Chapter 1

INTRODUCTION & MOTIVATION

The ability of a planning system to quickly adapt to environmental changes is critical in time-constrained domains. Online, heuristic plan repair approaches are sufficient for small changes in the environment; however, repeated or large changes can cause plan quality to degrade. I present an approach that uses available offline time to analyze the space of potential changes in the environment and creates a mapping between problem instances and solutions for use during runtime. I show that this approach allows a system to rapidly adapt to changes, while yielding plan quality that is comparable to traditional online approaches.

1.1 Planning

Planning is the branch of artificial intelligence concerned with efficiently generating sequences of actions, i.e., *plans*, to achieve goals. Typically, planning domain consists of a set of states, described by state variables; a set of available actions, described by their effect on state variables; and one or more goal states. A planning problem defines a starting state, and the task of the planner is to find a set of actions that transform the starting state into a goal state. Depending on the complexity of the problem, finding any feasible plan may be satisfactory; in other cases, finding the least expensive plan in terms of length or some total action cost is desired.

Planning in which the planner has a limited amount of time with which to produce a plan is called *time-constrained planning*. This type of planning applies to situations in which the usefulness, or *utility*, of a plan degrades over time. Typically, a tradeoff exists between spending more time searching the space for a better plan and quickly deciding on a plan that may have lower utility. When some prior plan already exists, the planner can either repair the current plan, which is typically faster, or replan, which typically yields better utility.

I will use the traveling salesman problem (TSP) planning problem as a reference problem throughout much of this dissertation. The TSP requires a solver to find the shortest route that visits a given set of locations. This is a classic NP-complete problem that has been studied widely in computer science. In the basic problem, all the locations are static. The dynamic variant, the DTSP, allows locations to be introduced to the planner after execution begins.

1.2 Overview of Problem Space Approximation

Instead of computing approximate solutions at runtime, my approach is to precompute a library of high-quality solutions *prior* to runtime. In the case of DTSP, one could imagine a library containing a high-quality[1] solution for every possible combination of potential new destinations. Obviously, as the scale of the planning problem increases, the level of complexity precludes creating a comprehensive library, so in practice a library can only contain a subset of all possible plans. Therefore, I also introduce methods to ensure that the library contains appropriate plans for use when the planning environment changes.

An understanding of the problem space characteristics can be used to choose the plan-

[1]"high-quality" refers to the plan generated by an offline solver. Since a solver generally cannot guarantee that the solution to an NP-complete problem is optimal, I describe the resulting solutions as "high-quality" rather than optimal, ideal, or exact.

ning scenarios for which to generate solutions. In particular, identifying regions of problem instances with identical solutions allows for the efficient creation of a mapping from problem instances to solutions, called a *Problem-Solution Map* (PS Map). A PS Map is a component of *problem space analysis* (PSA), which allows a system to make informed decisions about which solutions to include in the library.

Problem instances contain characteristics that are identical, called *static characteristics*, and characteristics that differ between them, the *variable features*, that lead to differences in the problem instance solutions. The PS Map represents a library of solutions for problem instances, indexed by the variable features of the set of problem instances. This map provides a mapping from a problem instance to its solution, showing the changes in the solutions as a function of the variable features within the problem instances. I will discuss several techniques to efficiently build this map.

1.3 Summary of Contributions

This dissertation contributes an approach to real-time planning that leverages offline time to generate a plan library. Chapter 3 introduces the concept of a PS Map and describes several novel approximation approaches in order to create the plan library. I then demonstrate this approach's applicability to multiple domains through experiments in Chapter 4. These experiments illustrate that good approximate PS Map can be obtained from a small number of samples in the problem space. This dissertation also briefly examines the practical tradeoffs between online and offline planning time.

Chapter 2

BACKGROUND & RELATED WORK

My work primarily focuses on developing a plan library for future use the planning environment changes. Related work for two plan reuse strategies, *universal planning* and *case-based reasoning* are presented below. I then present two alternatives to *a priori* planning: *Robust planning* techniques generate plans that may be viable even when the environment changes. *Plan repair* attempts to modify an existing plan during execution in response to changes in the environment.

I then discuss several works that leverage domain space and plan space information. The final sections present related work in sampling and classification literature.

2.1 Plan Reuse & Plan Caching

Building a plan library is similar to the general notion of plan caching and plan reuse. The concept of plan caching in anticipation of future use is evident in backbone planning, where partial plans are precomputed; case based reasoning (CBR), where previously executed plans are stored; and universal planning, where complete plans are precomputed. In this section I'll briefly discuss universal planning and case-based reasoning. Backbone planning is addressed in section 2.4.

2.1.1 Universal Planning

Universal planners, also called reactive planners, are those that preemptively store plans in order to react quickly to new information. One classic approach is Schopper's universal plans (Schoppers, 1987, 1989, 1994; Chapman, 1989), in which a solution to every possible situation is stored in a plan library. The drawback of this technique is the sheer number of states that must be considered (Ginsberg, 1989b,a; Jonsson and Bäckström, 1996). Jonsson and Bäckström (1996) formally bound the size of a universal plan library for general planning problems. They conclude that naïve universal planning is not feasible, but the advantage of reactive planning in dynamic environments makes exploration of efficient universal planning for specific applications worthwhile. My work attempts exactly this.

2.1.2 Case-Based Reasoning

Identifying the minimal solution set required to achieve competent coverage of a problem space is well studied in the case-based reasoning (CBR) literature. Typically, a CBR system will encounter a problem and store the solution for future use. CBR is normally used in domains with discrete representations, although this is not always the case (Ram and Santamariá, 1997). In most cases, CBR does not truly pre-plan; rather, all of its stored solutions are generated during runtime. Conversely, the strategies in this dissertation seek to generate its store of solutions prior to runtime. Still, my research does borrow from work in this field.

Smyth and McKenna (2001) measure the competence of a library by how well it covers the problem space. Smyth and McKenna rely on a "Solves" predicate to determine if a solution is suitable for a problem instance ("case" in CBR vernacular), and uses this to evaluate the library's competence. This can also be used to reduce the library size by removing redundant cases. My work is similar in that it seeks to determine a library's competence, but differs in the metric applied. I proactively generate solutions that cover the complete problem space, whereas CBR typically only stores solutions to problems encountered during runtime. In order to associate an unsolved problem instance with a solution, both our works may use a k-nearest neighbor approach. Interestingly, Massie et al. (2003) empirically demonstrate that Smyth and McKenna's model does not adequately predict a library's competence.

The McSherry (2000) coverage model attempts to explicitly enumerate the set of problems that a solution set can solve. As Smyth and McKenna note, this type of brute force approach is not scalable to most CBR systems. My approach creates a representation similar to McSherry's model, but attempts to resolve the scalability challenge by using approximations.

As an alternative to traditional case-based retrieval, McSherry's later 2003 work suggests a scheme in which cases beyond those chosen by a traditional nearest neighbor approach are considered. Within this scheme, compromises are suggested to a user based upon a more nuanced representation of the problem or user preferences. McSherry's system does not require an exclusively exact match of the user's preferences, and is guided by policies such as *more-is-better*, *less-is-better*, or *nearer-is-better*. Additionally, the scheme will offer solutions that may violate the constraints, but offer higher utility in other dimensions.

2.2 Planning Robust Solutions

The approaches in the previous section address adapting to the environment by caching multiple plans. *Conditional planning* is another approach to planning within changing environments in which a plan contains steps that branch based upon an environment state. For example a plan may dictate "if the left turn signal is green, then turn, otherwise go straight." *Contingency planning* also uses branches, but only in the case of failures. In one implementation of a conditional planner, Onder and Pollack (1996) identify the contingencies to plan for by calculating an expected *disutility* for an action that fails. Their planner chooses the actions with the highest disutility and generates a plan from a hypothetical state in which the action fails. Onder and Pollack define actions and their probabilistic effects as branches. For example, consider a factory that preprocesses parts for painting. If the part is not processed properly, then there is a 5% chance that the painted part will have a blemish. If the PAINT action is invoked from a state where a part is not processed, it will have two branches: one representing the transition to a state of a painted part with blemishes, and the other representing the state of the part without blemishes. To start, Onder and Pollack create a skeletal plan in a STRIPS-like manner, without regard for contingencies. After completing the plan, the planner searches the tree for high measures of *disutility*, such as that represented by the existence of a blemished part, and the plan is refined by adding actions that would resolve the effects of the failed PAINT action.

Limits of Onder and Pollack's research include the need to enumerate all the effects and contingencies related to actions. In a large or continuous domain, the effects or contingencies will be numerous or infinite. Additionally, this research is limited to plans that can be divided into hierarchical goals. Both Onder and Pollack's and my approaches generate contingencies for future adaptation needs. However, my work is intended to address a comprehensive set of changes instead of only preparing for a subset. Additionally, my work does not require enumeration of action effects, but does require some knowledge of the possible values of each state variable.

Burns et al. (2012) introduces *online continual planning problems* (OCPPs) in which a planner continually receives new goals that it must prioritize while executing its current plans. This situation is representative of domains such as using UAVs to monitor a region; because the environment continually changes, the region is never successfully "monitored".

Rather, success is the ability to continually respond to the new requirements within a suitable amount of time. They introduce *anticipatory online planning* in which they consider future changes to the environment in their current planning. Similar to my approach, they sample from the set of possible environmental changes. However, they do assume a known probability distribution for these changes. Also, they incorporate this information into the current plan in order to either resolve the goal or strategically place the system in a state that facilitates resolving goal. This is distinct from my approach in which plans are generated with the intent of being optimal for the known set of goals, but keeping plans in a library that are optimal for a future set of goals.

Conrad et al. (2009) describe an approach to planning in dynamic environments through building options into a high-level plan, thus allowing a planner to choose the best option during runtime. However, deciding between the choices can result in a significant time cost. This can be mitigated by generating the choices offline, and storing the choices efficiently by recording a baseline plan and representing additional plans by recording the difference from the baseline plan. This allows more rapid

2.3 Plan Repair & Replanning

My dissertation proposes algorithms that efficiently precompute a set of plans to mitigate changes in a planner's environment. The major alternative to my approach is replanning through plan repair. Typically planners employing this scheme will execute a plan until the environment changes, effectively creating a new problem instance. It then will modify, or *repair*, the existing plan until it is applicable to the new problem instance. This is generally faster than creating a new plan from scratch with the tradeoff that the repaired plan may not be as good as a plan generated by a complete replan.

One example of is the SALIX planner (Logan and Poli, 1997) starts with a complete plan and creates new plans through various deforming operations. In this way, the planner finds a suitable plan by searching through a solution space as opposed to a state space. This is closely related to planning schemes that employ plan repair techniques as their primary mechanism.

A domain-generic solution by van der Krogt and de Weerdt (2005) presents a framework that intends to encompass a verity of plan repair algorithms. They describe plan repair as consisting of removing actions from the original plan that conflict with or impede achieving the new goal during the *unrefinement* stage. This is accompanied by the *refinement* stage where actions are added to the original plan that allow it to achieve the new goal. The framework thus implements plan repair as a process alternating between unrefinement and refinement until a solution candidates satisfies the problem requirements. For the final test domain, I've implemented to create an online repair baseline that follows this framework.

2.4 Domain and State Space Analysis

Several related works leverage plan or problem space analysis to find critical partial plans for future use. These planners take advantage of characteristics specific to a domain or problem type. Bulka and desJardins describe learning features of a plan space to find a "backbone" common to a set of problem instances to use as a partial initial solution for planning (Bulka, 2006; Bulka and desJardins, 2008). In other cases, robots can learn critical components of plans as "skills" that may be applied to future plans (Konidaris, 2008; Konidaris and Barto, 2008). These works and my approach have similarities, but our approach focuses on storing complete plans in contrast to partial plans.

Hoffmann (2001) characterizes the topology of the planning spaces of benchmark planning problems to gain a measure of their difficulty. For example, a large number of states representing local minima may represent an easier problem, while a large number of states on local plateaus with few exit states ("benches"), or a large number of dead ends represents a difficult problem. This work demonstrates the relationship between the planning space characteristics and the success of the selected heuristic.

The hill-climbing algorithm takes advantage of the frequently continuous surface representing solution utility as a function of a specific problem instance. By slightly modifying the solution, the algorithm can determine the gradient of the hill and search in the proper direction for better solutions. The "restart hill-climbing" approach executes the hill-climbing algorithm for multiple starting solutions in order to increase the change of finding a globally optimal solution. Otherwise, the algorithm risks limiting its search to a locally optimal region.

The theme of characterization of a space of problem or solutions through a small set of samples is echoed in several works. Boyan and Moore (2000)'s *Stage* algorithm augments the traditional restart hillclimbing algorithm by using results from the multiple iterations of restart hillclimbing to estimate the relationship between the starting state and the quality of the final state, as measured by an objective function. In this way, Stage can estimate the initial state that is most likely to optimize the objective function.

Stage varies the initial state to map the relationship between the starting state and the final state within a single problem instance. By contrast, our approach varies the problem instance to map the relationship between a problem instance and problem solution within a set of problem instances. Thus, our approach is more analogous to Boyan and Moore's brief description of their *X-Stage* algorithm, which explores how information from one problem instance can be applied to other instances. X-Stage uses the Stage feedback from multiple previously solved instances as the input to a voting mechanism that informs the starting state for unsolved problem instances. Boyan and Moore's voting approach parallels our SC-based algorithms. In their case, the results were mixed. In both experiments, the

X-Stage algorithm approached the solution more rapidly than Stage, but in one experiment, the solution achieved by X-Stage was inferior to that achieved by Stage.

Gopal and Starkschall (2002) use plan space visualization to quickly compare tumor treatment plans. A plan consists of a vector trajectory over which to apply radiation. Because a trajectory will generally pass through both healthy tissue and tumor, plans that minimize healthy tissue's exposure and maximize the tumor's exposure are preferred. To assist physicians with choosing a treatment plan, the effects of multiple plans are calculated and plotted into an n-dimensional plan space with axes representing the effect on the various organs. Their work is similar to ours in terms of indexing of plans. However, our work indexes plans by the characteristics of the problem being solved, whereas Gopal and Starkschall's work indexes plans by characteristics of the plan. Additionally, any visualizations generated by my work are tangential artifacts, whereas Gopal and Starkschall's visualizations are intended as the primary product. A natural extension of their work would be to infer the utility of plans not explicitly addressed by their solver, similar to my motivations. The authors present some initial thoughts about more rapidly populating the plan space with better automation of the calculations, but do not consider inferring plan characteristics. Given the critical nature of their domain, explicitly performing calculations is likely the more appropriate approach.

The TIM domain analyzer, used within the STAN4 planner (Fox and Long, 2001), recognizes subproblems characteristic of path-planning or resource management problems and routes them to the FORPLAN planner, a planner optimized for those domains. Other subproblems are sent to the domain-generic STAN3 planner. Thus, Fox and Long decompose a planning problem into subproblems that map to domains for which domain-specific algorithms can be utilized. One aspect of this disseration's suggested future work is to decompose a homogenous planning problem in a general fashion, matching the problem instance to an appropriate algorithm chosen from a set of options.

Domshla et al. (2010) seeks to optimize the use of multiple heuristics in search. Their goal is to optimize the tradeoff between spending too much time calculating heuristics for states that will be expanding regardless of the results of more expensive heuristics versus spending too little time calculating heuristics and wasting time expanding states that do not contribute to the optimal solution. They introduce a map of the state space showing the ideal heuristic to employ at each state and their goal is to learn the map by taking samples from the state space as input to a Bayes Net, thus identifying the relative accuracy of the heuristics as a function of the location in the search space. During the search the heuristic is chosen by computing the tradeoff between each heuristic's computation time and expected accuracy. With this approach they achieve better results than the use of either individual heuristic. Their approach analogous to mine in that they explicitly define an ideal map which they attempt to approximate through sampling and classification.

2.5 Classification

The classification techniques used in my algorithms are based upon k-nearest neighbor (kNN) and support vector machines (SVM).

K-nearest neighbor (Cover and Hart, 1967) is a simple approach to classification in which a data point is classified by surveying the classification of its k nearest neighbors. The data point is then classified based on the plurality vote of the classifications. A variation of this technique is used in various algorithms that I present.

A support vector machine (Cortes and Vapnik, 1995) uses a hyperplane to divide a space such that distance between the hyperplane and points of differing classifications is maximized. Newer techniques allow for non-linear division by using the "kernel trick" in which a space is transformed to make a linear division of the space is possible.

2.6 Sampling Techniques

My research relies on an initial sampling of the planning space to seed the subsequent classification. The classification is thus dependent on a sample that adequately represents the planning space. The primary techniques used in this dissertation are random sampling and active learning and are described below. I also discuss several related alternates.

Active learning (Settles, 2010) techniques iteratively refine an interpolation by acquiring additional information after each completed interpolation. One approach for classification, *minimum marginal hyperplane*, requests information about points close to the hyperplane that a support vector machine would construct.

Maximum curiosity is an alternate approach that tests each unknown data point to see which would be most beneficial to increase accuracy. To scale to a large amount of data points, such a technique would have to choose a subset of the points to consider.

Several sampling techniques stem from the experimental design domain. Validating complex systems or models by exhaustive testing is not feasible due to the large number of variable combinations. However, Latin hypercube sampling (LHS) can identify critical combinations of variables for testing. Nearly orthogonal Latin hypercube sampling (NOLHS) (Cioppa, 2002) is an extension that, at high dimensions, results in a lower average distance between sample points and is computationally less costly. Early components of this dissertation considered adapting these techniques to problem space sampling (Holder, 2008). As a basis for initial sampling, schemes based on hypercube sampling (McKay et al., 2000; Ye et al., 1998; Cioppa, 2002) or stratified sampling variants (McKay et al., 2000; Kwok et al., 2006) are relevant. Following an initial sample, a biased sampling scheme like exponential sampling (Holder et al., 2006), in which samples become closer to each other in a geometric progression as they get closer to a target location, would assist with more thoroughly exploring areas of interest.

Instead of calculating the complete set of samples at one time, another approach is to start from a single point and stochastically expand. Rapidly exploring Random Trees (RRT) explore a space by branching out from an initial location, with a bias towards unexplored subregions. Unmodified, a RRT explores a space in a uniform manner. However, work such as bi-directional RRT (LaValle and Kuffner, 2001), Rapidly Exploring Evolutionary Trees (RET) (Martin et al., 2007), Extended Rapidly Exploring Random Trees (ERRT) (Bruce and Veloso, 2002), and other variants (Zucker et al., 2002; Ferguson et al., 2006) demonstrate biasing the tree growth towards areas of interest, even in a potentially changing environment.