOGBOFA-F	Imitation Net	Prost 2011	Prost 2014	Prost-DD-1	Prost-DD-2	Random Bandit	Our results
01	18.05	48.89	47.04	26.04	47.91	40.21	37.87	37.31	50.8	72.51
02	39.68	182.17	180.4	50.44	74.05	83.51	71.69	77.29	110.16	160.03
03	43.2	142.21	138.73	61.49	151.41	135.48	108.61	107.27	173.0	216.88
04	55.44	247.52	243.41	65.65	148.77	157.12	119.45	123.48	216.17	279.42
05	49.03	118.31	114.35	87.2	118.24	104.61	108.0	107.19	119.68	154.40
06	83.04	325.84	316.08	119.19	205.56	201.93	150.47	164.71	289.05	389.95
07	114.8	402.53	402.77	185.6	264.88	286.71	203.36	209.24	377.92	496.29
08	37.39	120.29	109.19	81.71	124.0	84.55	94.53	93.08	121.35	147.17
09	67.44	313.79	311.31	119.35	198.28	196.28	144.49	147.57	302.56	395.49
10	85.05	370.13	370.19	156.64	258.31	268.89	193.27	180.51	365.93	489.76
11	36.0	108.43	105.39	74.59	127.61	74.77	86.93	81.85	111.88	155.48
12	75.48	355.01	348.89		241.91	248.15	162.13	176.13	352.84	480.70
13	33.79	115.63	113.07	67.16	114.56	72.25	86.2	82.8	113.84	225.70
14	41.87	204.92	194.55	86.21	136.97	90.91	74.63	81.89	161.45	297.05
15	73.41	402.25	394.88		242.25	258.99	159.48	165.15	335.13	500.50
16	162.36	441.05	436.64		305.99	277.73	227.68	240.96	369.96	406.43
17	196.8	414.27	412.69		375.43	266.48	236.28	234.41	403.08	409.32
18	140.51	450.97	446.12	167.39	258.03	267.15	211.73	213.53	345.79	381.33
19	164.4	423.39	426.31		256.87	259.44	205.24	213.24	345.77	401.63
20	184.07	452.44	451.01		259.47	265.64	208.96	206.96	345.81	430.44

Chapter 5

Conclusion and Future work

In this thesis, we introduced a (near)-optimal solution for two probabilistic planning domains, Academic Advising and Chromatic Dice. Both of them are introduced in IPC 2018. To solve them we implemented a domain-independent solver for both domains.

In the Academic Advising domain, the first step to finding an optimal solution is simplifying the problem by removing irrelevant actions and state variables from the planning task. We used relevance analysis to preserve only relevant information. However, it is prohibitively expensive to compute an optimal solution using any search algorithm by only using this analysis. Therefore, we presented a new solution to our problem, which is mapping the probabilistic domain into a classical domain and getting the optimal expected values. However, this transition is possible only when the constraints $\sigma = 1$ and $h = \infty$ are satisfied. Using this approach, for $\sigma > 1$ and $h = \infty$, we lead to an admissible heuristic. When $h \neq \infty$ we assume that our results are near-optimal.

In the Chromatic Dice domain, we presented and implemented an optimal strategy. However, not all instances are solvable using this strategy because they have large and complex state space. Therefore, we presented a heuristic strategy to compute the harder instances efficiently. We presented the idea of dividing the instances into sub-instances using zero-one reward partitioning and summing them admissibly. Due to the horizon, this approach is almost as hard as the optimal strategy because the size of the MDP is the same in both approaches. Hence, we divide the instances into sub-instances where we shrink the horizon as much as possible and compute them efficiently. To use this heuristic strategy, we have to give up the admissibility.

In both domains, we described the methods of near-optimal solutions, but they do not give us any admissibility guarantees. For future work, we plan to simulate these policies and discern the lower bounds for both domains.

Bibliography

[1] Florian Geißer, David Speck, and Thomas Keller. An Analysis of the Probabilistic Track of the IPC 2018. In *ICAPS-2019 Workshop on the International Planning Competition (WIPC 2019)*, pages 27–35, 2019.

[2] Malik Ghallab, Craig Knoblock, David Wilkins, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David Smith, Ying Sun, and Daniel Weld. PDDL - The Planning Domain Definition Language. Technical Report 1165, Yale Center for Computational Vision and Control, Computer Science Department, 1998.

[3] James Glenn. An Optimal Strategy for Yahtzee. Technical Report 0002, Loyola College in Maryland, Computer Science Department, 2006.

[4] Malte Helmert. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[5] Malte Helmert and Carmel Domshlak. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pages 273–280, 2009.

[6] Michael Katz and Carmel Domshlak. Optimal Additive Composition of Abstraction-based Admissible Heuristics. In *18th International Conference on Automated Planning and Scheduling (ICAPS 2008)*, pages 174–181, 2008.

[7] Thomas Keller. *Anytime Optimal MDP Planning with Trial-based Heuristic Tree Search.* PhD Dissertation, Albert-Ludwigs-Universität Freiburg, Germany, 2015.

[8] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* 3rd edition, 2009.

[9] Tom Verhoeff. How to Maximize Your Score in Solitaire Yahtzee. 1999.

University of Basel

Declaration on Scientific Integrity

(including a Declaration on Plagiarism and Fraud)

Master's Thesis

Title of Thesis

(Near)-optimal policies for Probabilistic IPC 2018 domains

Author: Brikena Çelaj

Matriculation No.: 17-063-462

I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Basel, May 30, 2020

Brikena Çelaj

Signature

Universität
Basel

Erklärung zur wissenschaftlichen Redlichkeit

(beinhaltet Erklärung zu Plagiat und Betrug)

Masterarbeit

Titel der Arbeit

(Near)-optimal policies for Probabilistic IPC 2018 domains

Autor: Brikena Çelaj

Matrikelnummer: 17-063-462

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, Mai 30, 2020

Brikena Çelaj

Unterschrift