Proceedings of the Twentieth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2024)

A Model for Automating the Abstraction of Planning Problems in a Narrative Context

Mira Fisher, Stephen Ware

Narrative Intelligence Lab, Department of Computer Science, University of Kentucky
329 Rose Street
Lexington, Kentucky 40506
mira.fisher@uky.edu, sgware@cs.uky.edu

Abstract

Contemporary automated planning research emphasizes the use of domain knowledge abstractions like heuristics to improve search efficiency. Transformative automated abstraction techniques which decompose or otherwise reformulate the problem have a limited presence, owing to poor performance in key metrics like plan length and time efficiency. In this paper, we argue for a reexamination of these transformative techniques in the context of narrative planning, where classical metrics are less appropriate. We propose a model for automating abstraction by decomposing a planning problem into subproblems which serve as abstract features of the problem. We demonstrate the application of this approach on a low-level problem and discuss key features of the resulting abstract problem. Plans in the abstract problem are shorter, representing summaries of low-level plans, but can be directly translated into low-level plans for the original problem.

Introduction

Traditionally, automated planning is primarily used to represent complex tasks or logistical processes. In these domains, an optimal plan is one which achieves the goal with the fewest steps or lowest cost, with a secondary focus on time efficiency. Heuristic search has dominated in this environment. As such, state of the art abstractions focus on gathering just enough domain knowledge to identify the most promising regions of the search space. In this paper, however, we focus on transformative approaches to abstraction which identify and *extract* patterns within the problem to construct abstract problems. The ability to break a problem into less complex parts makes these approaches powerful, but loss of detail typically limits them to producing suboptimal plans, and the computational costs involved in both breaking down and reforming the problem may outweigh the benefits (Sebastia, Onaindia, and Marzal 2006).

Outside of its primary usage, automated planning also serves as a powerful tool for computationally modeling narratives. Narrative planning techniques can represent stories driven by intentional character behavior that is planned in advance, including actions motivated by misunderstandings or emotional responses to anticipated events (Riedl and Young 2010; Teutenberg and Porteous 2013; Shirvani and Ware 2020). As a representation it bears a resemblance to the Possible Worlds model of narratives, and it has been used to predict audience experiences of conflict, suspense, and story comprehension (Shirvani, Ware, and Farrell 2017; Ware and Young 2014; Cheong and Young 2015; Cardona-Rivera et al. 2016). In this context, the traditional performance metrics break down: rather than generating the shortest story in the least time, a narrative planner may be called upon to find the most suspenseful story, or a story where an emotional action leads to conflict, or all possible stories in a domain.

Cardona-Rivera and Young (2019) conclude that plan-based approaches to representing narratives can be leveraged as a model of the mental process of narrative sensemaking. Extending this conclusion, we suggest that abstraction techniques for plan-based narratives should be considered in terms of their potential to model the hierarchical aspects of this sensemaking process. Accordingly, we argue that the performance of an automatic abstraction process for plan-based narratives should be evaluated in terms of 1) the amount of detail removed at different levels of abstraction, 2) the utility of the remaining information for specific objectives such as identifying key story beats, and 3) the potential for introducing disruptive events in the process of reconstruction.

We motivate these ideas with one of the key concerns in narrative planning research: support for advanced features like intentionality or theory of mind seen in centrally-planned narrative systems like Ostari (Eger and Martens 2017) and Sabre (Ware and Siler 2021) is computationally expensive to a degree which limits the application of these systems to domains which are described at a high level of abstraction. As it stands, an author who wishes to utilize these systems in a domain which is not sufficiently abstract must develop an abstraction by hand, along with the transformations for *lifting* states or events in the original domain to the abstract representation, or *grounding* elements from the abstract domain in terms of the original. This process is difficult and error-prone, due in part to the possibility that the author's understanding of the domain is distorted with respect to what it ultimately models[1] (Cardona-Rivera 2020).

A process for automated abstraction could serve to bridge

[1] Such a distortion may result from the presence of unintended possibilities in the story, *i.e.* bugs.

the gap between current narrative planning limitations and the typical granularity of representation that appears in video games and other computational narrative works. Such a process could reduce the authorial burden and expertise required for constructing the abstraction and the necessary transformations, as well as having the potential to expose unintended possibilities by presenting the distortions those possibilities create in the abstract domain. To serve effectively in this capacity, the abstraction method would need to 1) discard as much non-critical detail as possible in the lifting process, 2) retain the essential information for modeling narrative events in the abstract domain, and 3) unobtrusively restore non-critical detail in the grounding process.

Having argued for reexamining the potential of automated abstraction for plan-based narratives, in the remainder of the paper we will introduce a method for the abstraction of a planning problem which supports the evaluation targets above. Our proposed method models this abstraction as a process of decomposing a problem into a network of subproblems, identifying equivalent situations within those subproblems, and discarding unnecessary detail to arrive at an abstract state. We hypothesize that 1) this method will result in an abstract problem with a significantly smaller state space than the original problem; and 2) that plans in the abstract problem will be shorter than equivalent plans in the original problem, with individual steps in the abstract plan summarizing key changes in the state of the problem.

We evaluate (1) by presenting the results of different decompositions of our experimental low-level problem, comparing it and the resulting abstract problems in terms of the size of the state space that each represents. We evaluate (2) by considering a selection of plans from the low-level problem and comparing them to equivalent plans from the abstract problem in terms of length.

Related Work

We first acknowledge the fields which hold the majority of recent research in problem abstraction, before then returning to our focus on automated planning.

State Abstraction

In the field of sequential decision making, state abstraction has been applied to Markov Decision Processes (MDPs) to more efficiently generate optimal policies (Abel et al. 2018). The methods typical of this approach derive an abstract MDP which groups states from the ground MDP if they have the same optimal policy (Li, Walsh, and Littman 2006). The importance of generating optimal or near-optimal policies for the ground MDP limits the degree of abstraction that can be applied. An aggressive abstraction may violate the Markov property expected of an MDP, making it impossible to identify the optimal policy in a state without tracking prior history. Some methods have been developed to carefully maintain this property and avoid removing too much information (Allen et al. 2021).

Hierarchical Abstraction

Hierarchical abstraction methods allow more aggressive reduction of detail, managing this loss by representing high-level transitions as subproblems in the lower level. Bai, Srivastava, and Russell (2016) argue for seeing abstract MDPs as Partially Observable MDPs (POMDPs) but use options, a common tool in hierarchical reinforcement learning (HRL), to allow transitions between abstract states without tracking prior history. This combination enables a fast approximate solution for the POMDP through tree search, which can be expanded into an optimal policy for the ground MDP. This result demonstrates promise in the ability of options to compensate for information loss, though it should be noted that the abstractions were crafted by hand.

Pateria et al. (2021) provide a thorough survey on the methods that have been explored in HRL literature. Typically these methods either break a problem into independent subtasks, or package useful strategies into subprograms which contribute to a solution. The options framework is most relevant to this paper. Options are a type of subprogram, treated as extended-length actions, enabling an agent to prioritize long-term goals with a reward signal distinct from the MDP's (Sutton, Precup, and Singh 1999). Options often serve as containers for expert knowledge that an agent can use, but automatic "discovery" of options has seen extensive investigation. Existing option discovery approaches include prioritizing return to frequently visited states, promoting exploration in a variety of ways, or seeking bottleneck states (Stolle and Precup 2002; Stolle 2004; Simsek and Barto 2007; Menache, Mannor, and Shimkin 2002). While these methods have benefits, the goals they serve are efficiency focused, *e.g.* reducing the overall number of training steps. They do not necessarily construct options with any relationship to planning at the abstract level.

Konidaris, Kaelbling, and Lozano-Perez (2018) show that options *can* enable automated planning in a low-level domain. Notably, their options allowed a deterministic planner to navigate a domain with non-deterministic transitions, a continuous feature space, and a continuous action space. The authors conclude that suitable options for planning should funnel an agent from a large set of initiation states to a smaller set of termination states, and that the set of termination states must be a subset of the initiation states of at least one other option. The options in this paper were handcrafted, but later work developed a process for discovering options that meet these criteria by explicitly connecting initiation and termination regions in a graph structure (Bagaria et al. 2021). This was accomplished by first generating options which initiate near the goal and lead directly to it, and then recursively adding new options that terminate in the initiation set of existing options. The resulting graph can be interpreted as an abstract state space, though the abstraction primarily expresses proximity to the goal. This is useful for efficiently solving an MDP, but for our purpose it is comparable to heuristic abstraction.

Abstraction in Automated Planning

Hierarchical Task Networks (HTNs) are one of the better known models for abstract planning, allowing the specification of abstract tasks which decompose into collections of primitive tasks (Georgievski and Aiello 2014). The abstract tasks and decomposition methods are typically au-

thored alongside the HTN, but automated abstract task discovery has been explored. Hayes and Scassellati (2016) develop an abstract task hierarchy by identifying bottleneck nodes in a graph of plan execution traces, grouping the tasks between those bottlenecks.

In classical planning, action decomposition has played a similar role to HTNs, allowing hierarchical planning using authored rules for decomposition—referred to as decomposition schemata (Young, Pollack, and Moore 1994). Macro-actions are conceptually equivalent, used to store useful action sequences to apply as a unit (Botea et al. 2005). Macros may be authored but are typically learned, and evaluated in terms of their value to search speed.

The approaches above, rooted in action abstraction, are limited. Transformative abstractions must ultimately reformulate the reachable state space of the problem.

Sebastia, Onaindia, and Marzal (2006) survey approaches to decomposition, which can be grouped into those which compose a solution from subgoal-solving plans (bottom-up), and those which solve an abstract problem and then fill in details to ground the plan (top-down, see *Highpoint* below). Bottom-up methods benefit from identifying independent subproblems with few negative interactions between them. The authors introduce a method which achieves this independence using landmarks—propositions which must hold at some point in any plan which solves the problem, or actions which must appear in that plan (Hoffmann, Porteous, and Sebastia 2004). This method performed well on classical planning benchmarks, but landmarks have dubious value in low-level story domains which may have too many possible paths to endings for landmarks to occur. Porteous, Cavazza, and Charles (2010) demonstrate the potential subgoals have for expressing hierarchical narrative control, but how to derive useful subgoals remains an open question.

Other approaches abstract the problem by simplifying the problem states, often by *projecting* states into an abstract space which considers only a portion of the state variables. *Highpoint* (Bacchus and Yang 1994) and similar top-down methods establish a hierarchy of abstract spaces. The most-abstract space considers only the variables critical for solving the goal directly, and each lower layer is more grounded, introducing details that are used to fill out the plan until it solves the original problem. Abstracting by projecting into abstract spaces is valuable for keeping simple lifting and grounding functions, and has recently been employed by Diamanti and Thue (2019) while simulating and synchronizing multi-agent behavior with adaptive level of detail. Our definition of subproblems is directly comparable to using abstract spaces with projections in this same way, though our abstraction ultimately differs by reinterpreting the subproblems themselves as variables.

Decoupled search is a more contemporary method comparable to the top-down approach, partitioning the set of variables to construct a "center" abstract space as well as a number of "leaf" spaces (Gnad and Hoffmann 2018). The center contains all the most critical details for search, while the leaves are arranged such that they are decoupled from each other and only ever influence the center component. As a result, the search process only ever branches on central transitions. This method has been formulated into a transformative abstraction that generates a planning domain with the leaf transitions embedded into a set of specialized variables (Speck and Gnad 2024). This is conceptually similar to our method defining regions which factor out subproblem transitions that do not impact external subproblems, though our method does not prioritize any center.

The Fast Downward planner is also a notable instance of successful abstraction, both in translating propositional planning problems into a multi-value representation that captures implicit dynamics in the problem, and for performing partial hierarchical decomposition for the causal graph heuristic (Helmert 2006). Our method draws some inspiration from Fast Downward and develops a representation of the problem that captures the implicit dynamics present in a low-level multi-value planning problem, but the similarity is only in concept and not method.

Finally, the merge-and-shrink abstraction framework presents a general approach for developing transformative abstractions of state spaces (Helmert et al. 2014). Arriving at an abstraction is a process of iterative merge steps that join two existing abstractions together, and shrink steps that reduce complexity by altering the state representation and transition labels. Like our method, it begins with a partition of the problem variables into abstractions which consider only a single variable, merges these, and then separately alters the representation to remove information. As originally introduced, it abstracts more aggressively due to its focus on providing heuristic information. However, as it is a framework, our approach can be cast in its terms: our merging strategy groups abstractions by hand, the shrinking strategy groups states that are similar with respect to the rest of the problem, and labels are reduced when operators have the same abstract reformulation.

Approach

We approach abstraction of a planning problem as a process with the following steps:

1. Decompose the problem into groups of related facts, reinterpreting them as subproblems.
2. Generate the state space associated with each subproblem, identifying all potentially reachable combinations of the subproblem facts and the transitions between them.
3. Identify regions of subproblem states which are equivalent in their effect on all other subproblems.
4. Generate the abstract problem by using subproblems as state variables, and subproblem regions as values. Retain abstract operators which transition between regions.

The results of this process are dependant on precisely what facts are treated as related and used to form subproblems. This may be manually defined, but a constructive approach is also possible—first dividing the problem such that each state variable is used to create a subproblem (closely resembling a domain transition graph) and then identifying groups of subproblems to combine. We used this constructive approach, the details of which are discussed later in this section, under the heading Identifying Subproblems.

Figure 1: The initial state in Grid Grandma. Grandma's location is represented by a house. The inventory of each character is shown: Tom has a coin, the Merchant has a sword and a potion, the Bandit has a sword and a coin, and the Knight has a sword.

Story Domain

Our approach targets stories described by low-level events interacting with a highly granular state—as is often the case with games. In service of this, the experimental problem we use is an adaptation of the "Save Grandma" narrative planning domain used by Ware et al. (2022) for a single-agent, grid world environment we call "Grid Grandma".

Grid Grandma has five characters: Tom (the player), his Grandma, a Bandit, a Knight, and a Merchant. There are six items: TomsCoin, BanditsCoin, MerchantsSword, BanditsSword, KnightsSword, and the Potion. The low-level state features include the health, grid row, grid column, criminal status, threatened status, armed status, and held item for every character; as well as who has each item, the state of trade between Tom and the Merchant, and which NPCs Tom is adjacent to. Figure 1 depicts most of the story's initial state.

The actions available to the player allow Tom to *hold* or *stow* items; to *heal*, *pay*, *slash*, *strike*, or *threaten* an NPC depending on what he is holding; to *select* an item to buy or *take* an item from an NPC; and to *move* in the cardinal directions. Finally, NPCs can *attack* Tom if they are armed, but take no other actions. Valid, complete stories are those in which Grandma is revived, or Tom is slain.

Tiny Grandma Grid Grandma is used for the results and other information later in the paper. For figures and examples in the rest of this section, we present "Tiny Grandma", which removes the Knight and Bandit and their items, places remaining characters on a 3x3 grid (see figure 2), and uses two values (Alive and Dead) to represent character health.

Domain Representation

We base our domain representation on the SAS+ formalism for classical planning problems (Bäckström and Nebel 1995), with minor adjustments.[2] A planning problem $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ is a tuple where:

- $\mathcal{V}$ is a set of state variables. Each **variable** $v \in \mathcal{V}$ is associated with a domain D_v. Together these describe the set of all **facts** in the problem $F_\Pi = \{(v, x) | v \in \mathcal{V}, x \in D_v\}$. For a given set of facts f we write $\mathbf{vars}(f)$ to denote the set of all variables among those facts.
- A **situation** t is a set of facts such that no variable appears twice (but no variables need appear at all). We may write $t[v]$ to denote the value associated with the variable v in the situation t. A **state** s is a situation in which every variable in the domain appears exactly once.
- $\mathcal{O}$ is a set of operators represented by a triple of situations $\langle \mathbf{pre}, \mathbf{post}, \mathbf{prv} \rangle$ (representing pre-, post-, and prevail-conditions respectively.

  Operators describe the potential to transition from a source state to a target state. **pre** indicates facts which must hold in the source, but not the target. **post** indicates facts which must hold in the target, but not the source. **prevail** indicates facts which must hold in both states.

  These situations are restricted by the following rules:

[2]Facts are introduced as variable-value pairs, *partial states* are redefined slightly and renamed to *situations* for the sake of clarity, the initial state must be fully specified, and no operator may introduce a fact for a previously undefined variable.

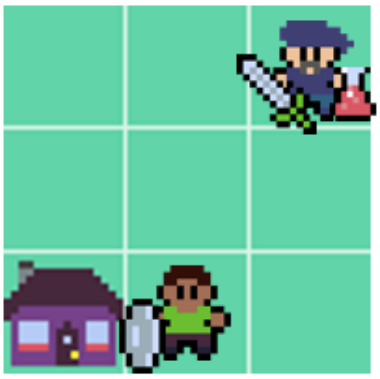

Figure 2: The initial state of the Tiny Grandma domain. Grandma's location is represented by her house, Tom has a coin, and the Merchant has a sword and a potion.

1. $\mathbf{vars}(\mathbf{pre}) = \mathbf{vars}(\mathbf{post})$
2. $\forall v \in \mathbf{vars}(\mathbf{pre}), \mathbf{pre}[v] \neq \mathbf{post}[v]$
3. $\mathbf{vars}(\mathbf{pre}) \cap \mathbf{vars}(\mathbf{prv}) = \emptyset$

Essentially, **pre** and **post** describe the state change, and **prv** describes additional fixed requirements.

Given an operator o we write $\mathbf{pre}(o)$, $\mathbf{post}(o)$, and $\mathbf{prv}(o)$ to denote each situation. Additionally, given an operator o and a pair of situations s and t such that $(\mathbf{pre}(o) \cup \mathbf{prv}(o)) \subseteq s$, $(\mathbf{post}(o) \cup \mathbf{prv}(o)) \subseteq t$ and $s - \mathbf{pre}(o) = t - \mathbf{post}(o)$, we say that o can be *applied* in s to *yield* t, a relationship which we denote as $s \xrightarrow{o} t$.

- s_0 is a state describing the initial state of the problem, and $s_\star$ describes the goal situation.

For example, the effects of move actions in Tiny Grandma are represented by operators. The situations describing these use the following state variables and associated values, with abbreviations representing each fact given parenthetically:

- *health(Tom)*: Alive (**A**), Dead (**D**)
- *column(Tom)*: 0 (**X:0**), 1 (**X:1**), 2 (**X:2**)
- *row(Tom)*: 0 (**Y:0**), 1 (**Y:1**), 2 (**Y:2**)
- *trade(Tom, Merchant)*: Trading (**T**), Not Trading (**N**)
- *criminal(Tom)*: False (**I**), True (**C**)

Each move action is represented by multiple operators. The operators that describe moving left **X:1** to **X:0** are:

1. $o_1 = \langle\{\mathbf{X:1}\}, \{\mathbf{X:0}\}, \{\mathbf{A}, \mathbf{Y:1}\}\rangle$
2. $o_2 = \langle\{\mathbf{X:1}\}, \{\mathbf{X:0}\}, \{\mathbf{A}, \mathbf{Y:0}, \mathbf{N}\}\rangle$
3. $o_3 = \langle\{\mathbf{X:1}, \mathbf{T}, \mathbf{I}\}, \{\mathbf{X:0}, \mathbf{N}, \mathbf{C}\}, \{\mathbf{A}, \mathbf{Y:0}\}\rangle$
4. $o_4 = \langle\{\mathbf{X:1}, \mathbf{T}\}, \{\mathbf{X:0}, \mathbf{N}\}, \{\mathbf{A}, \mathbf{Y:0}, \mathbf{C}\}\rangle$

Consider operators o_1 and o_2. $\mathbf{pre}(o_1)$ and $\mathbf{pre}(o_2)$ require that Tom's column position (**X**) is 1; $\mathbf{post}(o_1)$ and $\mathbf{post}(o_2)$ describe the effect of the movement, putting him in a state where **X** is 0. The prevail-condition $\mathbf{prv}(o_1)$ requires that Tom is alive (**A**) and his row position (**Y**) is 1, while $\mathbf{prv}(o_2)$ requires that **A** holds, **Y** is 0, and Tom is not trading with the Merchant (**N**). These facts must be true, but don't change.

o_1 captures the story event where Tom moves to the left without being next to the Merchant, while o_2 represents when Tom moves to the left away from the Merchant without being in a trade. o_3 and o_4 represent the consequences of Tom moving away from the Merchant to the left *after* having selected an item. This ends the trade and causes Tom to be considered a criminal (o_3) unless he already was one (o_4).

A planning problem Π is has a state space S_Π, a graph with vertexes representing the initial state and any state that can be reached from it by sequential application of operators, and an edge (s, t) for every pair of vertexes s, t such that $s \xrightarrow{o} t$. The nature of low-level domains, where operators make small adjustments to a potentially large number of state features, gives rise to a very large state space. Tiny Grandma, with 19 state variables (46 facts), has a state space of 2,582 vertexes and 10,240 edges. Grid Grandma, with 32 variables (98 facts), produces a graph with 2,242,343 vertexes and 9,390,133 edges. Even seemingly small problems like these can benefit from abstraction—they can be overwhelming when every detail looks equally important.

Decomposition

For a planning problem Π, the **decomposition network** is a group of interconnected subproblems defined by a partition of F_Π into k sets, $\mathcal{F}_N = F_0, F_1, ..., F_{k-1}$. This definition allows us to derive additional information for the network:

- For each set of subproblem facts F_i, $\mathcal{O}_i$ is an associated set of **local** operators which only cause transitions confined within F_i. That is, $\mathcal{O}_i = \{o | o \in \mathcal{O}, \mathbf{pre}(o) \subset F_i \wedge \mathbf{post}(o) \subset F_i\}$
- $\mathcal{O}_N = \mathcal{O} - \bigcup_{F_i \in \mathcal{F}_N} \mathcal{O}_i$ is the set of **global** operators. That is, it is the set of all operators that change facts in multiple subproblems at the same time.

We denote the subproblem defined by a set $F_i \in \mathcal{F}_N$ as $\pi_i = \langle \mathcal{V}_i, \mathcal{O}_i^N, s_{0,i}, s_{\star,i} \rangle$, where:

- $\mathcal{V}_i = \mathbf{vars}(F_i)$
- $\mathcal{O}_i^N = \mathcal{O}_i \cup \{o | o \in \mathcal{O}_N, \mathbf{pre}(o) \cap F_i \neq \emptyset\}$
- $s_{i,0} = s_0 \cap F_i$
- $s_{i,\star} = s_\star \cap F_i$

Just as a Π gives rise to the state space S_Π, each subproblem π_i defines a situation space S_{π_i}. In this context, the rules for application are relaxed: only the facts in F_i are considered when evaluating if an operator applies to a situation in S_{π_i} and determining what it yields. This relaxation ensures that the situation space contains any situation that might occur in the problem, though it may contain some that never occur in practice. Edges in the situation space are labeled with the ignored parts of the operator—for an operator o, an edge will have a condition given by $\mathbf{prv}(o) - F_i$. If o is a global operator, the edge is also labeled with side-effects described by $\langle \mathbf{pre}(o) - F_i, \mathbf{post}(o) - F_i \rangle$.

For example, three possible subproblems relevant to Tiny Grandma, and their associated facts, are as follows:

- **X**: {**X:0**, **X:1**, **X:2**}
- **Y**: {**Y:0**, **Y:1**, **Y:2**}
- **X**∪**Y**: {**X:0**, **X:1**, **X:2**, **Y:0**, **Y:1**, **Y:2**}

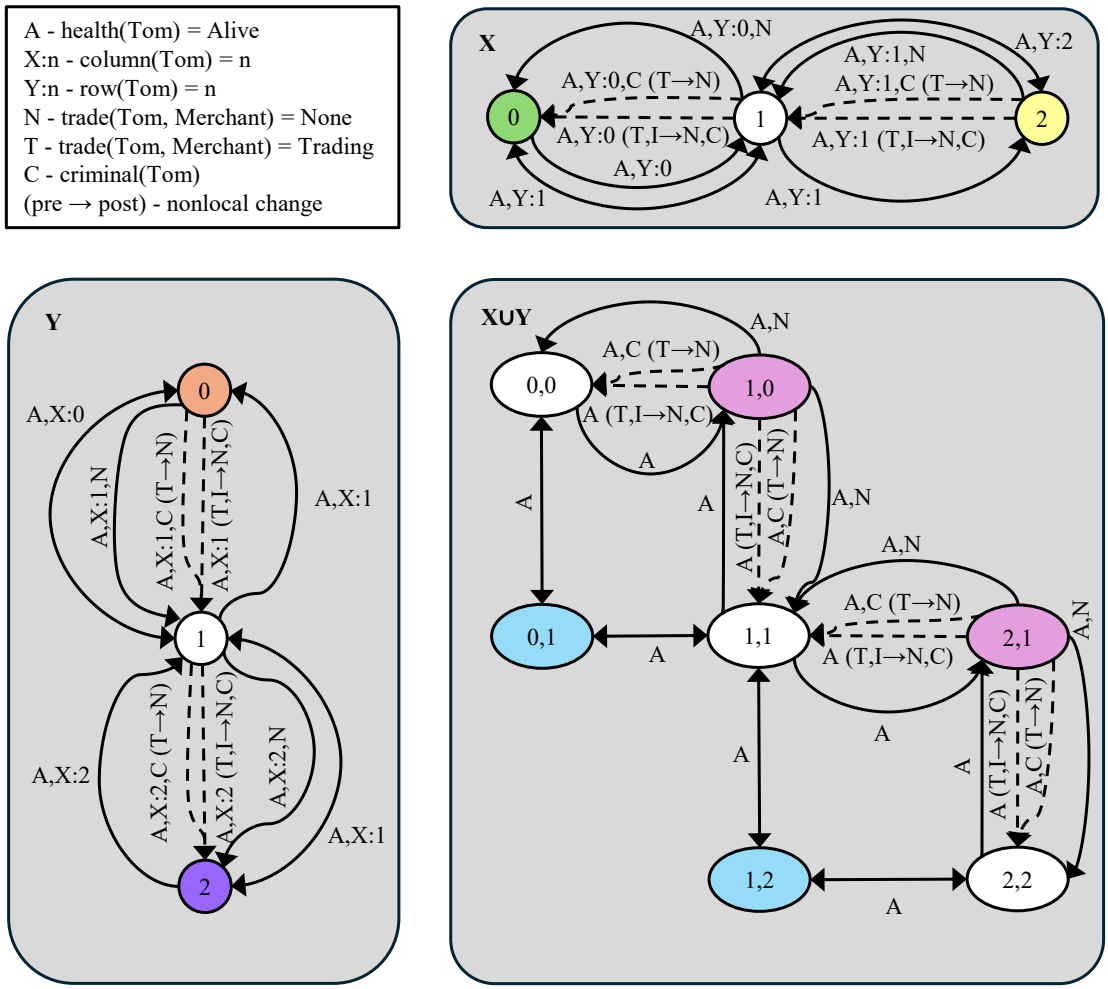

Figure 3: The situation spaces for the subproblems **X**, **Y**, and **X**∪**Y**. Edges due to local operators are solid, and dashed for global ones. In either case, the edges are marked with appropriate conditions. For clarity, local edges which have the same condition are condensed into bidirectional edges where possible. For each subproblem, vertexes are colored to denote the regions that are induced by the rest of the problem (not pictured).

Figure 3 shows the resulting situation spaces for each subproblem. While a naïve combining of the situations from **X** and **Y** would result in nine situations, access to more information results in identifying the seven *reachable* situations, accounting for the positions (0,2) and (2,0) being occupied by Grandma and the Merchant.

Abstraction

Conceptually, the conditions on edges in a situation space represent the ways in which transitions in a subproblem depend on other subproblems in the network. To determine if a transition is valid, we must "observe" the state of the subproblems containing the facts the condition checks. To avoid mistakes, the way each subproblem interacts with the rest must be tracked. Conversely, a subproblem that is not heavily observed does not need to expose as much information to the network—more detail can be ignored. It is this fundamental relationship that allows us to abstract information away without risking the introduction of obtrusive errors.

For a subproblem π_i, we identify situations that can be safely grouped into abstract *regions* by considering the relationship of each situation to every operator in the rest of the network. If we assume that the facts of the situation are true and doing so does not make that operator impossible by conflicting with any of the situations the operator describes, then the operator is considered available (along with any transition it is responsible for). After considering every situation of the subproblem in this way, the situations are grouped into regions according to which transitions they make available. Subproblem situations which satisfy part or all of the goal are treated as enabling a special END transition.

Consider the difference between **X** and **X**∪**Y** in figure 3.

In **X**, transition between **X:0** and **X:1** is impossible if **Y:2** is true, and transition between **X:1** and **X:2** is impossible if **Y:0** is true. If **Y:1** is true, then the state of **Y** does not prevent transition between those situations at all. The detailed observation of **Y** that is required to navigate **X** results in every situation in **Y** appearing different. We need three regions to represent three situations—no abstraction is possible.

However, by combining **X** and **Y** into **X**∪**Y**, the external observations between these two subproblems are tracked internally instead. The external observations that remain in the rest of the problem are such that some transitions are only available in the situations {**X:1**,**Y:0**} and {**X:2**,**Y:1**} (positions adjacent to the Merchant), and others are only available in the situations {**X:0**,**Y:1**} and {**X:1**,**Y:2**} (positions adjacent to Grandma). This allows the seven situations in **X**∪**Y** to be represented as three abstract regions.

Abstract regions form the basis of our reinterpretation of the decomposition network as an abstract planning problem. For a decomposition F_N of the problem Π, the abstract problem $\Pi_N = \langle \mathcal{V}_A, \lambda, \mathcal{O}_A, \sigma_0, \sigma_\star \rangle$ is a tuple such that:

- The set of abstract variables $\mathcal{V}_A = \{v_0, v_1, ..., v_{k-1}\}$ where $k = |F_N|$ and v_i is a unique symbol representing π_i. We denote the region associated with a subproblem situation t as $\rho(t)$. For each abstract variable $v_i \in \mathcal{V}_A$, the domain $D_{v_i} = \{\rho(t) | t \in S_{\pi_i}\}$. In other words, each subproblem is used as an abstract state variable, and the regions of that subproblem are used as abstract values.
- The lifting function λ maps states and situations from Π to abstract states and situations in Π_N. Given a state s, $\lambda(s) = \{(v_i, \rho(s \cap F_i)) | v_i \in \mathcal{V}_A\}$. That is, the facts for the abstract state are derived by identifying the matching situation in each subproblem, then producing the abstract fact associated with that situation in that subproblem.

  The lifting function applies differently in the case of a situation. Since a situation is not fully specified, it may hold in multiple subproblem situations. As such, for a situation t the lifting function $\lambda(t)$ yields a set of abstract facts which associate an abstract fact with multiple values. To manage this, for a set of facts u we write $\mathbf{poss}(u)$ to denote the set of all situations that can be derived by removing abstract facts until u is a situation.
- The set of abstract operators $\mathcal{O}_A$ is obtained by considering each operator from the original problem. For each operator $o \in \mathcal{O}$, $\mathcal{O}_A$ contains an operator $\langle t_{pre}, t_{post}, t_{prv} \rangle$ for each $t_{pre} \in \mathbf{poss}(\mathbf{pre}(o))$, $t_{post} \in \mathbf{poss}(\mathbf{post}(o))$, and $t_{prv} \in \mathbf{poss}(\mathbf{prv}(o))$, if $t_{pre} \neq t_{post}$
- The abstract state initial state $\sigma_0 = \lambda(s_0)$.
- The abstract goal situation $\sigma_\star = \lambda(s_\star)$. (Regions are derived so goal situations in a subproblem share a region.)

In the abstract problem for Tiny Grandma, the subproblem **X**∪**Y** hides the low-level detail of grid row and column, instead representing the conceptual states of Tom's position as "adjacent to Grandma" (**G**), "adjacent to Merchant" (**M**), and "elsewhere" (**E**). Even if we expand the underlying grid in Tiny Grandma, the abstract problem always represents position with these three values.

As a result, the initial state will now contain the abstract fact **G** and no facts from **X** and **Y**, and we reinterpret the operators used earlier as:

1. $o_1 = \langle \{\mathbf{E}\}, \{\mathbf{G}\}, \{\mathbf{A}\} \rangle$
2. $o_2 = \langle \{\mathbf{M}\}, \{\mathbf{E}\}, \{\mathbf{A}, \mathbf{N}\} \rangle$
3. $o_3 = \langle \{\mathbf{M}, \mathbf{T}, \mathbf{I}\}, \{\mathbf{E}, \mathbf{N}, \mathbf{C}\}, \{\mathbf{A}\} \rangle$
4. $o_4 = \langle \{\mathbf{M}, \mathbf{T}\}, \{\mathbf{E}, \mathbf{N}\}, \{\mathbf{A}, \mathbf{C}\} \rangle$

The goal situation is unchanged.

Identifying Subproblems

The problem of identifying ideal subproblems for a given problem is fundamental to the pursuit of useful abstractions. In this section, we present the informal considerations which lead to the partition of facts used for Grid Grandma.

In general the ideal subproblem is *isolated*, having only a minimal impact on the rest of the problem as it is navigated. In our model, these impacts are considered in terms of observations and regions. Observations from outside of a subproblem divide it into regions such that moving from one region to another represents the potential for a negative interaction. Considering this, an exemplary isolated subproblem should group many situations into a few (ideally contiguous) regions. Regions are determined by observations, so the intuitive approach is to seek groupings of facts that will reduce the number of unique observations present in the network.

In Grid Grandma, the subproblem of grid navigation exemplifies this circumstance. Due to obstacles on the map, the operators which change Tom's grid row frequently observe the grid column, and vice versa. This relationship of mutual dependency produces a large number of external observations for both subproblem—observations that become internal if these subproblems are combined. The state variables which track adjacency are not heavily observed, but they are the only other source of observations of row and column both. Combined, the 32 facts related to these six state variables form the **Adj** subproblem, with 56 situations that are grouped into 5 regions. (The **Adj** subproblem is analagous to the **X**∪**Y** subproblem depicted in figure 3).

Another use for abstractions is to make implicit dynamics between state variables explicit. If a subproblem is small, it is possible to completely enumerate all interactions between those facts and all possible states that subproblem may take on. Identifying the possible states of a subproblem, and all *im*possible states by extension, can provide information that makes the rest of the problem easier to understand or solve.

In Grid Grandma, this intuition lead to forming four subproblems, each composed from facts relevant to one of the characters in the problem. The **Prog** subproblem combines facts about Grandma's health with the location of the Potion. **TStat** combines Tom's health and armed status with the location of the MerchantsSword, as well as details about the Merchant's condition. **BStat** and **KStat** respectively combine the Bandit's and Knight's health and other conditions with the location of each character's sword. Each of these subproblems captures dependencies between the facts, resulting in subproblems that are still of a manageable size while condensing more information about the problem.

Domain	Subproblems	Situations	Regions
GG-Base	32	98	85
GG-Adj	27	122	59
GG-Prog	26	121	58
GG-TStat	20	125	52
GG-BStat	16	125	49
GG-KStat	12	125	44
GG-Coins	11	126	45

Table 1: Details about the domain network after each subproblem has been created, listing total number of subproblems, subproblem situations, and subproblem regions in each the network.

Domain	States	Endings	Edges
Grid Grandma	2,242,343	4,369	9,390,133
GG-Base	461,658	2,353	1,625,311
GG-Adj	92,027	1,562	298,051
GG-Prog	92,027	1,562	298,051
GG-TStat	57,840	802	149,008
GG-BStat	49,944	802	124,972
GG-KStat	40,440	802	109,504
GG-Coins	40,440	802	10,504

Table 2: Details about the state space for the original problem, as well as the the state space for the abstract problem after each subproblem has been created.

Finally, we considered similar use-cases of state variables as a motivation to group them into a subproblem. Since regions are a determination of what can be accomplished outside of a subproblem based on the internal state, grouping facts which contribute to enabling similar operators has the potential to widen the associated region. This lead us to join the facts about the location of the two coins, TomsCoin and BanditsCoin, into the single subproblem **Coins**.

Results

We reached the final abstract graph by grouping each set of identified subproblem facts in turn. We provide an identifier for each of the following:

1. **GG-Base**: the default partition in which each set of subproblem facts is the set of all facts associated with a single state variable, corresponding directly with domain transition graphs.
2. **GG-Adj**: as (1), but with the facts associated with the state variables for grid row, grid column, and Tom's adjancency to each NPC grouped together.
3. **GG-Prog**: as (2), but with the facts for Grandma's health and the Potion location grouped together.
4. **GG-TStat**: as (3), but with the facts for Tom's health, Tom's armed status, the MerchantsSword's location, and the Merchant's health, armed status, threatened status, and held item grouped together.
5. **GG-BStat**: as (4), but with the facts for the BanditsSword's location, and the Bandit's armed status, threatened status, health, and held item grouped together.
6. **GG-KStat**: as (5), but with the facts for the KnightsSword's location, and the Knight's armed status, threatened status, health, and held item grouped together.
7. **GG-Coins**: as (6), but with the facts associated with the location of TomsCoin and BanditsCoin grouped together.

Key information about the each decomposition network is given in table 1: number of subproblems, the number of situations across all subproblems, and the number of abstract regions across all subproblems. Table 2 gives details about the size of each state space associated with the abstract problem for each network, as well for the original problem.

The primary benefit of finding good combinations of subproblem facts, especially ones which isolate subproblems, is conveyed through decreases in the total number of regions. As seen in the **Adj** subproblem, this benefit may come at the cost of inflating the number of situations internal to that subproblem. As long as the subproblem's situation space does not become unwieldy to generate fully, this cost should be seen as minor—at 56 situations total, **Adj** is still very small.

The subproblems that lead to reductions in the size of the state space were **Adj** (86.6%), **TStat** (37.2%), **BStat** (13.7%), and **KStat** (19%). These were associated with decreases in the number of regions by 26, 6, 3, and 5 respectively. It should also be noted that even the abstract domain derived from the default network was associated with a 69.5% reduction from the size of the raw state space.

The reduction in unique endings in the default network, as well as the networks after forming **Adj**, and **TStat** represent some of the summary power of our approach. Since regions hide information that is only internally important, the abstract problem presents states that only differ in terms of that minutiae as equivalent. For example, in the original problem there are three positions from which Tom can heal Grandma. As a result, there are three versions of every ending where Tom heals Grandma, varying only by position. The **Adj** subproblem hides irrelevant position information, so these endings (and others) are equivalent. The base abstraction reduces the number of endings by 46.1% by hiding some information about health values and which sword Tom is holding. By abstracting precise positions, **Adj** reduces the number of endings by another 33.6%. By abstracting information about the Merchant's health and threatened status, **TStat** reduces it by a further 48.7%.

Overall, the final abstract state space is 1.8% of the size of the original state space, and it reduces the number of endings that are considered unique by 81.6%.

To understand the impact of the problem on plan lengths, we consider the following specific stories:

1. Tom buys the Potion and heals Grandma.
2. Tom is slain by the Bandit.
3. Tom steals the Potion and heals Grandma.
4. Tom steals the Potion and is defeated by the Knight.
5. Tom buys the MerchantsSword, defeats the Bandit, uses her coin to buy the Potion, and heals Grandma.
6. Tom buys the MerchantsSword, slays the Merchant, steals the Potion, and heals Grandma.

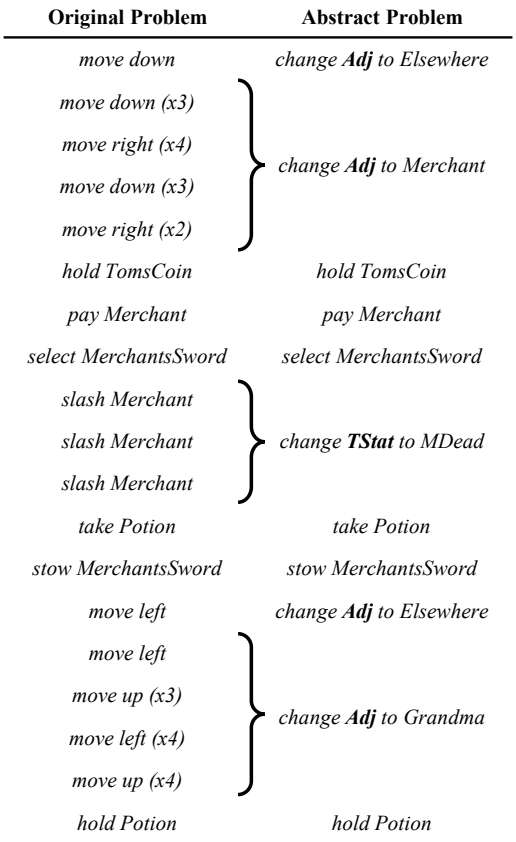

Figure 4: A depiction of a plan in the low-level problem and the corresponding events in the abstract problem.

The 6th story is shown in detail in figure 4, with the associated abstract events.

The shortest plans for producing these stories are:

1. 30 low-level actions; 8 abstract events.
2. 18 low-level actions; 3 abstract events.
3. 28 low-level actions; 6 abstract events.
4. 18 low-level actions; 6 abstract events.
5. 71 low-level actions; 18 abstract events.
6. 36 low-level actions; 12 abstract events.

Stories 2 and 4 are tied for the shortest low-level plans that reach an ending of the story. Story 3 is the shortest story in which Tom heals Grandma.

By condensing sequences of movement actions and attacks, the length of plans are significantly reduced, and the events that replace sequences of small changes instead summarize the change at a higher level of abstraction.

Conclusions

In this paper, we argued for reexamining the potential that transformative abstraction methods have in the context of plan-based representations of narrative, and proposed criteria to consider in evaluating such methods.

We described a model that builds on existing ideas of using decomposition to break a planning problem into smaller subproblems, but 1) ultimately derives an abstract planning problem which can be planned within without considering the original problem, and 2) incorporates a novel approach to deriving abstract symbols by considering the potential for interactions between subproblems.

We also showed that the abstract problem that our method constructs is significantly smaller than the original problem in terms of the number of vertexes and unique endings present in terms of the state spaces each problem represents. We also showed that the length of the plans that represent key stories in the original problem are represented by significantly shorter plans in the abstract problem.

The results of this paper represent a first step in the goal of providing robust abstraction of planning problems with a focus on supporting narrative planning.

Future Work

This paper lays groundwork that we anticipate expanding on by experimenting with more problems drawn from existing computational narratives, incorporating further considerations useful for narrative planning, and evaluating our abstraction in terms of how human subjects perceive it.

This paper presents the results of early development of this model on a problem written to serve as a low-level model of an existing narrative planning problem. We plan to experiment with this approach on low-level domains that exist "in the wild", adapting our approach and expanding our methods to be more generally applicable. In particular, we plan to develop planning domain versions of narratives drawn from interactive fiction, emergent-narrative story generation systems, and narrative-focused video games.

We plan to expand on the approach in this paper by incorporating concepts such as character goals and long-term consequences of events. In addition to potentially guiding behavior, character goals enable a better understanding of the interactions between characters, allowing us to recognize and respond to the potential for conflict or cooperation. By incorporating character goals into our model, we might produce an abstract problem and additionally model a "conflict space" that describes the way that character goals relate to the abstract problem and each other. Similarly, we may also model the ways that events in a narrative conflict with the other events that might have happened instead, leading to a better representation of the problem in terms of the long term impact of choices between these events.

Finally, we briefly mentioned the connection between plan-based narratives and the mental process of narrative-sensemaking. We plan to investigate abstraction of narrative planning problems in terms of this relationship. We hope to discover if the methods of abstraction that prove fruitful in terms of our proposed evaluation targets are also methods of abstraction that seem familiar or reasonable to human subjects. We also hope to explore an approach to abstraction that attempts to specifically model how humans process narrative information into abstract hierarchies.

References

Abel, D.; Arumugam, D.; Lehnert, L.; and Littman, M. 2018. State abstractions for lifelong reinforcement learning. In *International Conference on Machine Learning*, 10–19. PMLR.

Allen, C.; Parikh, N.; Gottesman, O.; and Konidaris, G. 2021. Learning Markov State Abstractions for Deep Reinforcement Learning. *CoRR*, abs/2106.04379.

Bacchus, F.; and Yang, Q. 1994. Downward refinement and the efficiency of hierarchical problem solving. *Artif. Intell.*, 71(1): 43–100.

Bäckström, C.; and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4): 625–655.

Bagaria, A.; Senthil, J.; Slivinski, M.; and Konidaris, G. 2021. Robustly Learning Composable Options in Deep Reinforcement Learning. In *Proceedings of the 30th Int. Joint Conf. on Artificial Intelligence*.

Bai, A.; Srivastava, S.; and Russell, S. 2016. Markovian State and Action Abstractions for MDPS via Hierarchical MCTS. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, 3029–3037. AAAI Press. ISBN 9781577357704.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24: 581–621.

Cardona-Rivera, R. 2020. Foundations of a computational science of game design: Abstractions and tradeoffs. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, 167–174.

Cardona-Rivera, R. E.; Price, T.; Winer, D.; and Young, R. M. 2016. Question answering in the context of stories generated by computers. *Advances in Cognitive Systems*, 4: 227–245.

Cardona-Rivera, R. E.; and Young, R. M. 2019. Desiderata for a computational model of human online narrative sensemaking. In *Working notes of the 2019 AAAI Spring Symposium on Story-enabled Intelligence*.

Cheong, Y.-G.; and Young, R. M. 2015. Suspenser: a story generation system for suspense. *IEEE Transactions on Computational Intelligence and Artificial Intelligence in Games*, 7(1): 39–52.

Diamanti, M.; and Thue, D. 2019. Automatic abstraction and refinement for simulations with adaptive level of detail. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, 17–23.

Eger, M.; and Martens, C. 2017. Practical specification of belief manipulation in games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 13, 30–36.

Georgievski, I.; and Aiello, M. 2014. An overview of hierarchical task network planning. *arXiv preprint arXiv:1403.7426*.

Gnad, D.; and Hoffmann, J. 2018. Star-topology decoupled state space search. *Artificial Intelligence*, 257: 24–60.

Hayes, B.; and Scassellati, B. 2016. Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 5469–5476. IEEE.

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

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM (JACM)*, 61(3): 1–63.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research*, 22: 215–278.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research*, 61: 215–289.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a Unified Theory of State Abstraction for MDPs. In *International Symposium on Artificial Intelligence and Mathematics, AI&Math 2006, Fort Lauderdale, Florida, USA, January 4-6, 2006*.

Menache, I.; Mannor, S.; and Shimkin, N. 2002. Q-Cut - Dynamic Discovery of Sub-Goals in Reinforcement Learning. In *Proceedings of the 13th European Conf. on Machine Learning*, ECML '02, 295–306.

Pateria, S.; Subagdja, B.; Tan, A.-h.; and Quek, C. 2021. Hierarchical Reinforcement Learning: A Comprehensive Survey. *ACM Comput. Surv.*, 54(5).

Porteous, J.; Cavazza, M.; and Charles, F. 2010. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 1(2): 1–21.

Riedl, M. O.; and Young, R. M. 2010. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research*, 39: 217–268.

Sebastia, L.; Onaindia, E.; and Marzal, E. 2006. Decomposition of planning problems. *Ai Communications*, 19(1): 49–81.

Shirvani, A.; and Ware, S. G. 2020. A formalization of emotional planning for strong-story systems. In *Proceedings of the 16th AAAI international conference on Artificial Intelligence and Interactive Digital Entertainment*, 116–122.

Shirvani, A.; Ware, S. G.; and Farrell, R. 2017. A possible worlds model of belief for state-space narrative planning. In *Proceedings of the 13th AAAI international conference on Artificial Intelligence and Interactive Digital Entertainment*, 101–107.

Simsek, O.; and Barto, A. 2007. Betweenness centrality as a basis for forming skills. *Technical report, University of Massachusetts, Department of Computer Science*.

Speck, D.; and Gnad, D. 2024. Decoupled Search for the Masses: A Novel Task Transformation for Classical Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 546–554.

Stolle, M. 2004. *Automated Discovery of Options in Reinforcement Learning*. Master's thesis, McGill University.

Stolle, M.; and Precup, D. 2002. Learning Options in Reinforcement Learning. In *Proceedings of the 5th Int. Symp. on Abstraction, Reformulation, and Approximation*, 212–223. Springer.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1–2): 181–211.

Teutenberg, J.; and Porteous, J. 2013. Efficient intent-based narrative generation using multiple planning agents. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 603–610.

Ware, S. G.; Garcia, E. T.; Fisher, M.; Shirvani, A.; and Farrell, R. 2022. Multi-agent narrative experience management as story graph pruning. *IEEE Transactions on Games*. (forthcoming).

Ware, S. G.; and Siler, C. 2021. Sabre: A Narrative Planner Supporting Intention and Deep Theory of Mind. In *Proceedings of the 17th AAAI International Conference on Artificial Intelligence and Interactive Digital Entertainment*, 99–106.

Ware, S. G.; and Young, R. M. 2014. Glaive: a state-space narrative planner supporting intentionality and conflict. In *Proceedings of the 10th AAAI international conference on Artificial Intelligence and Interactive Digital Entertainment*, 80–86.

Young, R. M.; Pollack, M. E.; and Moore, J. D. 1994. Decomposition and Causality in Partial-order Planning. In *AIPS*, 188–194.