APPROVAL SHEET

Title of Thesis: Rapid Plan Adaptation Through Offline Analysis of Potential Plan Disruptors

Name of Candidate: Robert H. Holder, III
Ph.D. in Computer Science,
2015

Thesis and Abstract Approved: _____
Dr. Marie desJardins
Professor
Department of Computer Science and
Electrical Engineering

Thesis and Abstract Approved: _____
Dr. Timothy Finin
Professor
Department of Computer Science and
Electrical Engineering

Date Approved: December 9, 2015 _____

Curriculum Vitae

Name: Robert H. Holder, III.

Degree and date to be conferred: Ph.D. in Computer Science, Dec 2015.

Collegiate institutions attended:

University of Maryland Baltimore County, Ph.D. Computer Science, 2015.
Tulane University, M.S. Computer Science, 2000.
Tulane University, B.S. Computer Science, 1999.

Professional publications:

B. Bauer, D. Scheidt, P. Rosendall, N. Rolander, **R. Holder**, E. Schmidt, and F. Ferrese. "Evaluating Test Methods for a Complex Connected System." American Institute of Aeronautics and Astronautics, March 2011.

R. Holder. "Problem Space Analysis for Plan Library Generation and Algorithm Selection in Real-time Systems." In Proceedings of The 23rd International FLAIRS Conference, May 2010, Daytona Beach, FL. **Best Poster Award.**

R. Holder and R.S. Cost. "Utilizing Resource Brokering Within Virtual Environments to Support Distributed Collaboration and Rapid Team Configuration." Proceedings of the 14th ICCRTS, June 2009, Washington, D.C.

R. Holder and R.S. Cost. "Challenges of Resource Integration for C2 Collaborations within Virtual Environments." Proceedings of 2008 Proteus Futures Academic Workshop, Sept 2008, Carlisle, PA.

Professional positions held:

Computer Scientist, Johns Hopkins University Applied Physics Laboratory (August 2000 – December 2014).
Senior Software Engineer, ClearEdge IT Solutions, LLC (January 2015 – Current).

ABSTRACT

Title of Thesis: Rapid Plan Adaptation Through Offline Analysis of Potential Plan Disruptors

Robert H. Holder, III, Ph.D. in Computer Science, 2015

Thesis directed by: Dr. Marie desJardins, Professor and
Dr. Tim Finin, Professor
Department of Computer Science and
Electrical Engineering

Computing solutions to intractable planning problems is particularly problematic in dynamic, real-time domains. For example, visitation planning problems, such as a delivery truck that must deliver packages to various locations, can be mapped to a Traveling Salesman Problem (TSP). The TSP is an NP-complete problem, requiring planners to use heuristics to find solutions to any significantly large problem instance, and can require a significant amount of time. Planners that solve the dynamic variant, the Dynamic Traveling Salesman Problem (DTSP), calculate an efficient route to visit a set of potentially changing locations (Psaraftis, 1988). When a new location becomes known, DTSP planners typically use heuristics to add the new locations to the previously computed route. Depending on the placement and quantity of these new locations, the efficiency of this adapted, approximated solution can vary significantly (Psaraftis, 1995; Laporte et al., 2000; Larsen, 2000). Solving a DTSP in real time thus requires choosing between a TSP planner, which produces a relatively good but slowly generated solution, and a DTSP planner, which produces a less optimal solution relatively quickly.

Instead of quickly generating approximate solutions or slowly generating better solutions at runtime, this dissertation introduces an alternate approach of precomputing a library of high-quality solutions *prior* to runtime. One could imagine a library containing

a high-quality solution for every potential problem instance consisting of potential new locations, but this approach obviously does not scale with increasing problem complexity. Because complex domains preclude creating a comprehensive library, I instead choose a subset of all possible plans to include. Strategic plan selection will ensure that the library contains appropriate plans for future scenarios.

Experimental results demonstrate that plan quality comparable to online repair can be achieved by calculating solutions for a sample of the potential problem instances. I present novel algorithms that accept the solution sample and then approximate solutions to other problem instances by exploiting structure in the solution space. For domains with solution spaces that do not contain sufficient structure, I show that applying abstraction, normalization, and reindexing operations to the solutions can create the necessary structure. For the domains tested, the algorithms generated full plan libraries containing solutions as good or better than online repair by calculating solutions to as few as 0.2% of the potential problem instances.

This dissertation thus contributes (1) a representation framework to reason about the structure of solution spaces, (2) novel algorithms to exploit structure in the solution space in order to generate plan libraries, (3) techniques to transform the structure of the solution space to facilitate the use of the algorithms, and (4) an evaluation of the algorithms in several test domains.

Rapid Plan Adaptation Through Offline Analysis of Potential Plan Disruptors

by

Robert H. Holder, III

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science
2015

Dedicated to (soon-to-be-Dr.) Michelle Beadle Holder.

ACKNOWLEDGMENTS

I would like to thank Dr. Tim Finin and Dr. Marie desJardins for their help and patience in helping me complete this process on a part-time basis since 2004. Appreciation is also extended to my committee, Dr. Tim Oates, Dr. Scott Cost, and Dr. Donald Miner for their feedback and flexibility.

I also would like to thank the PROMISE program, directed by Dr. Renetta Tull, for the amazing amount of support supplied over the course of completing this degree.

Thanks also are extended to the Johns Hopkins University Applied Physics Laboratory for its support of my continuing education. Many individuals from the Lab were very supportive in this endeavor. Rose Daley introduced me to this problem area as part of a DARPA project. Dr. I-Jeng Chang, Dr. Scott Cost, John Piorkowski, Dr. Ralph Semmel, Dr. David Silberberg, Dr. Paul McNamee, and David Watson were some of many individuals who guided me in various ways.

Various fellow graduate students were also instrumental in accomplishing this goal. In particular, I would like to mention Dr. Blazej Bulka, Dr. Eric Eaton, Dr. James MacGlashan, Dr. Patti Ordóñez Rozo, and Dr. Donald Miner.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

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, a planning domain consists of a set of states, described by state variables; a set of available actions, described by their effects 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 for environments in which the planner has a limited amount of time 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 of 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 heuristic solver. Since a heuristic solver creates an approximate solution, the result cannot be assumed to be optimal. Thus, 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 Problem-Solution (PS) Map, and describes several novel approximation approaches in order to create the plan library. I note how the solution spaces of a domain can have homogeneous regions that can be exploited to efficiently find solutions to a large number of problem instances. The most promising algorithms are those that are able to quickly find the borders of the solutions regions.

I then demonstrate this approach's applicability to multiple domains through experiments in Chapter 4. These experiments illustrate that good approximate PS Maps can be obtained from a small number of samples in the problem space. I also demonstrate how creating abstract solutions allows these algorithms to be utilized in a domain in which the

solutions do not form homogeneous regions.

This dissertation also briefly examines practical tradeoffs between online and offline planning time in Chapter 5. This includes some timing results and thoughts on choosing a sample rate and the appropriate algorithm. This chapter also revisits the issue of irregular solutions spaces, and discusses reindexing a solution space as a technique to facilitate the use of PS Map approximation algorithms.

Finally, Chapter 6 suggests extensions to this work and concludes the dissertation.

Chapter 2

BACKGROUND & RELATED WORK

My work primarily focuses on developing a plan library for future use as 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 in this chapter present related work in the 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, preemptively store plans in order to react quickly to new information. One classic approach is Schoppers' 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 to provide exactly this capability.

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 whether a solution is suitable for a problem instance ("case" in CBR vernacular), and uses this information to evaluate the library's competence. This process can also be used to re-

duce 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 work (McSherry, 2003) 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 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 that 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 depend on the 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.

The limitations of Onder and Pollack's research include the need to enumerate all of 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) introduce *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 the goal. This method is distinct from my approach, which always generates plans that are specifically tailored for the goals in the new environment. Also, the plans that I generate are stored as separate plans in a library rather than being incorporated into an existing plan.

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, storing the choices efficiently by recording a baseline plan, and then representing additional plans as differences from the baseline plan. This allows more rapid traversal of plans during the selection process.

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, a planner employing this scheme will execute a plan until the environment changes, effectively creating a new problem instance. It will then modify, or *repair*, the existing plan until it is applicable to the new problem instance. This

process 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 a plan repair system is the SALIX planner (Logan and Poli, 1997), which 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-indepedent solution by van der Krogt and de Weerdt (2005) presents a framework that intends to encompass a variety 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. Unrefinement is followed by the *refinement* stage, in which 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 candidate satisfies the problem requirements. The online repair baseline for the final test domain in this dissertation 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 that are 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 my

approach focuses on storing complete plans rather than 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 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, my approach varies the problem instance to map the relationship between a problem instance and a problem solution within a set of problem instances. Thus, my 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 my 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 mine in terms of indexing of plans. However, my 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 dissertation's suggested future work is to decompose a homogeneous planning problem in a general fashion, matching the problem instance to an appropriate algorithm chosen from a set of options.

Domshla et al. (2010) seek 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 expanded, regardless of the results, 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. 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 search, the heuristic is chosen by computing the tradeoff between each heuristic's computation time and expected accuracy. This approach achieves better results than the use of either individual heuristic. Their approach is analogous to mine in that they explicitly define an ideal map that 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 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 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 sampling techniques – random sampling and active learning – are described below, along with 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 number 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, an 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.

Chapter 3

PROBLEM SPACE ANALYSIS

I use the term *problem space analyais* (PSA) to describe methods that attempt to estimate the solutions for a large number of problem instances by analyzing patterns of solutions of a small number of problem instances. In many domains, problem instances that are adjacent when indexed by their *variable features* tend to have the same or similar solutions. This chapter describes seven PSA algorithms for plan adaptation and presents a complexity analysis in the final section.

A graphical rendering of a Problem Solution (PS) Map for a set of small Traveling Salesman Problems (TSPs) is shown in Figure 3.1. The static characteristics are the x- and y-coordinates of four destinations that are common to all the problem instances (i.e., that the initial plan solution uses), plus the location of the start of the path (at the central solid circle). The coordinates of the destinations are (10,10), (20,30), (5,35), and (35,25). The variable features of the problem instances are the x and y coordinates of a fifth destination, that could be added to the route as a dynamic change that requires plan adaptation. The ranges of these latter features – the x and y coordinates of the added fifth destination – are represented by the axes of the PS Map. At each location in the map, the shortest route for the new five-city problem is generated as the solution. Finally, each unique solution, consisting of a sequence of city identifiers, is assigned a color and plotted. For example,

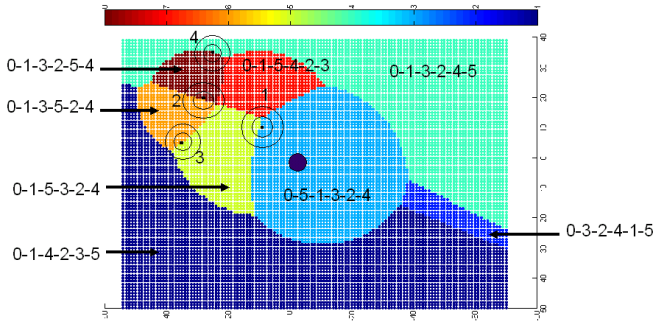

FIG. 3.1: Problem-Solution Map for a 5-city TSP. Hollow circles represent the locations of the four static city locations, and the axes represent the x and y coordinates of possible locations of the fifth city. The map shows eight unique high-quality solutions for all possible problem instances at the given granularity. (Best viewed in color.)

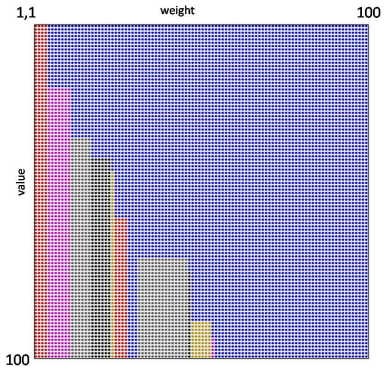

FIG. 3.2: Problem-Solution Map for knapsack problem. Axes represent the possible weight and value characteristics of one additional item that the planner may add to the knapsack. The map shows 11 unique high-quality solutions for all possible problem instances, for objects in the integral weight and value range [1,100]. (Best viewed in color.)

(20,10) represents a problem instance in which the fifth city is located at (20,10), and has a shortest path solution of 0-5-1-3-2-4. Proceeding in this fashion results in a mapping between each DTSP problem instance and the solution representing the shortest route.

As another example, a PS Map for a set of 0-1 Knapsack Problems is depicted in Figure 3.2. In this case, the problem instances' static set of characteristics is a set of 22 items, each with a weight and value. The problem instances have two variable features, consisting of the weight and value of an additional item, which are used as the axes of the PS Map. The solution is the set of items that maximizes the total value of the knapsack, while constraining the total weight to less than a fixed quantity.

The dimensions of the PS Maps are represented as ordinal domains, which requires the ability to enumerate the values of each dimension. Planning problems containing dimensions with discrete domains must define an ordering of the values and nearness metric that defines how "close" any two values are. For example, a "color" dimension with domain {red, green, blue} must define a strict ordering and nearness metric in order to be used by the algorithms described here. Dimensions consisting of real values must define a granularity to be used within the algorithms.

There is also an implicit assumption that similar problem instances have similar solutions when indexed by the problem characteristics. If this assumption holds, then problem instances with similar solutions will appear in homogeneous groups within the PS Map, which is the feature that these algorithms exploit. In the case that a domain does not adhere to this assumption, there are methods, analogous to the SVM kernel trick, that may allow for my algorithms to be applied. These ideas are discussed in Chapter 6 as future work.

As previously mentioned, it is impractical to generate a high-quality PS Map through brute-force mechanisms. For example, finding a high-quality map for a problem space with four fixed and one variable city, consisting of 12,000 problem instances, can be generated in a few seconds with my current implementation on a standard laptop. However, the

PS Map for the same problem with two variable cities requires solving $12{,}000^2$ problem instances, which would take approximately 17 hours to complete. Adding more dimensions of variability increases the size of the PS Map exponentially. Since real-world problems can have many more dimensions and problem instances than in these experiments, it is imperative to develop efficient approaches for creating approximate PS Maps.

I present seven novel techniques for creating PS Map approximations. All seven methods begin with generating high-quality solutions to a random sample of problem instances, computed using heuristic search. The *sampling-classification* (SC) and *sampling-classification with bias* (SC+bias) techniques use the solved problems and their solutions as a training set to classify new problem instances into one of the solutions discovered during the initial sampling. The former uses random selection to select the initial problem instances for solving. The latter attempts to bias the initial random selection towards problem instances that are close to the borders between solution regions. The *solution border estimation* (SBE) technique uses the heuristic search objective function and the solutions of the sampled instances to estimate where the boundaries between solution clusters lie. The *select from sampled solutions* (SSS) technique applies each known solution to an unsolved problem instance and assigns the solution with the best utility. The *sampling-classification with active learning* (SC+AL) technique attempts to bias computational time towards solving problem instances that are potentially ambiguous. The *support vector machine* (SVM) technique utilizes a support vector machine (SVM) to classify problem instances into solutions. The *support vector machine with solution border estimation* (SVM+SBE) technique also utilizes an SVM, but augments the training samples by finding problem instances near the borders of solution regions. These methods are described in more detail below. This chapter then concludes with a complexity analysis of the algorithms.

Algorithm 1 Sampling-Classification

Let $alpha \in (0.0, 1.0)$
Let $sampleRate \in (0.0, 1.0)$
Let $problemSpace \leftarrow$ set of problem instances
Let $pollingRadius \in \mathbb{Z}^+$
$totalNumSamples \leftarrow |problemSpace| * sampleRate$
for $1 \ldots totalNumSamples$ **do**
    Randomly select unsolved problem instance
    Generate solution for unsolved problem instance
    Add problem instance & solution to PS Map
end for
for all $u \in$ unsolved problem instances **do**
    Let $rad \leftarrow pollingRadius$
    while u is unsolved & $rad <$ *radiusOf*$(problemSpace)$ **do**
        Score solutions of problem instances within rad of u
        if there is a solution with a unique maximum score **then**
            Assign solution to u
        else
            $rad \leftarrow rad * 2$
        end if
    end while
    if there does not exist a unique solution with the maximum score **then**
        Randomly choose one of the top solutions
    end if
    Add problem instance & solution to set of pending entries
end for
Add pending entries to PS Map

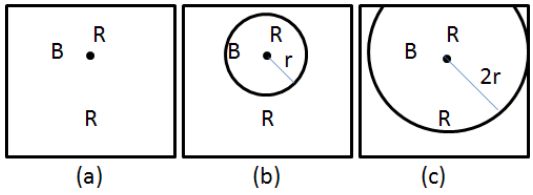

FIG. 3.3: SC procedure. The dot represents the problem instance for which a solution will be inferred, and the solutions to sampled problem instances are represented by letters.

3.1 Sampling-Classification (SC)

The *sampling-classification* (SC) technique (Algorithm 1) computes solutions to a random sample of the problem instances, then uses an expanding fixed-radius neighbor classification to assign solutions to the remaining (unsolved) problem instances. Figure 3.3 illustrates the steps involved. First, an initial random sample of solutions is solved by the heuristic solver (a). Next, solutions to unsolved problem instances are assigned by polling the solutions of the sampled problem instances within a specified radius (b). Finally, if the polling does not result in a plurality, then the polling radius is doubled until a plurality is achieved (c). Polling does not include inferred solutions; each unsolved instance is assigned a solution based solely on the solutions to the original sample of problem instances. Thus, the order in which the instances are solved does not affect the configuration of the resulting PS Map.

3.2 Sampling-Classification with Bias (SC+bias)

The *sampling-classification with bias* (SC+bias) technique, described in Algorithm 2, attempts to exploit the observation that certain constraints – for example, known city locations – indicate boundaries between solutions. The proof in Figure 3.4 demonstrates that static city locations must lie on a border between two solutions. Thus in SC+bias,

Let $F = \{f_1, f_2, \ldots, f_n\}$ be a set of fixed points, o be the start of the tour, and p be a variable point. An optimal solution to the TSP problem is a sequence $S = \{s_1, \ldots, s_{n+2}\}$ of the points $F \cup \{p, o\}$ such that $\sum_{i=1}^{n+1} dist(s_i, s_{i+1})$ is minimized.
Consider two solutions, $S_1 = \{o, \ldots, s_a, s_b, p, s_c, \ldots, s_{n+2}\}$ and $S_2 = \{o, \ldots, s_a, p, s_b, s_c \ldots, s_{n+2}\}$, differing only in the order in which p and s_b are visited. Without loss of generality, let s_b be any static city location. The border between S_1 and S_2 is the set of points where S_1 and S_2 have equal quality, which are the points that satisfy:

$$\begin{aligned} dist(o, s_1) + dist(s_2, s_3) + \cdots + dist(s_{a-1}, s_a) + \\ dist(s_a, s_b) + dist(s_b, p) + dist(p, s_c) + \\ dist(s_c, s_{c+1}) + \cdots + dist(s_{n+1}, s_{n+2}) = \\ dist(o, s_1) + dist(s_2, s_3) + \cdots + dist(s_{a-1}, s_a) + \\ dist(s_a, p) + dist(p, s_b) + dist(s_b, s_c) + \\ dist(s_c, s_{c+1}) + \cdots + dist(s_{n+1}, s_{n+2}). \end{aligned}$$

Reducing, we obtain

$$dist(s_a, s_b) + dist(p, s_c) = dist(s_a, p) + dist(s_b, s_c).$$

Substituting the known point s_b for the variable point p results in a valid equation. Therefore, s_b is on the border between S_1 and S_2.

FIG. 3.4: Proof that static cities in the DTSP must lie on a border between two solution regions

Let A-p-B and A-B-p represent the routes specified by two solutions. To find the shape of the border, we set the distances of the routes to be equal.

$$\begin{aligned} dist(p, A) + dist(p, B) &= dist(A, B) + dist(p, B) \\ dist(p, A) &= dist(A, B) \\ \sqrt{(p_x - A_x)^2 + (p_y - A_y)^2} &= dist(A, B) \\ (p_x - A_x)^2 + (p_y - A_y)^2 &= dist(A, B)^2 \end{aligned}$$

FIG. 3.5: The border between solutions A-p-B and A-B-p simplifies to a circle

Algorithm 2 Sampling-Classification+Bias

Let $alpha \in (0.0, 1.0)$
Let $sampleRate \in (0.0, 1.0)$
Let $problemSpace \leftarrow$ set of problem instances
Let $pollingRadius \in \mathbb{Z}^+$
Let $biasFactor \in \mathbb{Z}^+$
Let $cityRadius \in \mathbb{Z}^+$
$totalNumSamples \leftarrow |problemSpace| * sampleRate$
$numNearSamples \leftarrow \frac{biasFactor * totalNumSamples}{1 + biasFactor}$
for $1 \ldots numNearSamples$ **do**
    Randomly select unsolved problem instance within $cityRadius$ of city
    Generate solution for unsolved problem instance
    Add problem instance & solution to PS Map
end for
for $numNearSamples + 1 \ldots totalNumSamples$ **do**
    Randomly select unsolved problem instance outside of $cityRadius$ of city
    Generate solution for unsolved problem instance
    Add problem instance & solution to PS Map
end for
for all $u \in$ unsolved problem instances **do**
    Let $rad \leftarrow pollingRadius$
    while u is unsolved & $rad < radiusOf(problemSpace)$ **do**
        Score solutions of problem instances within rad of u
        if there exists a unique solution with the maximum score **then**
            Assign solution to u
        else
            $rad \leftarrow rad * 2$
        end if
    end while
    if there does not exist a unique solution with the maximum score **then**
        Randomly choose one of the top solutions
    end if
    Add problem instance & solution to set of pending entries
end for
Add pending entries to PS Map

Let A-B-C-p-D and A-p-B-C-D represent the routes specified by two solutions. To find the shape of the border, we set the distances of the routes to be equal.

$$dist(A,B) + dist(B,C) + dist(p,C) + dist(p,D) = dist(p,A) + dist(p,B) + dist(B,C) + dist(C,D)$$

$$dist(p,A) - dist(p,B) + dist(p,C) - dist(p,D) = dist(B,C) + dist(C,D) - dist(A,B)$$

FIG. 3.6: The border between solutions A-B-C-p-D and A-p-B-C-D has a non-trivial simplification

Algorithm 3 Sampling-Classification + Active Learning

Let $alpha \in (0.0, 1.0)$
Let $sampleRate \in (0.0, 1.0)$
Let $problemSpace \leftarrow$ set of problem instances
Let $pollingRadius \in \mathbb{Z}^+$
Let $landslide \in \mathbb{Z}^+$
$totalNumSamples \leftarrow |problemSpace| * sampleRate$
$numInitialSamples \leftarrow totalNumSamples * alpha$
$usedSamples \leftarrow numInitialSamples$
for $1 \ldots numInitialSamples$ **do**
    Randomly select unsolved problem instance
    Generate solution for unsolved problem instance
    Add problem instance & solution to PS Map
end for
for all $u \in$ unsolved problem instances **do**
    Let $V \leftarrow$ solutions of problem instances within $pollingRadius$ of u ordered by decreasing count
    if $|V| = 1$ **then** //there is only one solution
        Assign solution to u
    else if $\frac{count(V_0)}{count(V_1)} \geq landslide$ **then** //highest score divided by second-highest
        Assign V_0 to u
    else if $usedSamples < totalNumSamples$ **then**
        Solve u and assign solution
    else
        Expand radius and assign solution as with Sampling-Classification
    end if
end for

the problem instance samples are biased towards the known city locations in the hope that additional samples in these regions will allow the classification step to discover the borders between solutions with greater accuracy. After gathering the additional samples, this technique assigns solutions to unsolved instances in the same manner as the SC technique.

This technique relies on two additional parameters. The *city radius* parameter is a radius defining a pool of problem instances that are "near" a city location; instances outside this radius are considered not to be near the city. The *bias factor* parameter defines the ratio of the number of near city points to the number of non-near city points selected in the initial random sample. For example, a bias factor of three indicates that three times as many near-city points as non-near points will be selected.

The use of this technique is specific to TSP problems. Although there is consideration for taking into account specific constraints of static problem characteristics to inform sampling bias, it is not clear how this applies in a general case. Thus, this technique was tested only on the TSP domain.

3.3 Sampling-Classification with Active Learning (SC+AL)

The *sampling-classification with active learning* (SC+AL) algorithm modifies the SC (sampling and classification) technique to utilize active sampling rather than random sampling to select problem instances to solve (Algorithm 3). This algorithm adds two paramters, *alpha* and *landslide*. The alpha parameter represents the fraction of the total number of problem instances that will be selected through random sampling. The landslide threshold is used to determine whether the voting by the nearest neighbors is ambiguous. For example, a sample rate of .01 in a problem space with 10,000 instances results in a total of 100 samples. Assuming an alpha of 0.2, an initial random sampling of 20 problem instances will be solved. The remaining 80 samples will be chosen after evaluating the

Algorithm 4 Solution Border Estimation - trace

Let $sampleRate \in (0.0, 1.0)$
Let $problemSpace \leftarrow$ set of problem instances
$totalNumSamples \leftarrow |problemSpace| * sampleRate$
for $1 \ldots totalNumSamples$ **do**
    Randomly select unsolved problem instance
    Generate solution for unsolved problem instance
    Add problem instance & solution to PS Map
end for
$borderSet \leftarrow \emptyset$
for each pair of problem instances p, q with differing solutions s_p, s_q **do**
    use binary search to find pair of adjacent problem instances with differing solutions
    $border \leftarrow$ DoTrace($p, s_p, s_q, \emptyset$)
    Add $border$ to $borderSet$
end for
find intersections of borders to determine regions
for all region r **do**
    find problem instance to serve as regional representative
    find best solution for this problem instance
    assign solution to all problem instances in the region
end for
function DOTRACE($instance$,$solution$,$altSolution$,$border$)
    add $instance$ to $border$
    for all problem instance p_a adjacent to $instance$ **do**
        if p_a is adjacent to a problem instance with where $altSolution$ is better than $solution$ **then**
            add p_a to $border$
            DoTrace(p_a,$solution$,$altSolution$,$border$)
        end if
    end for
    return $border$
end function

fixed radius neighbors of unsolved problem instances. If the fixed-radius neighbors of an unsolved problem instance indicate little or no ambiguity when approximating its solution, then the problem instance is assigned a solution as in SC, by a plurality vote. However, if the fixed-radius neighbors do indicate ambiguity, then the problem instance is solved heuristically if the total allocation of problem instance samples has not been exhausted. In this algorithm, ambiguity refers to either zero fixed-radius neighbors, or more than one fixed-radius neighbor in which the number of occurences of the best solution divided by the number of occurences of the second-best solution does not meet the *landslide* threshold.

3.4 Solution Border Estimation (SBE)

The *solution border estimation* (SBE) technique calculates solutions to a random sample of the problem instances. Then, for every pair of solutions, SBE calculates a border in the problem space where one solution becomes better than the other. The combination of these borders creates a set of regions within the problem space. Because the borders that create the regions are determined only by a pair of solutions, there is no guarantee that some third solution is not preferable within any region. To resolve this uncertainty, the algorithm determines the best solution within a region by solving one problem instance within each region, and assigning that solution to all problem instances in the region.

Ideally, these borders would be calculated by equating the objective functions representing each solution and finding a closed-form expression for the boundary location, such as shown in Figure 3.5 for a 3-city TSP problem. However, this approach is not practical for large problems or problems that are not easily expressed with an objective function. As an example of the difficulty presented by a larger problem, consider resolving the border between 5-city TSP solutions A-B-C-p-D and A-p-B-C-D, where p is the unknown location and A, B, C, and D represent known locations. This results in a non-trivial equation in

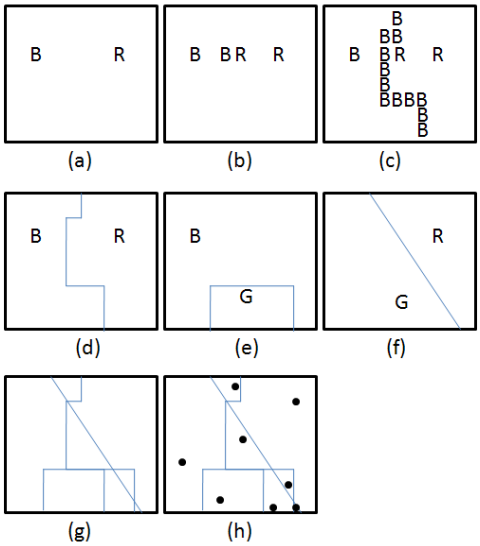

FIG. 3.7: Skeletal PS Map created by SBE-trace procedure

Figure 3.6 with four radicals (pairwise distances) and a constant. Therefore, my implementation, *SBE-trace*, uses an approximation of the SBE technique, as described in Algorithm 4.

Figure 3.7 illustrates the SBE-trace algorithm. First, two solved problem instances with differing solutions are selected (a). Next, a binary search is applied to the space between the two solved instances to find two adjacent problem instances that have different solutions (b). Then, the remainder of the border is discovered by testing neighboring points for adjacency to a problem instance with the alternate solution, forming a continuous border between the solution regions (c,d). Applying this procedure in a pairwise fashion to the remaining discovered solutions (e,f) creates an approximation of the skeletal PS Map (g). Finally, sampling within each region yields an approximate PS Map (h).

3.5 Support Vector Machine (SVM)

The *support vector machine* approach, described in Algorithm 5, utilizes a support vector machine (Cortes and Vapnik, 1995) to classify unsolved problem instances into classes consisting of known solutions. A support vector machine classifies inputs into one of two classes by calculating a hyperplane that splits the input space into two regions, one for each class, that lies as far as possible from any input instance. An advantage of this classifier is that it scales to high-dimensional spaces. SVMs employ a "kernel trick" that allows them to calculate a hyperplane when the inputs are not linearly separable, as is typically the case in the plan spaces that I have studied.

In this algorithm, an initial sample of problem instances are solved to generate solutions, as in the SC technique. I train the SVM with the problem instances' variable characteristics and the high-quality solution generated by the heuristic solver. After training, the unsolved instances are assigned solutions based on the SVM's classifications.

Algorithm 5 Support Vector Machine

Let $sampleRate \in (0.0, 1.0)$
Let $problemSpace \leftarrow$ set of problem instances
$totalNumSamples \leftarrow |problemSpace| * sampleRate$
for $1 \ldots totalNumSamples$ **do**
    Randomly select unsolved problem instance
    Generate solution for unsolved problem instance
    Add problem instance & solution to PS Map
    Add problem instance features & solution to SVM training set
end for
Train SVM
for all unsolved problem instances **do**
    Add problem instance and SVM classification to PS Map
end for

Algorithm 6 Support Vector Machine + Solution Border Estimation

Let $alpha \in (0.0, 1.0)$
Let $sampleRate \in (0.0, 1.0)$
Let $problemSpace \leftarrow$ set of problem instances
$totalNumSamples \leftarrow |problemSpace| * sampleRate$
$numInitialSamples \leftarrow totalNumSamples * alpha$
for $1 \ldots numInitialSamples$ **do**
    Randomly select unsolved problem instance
    Generate solution for unsolved problem instance
    Add problem instance & solution to PS Map
    Add problem instance features & solution to SVM training set
end for
for each pair of problem instances with differing solutions s_p, s_q **do**
    use binary search to find pair of adjacent problem instances with differing solutions
    Add pair of problem instances and their solutions to SVM training set
end for
Train SVM
for all unsolved problem instances **do**
    Add problem instance and SVM classification to PS Map
end for

Algorithm 7 Select from Sampled Solutions

Let $alpha \in (0.0, 1.0)$
Let $sampleRate \in (0.0, 1.0)$
Let $problemSpace \leftarrow$ set of problem instances
Let $pollingRadius \in \mathbb{Z}^+$
$totalNumSamples \leftarrow |problemSpace| * sampleRate$
for $1 \ldots totalNumSamples$ **do**
    Randomly select unsolved problem instance
    Generate solution for unsolved problem instance
    Add problem instance & solution to PS Map
end for
for all $u \in$ unsolved problem instances **do**
    Generate utility of u for each known solution in PS Map
    if there does not exist a unique solution with the maximum score **then**
        Randomly choose one of the top solutions
    end if
    Add problem instance & solution to set of pending entries
end for
Add pending entries to PS Map

3.6 Support Vector Machine with Solution Border Estimation (SVM+SBE)

The *support vector machine with solution border estimation* (SVM+SBE) technique (Algorithm 6) utilizes a fraction of the total allocated samples to create an initial sample of problem instances from which to generate a set of known solutions. For each combination of pairwise problem instances that have different solutions, the SBE technique is used to find a pair of problem instances that lie on the border between the two solutions. These border points and their solutions are added to the SVM training set. Finally, the unsolved problem instances are assigned solutions as dictated by the SVM results.

3.7 Select from Sampled Solutions (SSS)

The *select from sampled solutions* (SSS) technique calculates solutions to a random sample of the problem instances. This technique assigns solutions to each unsolved problem instance by computing the utility of each of the discovered solutions when applied to the unsolved instance, and assigning the maximum-utility solution. In the case of continuous objective functions, which creates large homogeneous solution regions, this process generates a PS Map identical to that of SBE-trace. However, because it must determine the maximum utility solution for every problem instance in the space, this algorithm risks performance degradation as the problem size increases. For example, for a map representing a DTSP with two variable cities consisting of $12{,}000^2$ problem instances, SSS would entail evaluating every unknown solution for every problem instance. However, in situations in which the number of stored solutions is small or the cost of calculating the utility is cheap enough, the discovered solutions could be stored, rather than a complete map. This could mitigate the disadvantage that SSS may encounter relative to SBE-trace.

One interesting feature of this algorithm is that it can be used to remove errors in ideal maps caused by the use of heuristics when solving problem instances. Heuristic

solvers may assign different solutions to differing problem instances that in fact do have identical solutions. The application of this algorithm can mitigate this type of error by considering all the discovered solutions within the problem space. Visually, this has the effect of "smoothing" the solution regions into more regular shapes within the TSP and knapsack problem domains.

3.8 Algorithm Analysis

I have analyzed these algorithms primarily in terms of the number of problem instances that must be resolved by the heuristic solver. Solving a problem instance with the heuristic solver takes the highest amount of time for a single problem instance; however, many of the solutions involve less expensive operations over a large number of problem instances and thus the heuristic solve time cannot be assumed to dominate the complexity expresssion.

Let H represent the time complexity required to solve a single problem instance with a heuristic solver, and let s represent the sample rate. P will represent the size of the problem space and K will represent the complexity of the fixed-radius neighbor search. A brute-force fixed-radius neighbor search is $O(n)$ in the number of candidate neighbors. However, other approaches can be appropriate depending on the number of candidate neighbors, which varies as function of the sample size. To accommodate this variability, the final complexities listed in Table 3.1 present the complexities using a generic K for the fixed-radius neighbor search as well as assuming a worst-case complexity of $O(n)$.

SC Algorithm For the SC algorithm, the initial loop samples the complete problem space and solves an initial sample of problem instances. This complexity is $O(HsP)$. Next, the remaining $(1-s)P$ problem instances must be solved. For each instance, the algorithm runs the fixed-radius neighbor search repeatedly until either there is a plurality of solutions

within SC's expanding radius, or the radius encompasses the complete problem space. Since the radius doubles with each iteration, the maximum number of iterations possible per problem instance is $log_2 P$. Thus the complexity for SC is $O(HsP + (1-s)PKlogP)$, where K is the complexity of fixed-radius neighbor search. The intent of these algorithms is for s to be small, particularly when H is large. Therefore, it is not clear whether the first term, which is a product of a large and small number, dominates or is dominated by the second term, which is a product of a number near one, the size of the problem space, its log, and the complexity of the fixed-radius neighbor search.

SC+Bias Algorithm The SC+Bias algorithm is identical to SC except that the sampling is biased to be closer to specific locations in the problem space. There is some expense to identify the set of problem instances that are within the radius of a city, but this one-time cost, amortized over repeated runs, is negligible. Thus the calculations are the same as the SC algorithm, resulting in the complexity of $O(HsP + (1-s)PKlogP)$.

SC+AL Algorithm The complexity of the SC+AL algorithm must consider the alpha parameter that determines the initial fraction of problem instances to be solved heuristically through random sampling. The complexity of this step is $O(Hsp\alpha)$. The remaining problem instances to be solved heuristically are determined by the utility of the solutions discovered when polling within the radius of the problem instance. The complexity of this step is $O(Hsp(1-\alpha))$. Combining these two terms, $HsP\alpha$ + $HsP(1-\alpha)$, simplifies to the same initial term as the previous algorithms, HsP. The cost of solving the remaining $(1-s)P$ problem instances is, in the worst case, the cost of expanding the polling radius as in SC. This results in a total complexity of $O(HsP + (1-s)PKlogP)$, again the same as the SC algorithm.

SBE-trace Algorithm The SBE-trace algorithm is limited to two dimensions, which is used to simplify its complexity analysis. As with the previous algorithms, the initial sampling is again of complexity $O(HsP)$. The loop starting at line 10 runs for each pairwise combination of solutions for a total of $n(n-1)$ iterations, where n is the number of solutions. The number of solutions is the result of solving the initial sample of problem instances. Thus, n is equal to sP and the loop executes $sP(sP-1)$ times.

Each loop iteration executes a binary search that may in the worst case span the problem space and therefore has a complexity of $logP$. Each iteration also executes the DOTRACE function, which executes a loop that considers the seven adjacent problem instances to a given instance. The recursion then continues the evaluation for the length a complete border, which is at worst the size of the problem space. Thus, the complexity of the function is $7P$. As mentioned above, these two operations execute $sP(sP-1)$ times, for a total complexity of $sP(sP-1)(logP+7P)$, simplifying to $O(s^2P^3)$.

The process at line 15 of finding the points at the intersections of borders requires looping through each pairwise set of borders to find points that exist in both borders. This requires $B(B-1)$ loop iterations, where B is the number of borders, for a complexity of $O(B^2)$. Recalling that the number of borders is $O((sP)^2)$ and that a border may at most contain P points, the overall complexity of this operation is $O(((sP)^2)^2 \times P)$, which simplifies to $O(s^4P^5)$.

The final loop at line 16 requires selecting a solution for one problem instance in each region resulting in a complexity of $O(sPR)$, where sP is the number of solutions to evaluate, and R is the number of regions generated by the border intersections. In two-dimensional spaces, the number of regions generated by dividing a space with n lines or circles is $O(n^2)$. Intuitively, this can be demonstrated by observing that the i^{th} line that divides a space adds at most i regions to the space, thereby creating $\sum_{i=1}^{n} i = \frac{n(n-1)}{2}$ regions for a complexity of $O(n^2)$. Replacing n with the number of borders, $O((sP)^2)$,

and substituting for R, the complexity of this loop is $O(sP \times s^4P^4) = O(s^5P^5)$.

The sum of all of these terms is $O(HsP + s^2P^3 + s^4P^5 + s^5P^5)$. The fourth term dominates the second and third terms, simplifying to $O(HsP + s^5P^5)$. As before, it is not clear which, if either, of the terms dominates the expression, and thus both of them are preserved.

SVM Algorithm The SVM and SVM+SBE algorithms rely heavily on support machine training algorithms, which has a generally accepted upper bound of $O(n^3)$ in the number of training instances (Bottou and Lin, 2007; List and Simon, 2009). The complexity of the SVM algorithm is readily calculated as the sum of the complexity of sampling, training, and possibly classification: $O(HsP + (1-s)^3P^3 + (1-s)P)$. Dropping the final term because of the domination of the middle term results in an SVM algorithm complexity of $O(HsP + (1-s)^3P^3)$

SVM+SBE Algorithm The SVM+SBE algorithm complexity is similar to the SVM complexity, but uses an alpha parameter that determines the fraction of sampled instances that will be derived from solution border estimation. Because of this, the complexity of the initial sample is $O(HsP\alpha)$. The loop starting at line 12 runs a maximum of $(1-\alpha)sP$ times and has the same binary search as line 11 of the SBE - trace algorithm. Each of the border problem instances is solved with an $O(H)$-complexity heuristic search. Thus the complexity of this loop is $O((1-\alpha)P(H+logP))$. The SVM training is again $O(n^3)$ in the number of training instances for a complexity of $O(s^3P^3)$. The last loop at line 17 iterates over the $(1-s)P$ unsolved problem instances and places them in the PS Map. Summing the terms results in a complexity of $O(HsP\alpha + (1-\alpha)sP(H+logP) + s^3P^3 + (1-s)P)$, which simplifies to $O(HsP + s^3P^3)$.

Algorithm	Complexity	K = O((1-s)P)
SC	$O(HsP + (1-s)PKlogP)$	$O(HsP + (1-s)^2P^2logP)$
SC+bias	$O(HsP + (1-s)PKlogP)$	$O(HsP + (1-s)^2P^2logP)$
SC+AL	$O(HsP + (1-s)PKlogP)$	$O(HsP + (1-s)^2P^2logP)$
SBE	$O(HsP + s^5P^5)$	
SSS	$O(HsP + sP^2)$	
SVM	$O(HsP + (1-s)^3P^3)$	
SVM+SBE	$O(HsP + s^3P^3)$	

Table 3.1: Summary of PS Map approximation complexity. H is the complexity of generating a high-quality solution, s is the sample rate, P is the number of instances in the problem space, and K is the complexity of the fixed-radius neighbor search.

SSS Algorithm Finally, the SSS algorithm requires $O(HsP)$ for the initial sample and for each of the remaining $(1-s)P$ unsolved instances must evaluate each of the discovered solutions. Assuming each sample results in a unique solution, this results in the worst case, $(1-s)P \times sP$ evaluations, or $O(sP^2)$ assuming a small s. Thus, the complexity of SSS is $O(HsP + sP^2)$.

Chapter 4

EVALUATION IN TEST DOMAINS

This chapter presents the results of applying the algorithms described in the previous chapter to three test domains. I show that, in many cases, the utility loss from PS Map approximation is comparable to that of online repair. However, the performance of the approximation algorithms varies between problem domains and between different problem configurations within the same problem domain. In both cases, the differences in the size and quantity of heterogeneous regions intrinsic to the problem domain and configuration appear to be suggestive as reasons for the differences in approximation accuracy. I tested the algorithms using the traveling saleman problem (TSP), the knapsack problem, and an elevator problem. The TSP and knapsack problems are classic domains in optimization and computer science. The elevator problem is a challenge domain created for the AAAI International Planning Competition (IPC) (Coles et al., 2013). All of the problems are NP-complete or NP-hard, and quickly become intractable with complex enough problem instances.

Traditional TSP problems consist of a set of unordered locations, sometimes referred to as "cities," that must be ordered such that the length of route that traverses the set is minimized. In the dynamic variant, one or more additional locations become known after the initial ordering is computed, and must be incorporated into the route while minimizing

computation time and total route distance.

In the knapsack problem, one chooses from a set of given items, each with a weight and value characteristic, such that the total value of the knapsack is maximized and the total weight does not exceed a given weight constraint. I use the 0-1 variant, in which each item may be selected a maximum of one time. After computing an initial solution, I present one or more additional items with which the system may revise its solution.

The elevator domain defines the initial and desired locations of a set of passengers, and several elevators of varying speeds with which to transport passengers. The goal is to move all the passengers to their desired floors as cheaply as possible through efficient use of elevator movements. For my testing, I use a variant in which one or more passengers' initial location may change after the initial plan is computed.

My primary metric for evaluation is utility loss, measured as a fraction of the utility of a problem instance's high-quality[1] solution, as calculated by a heuristic solver. For example, if the total value of a high-quality knapsack solution is 100, and the solution retrieved from the approximated PS Map has a value of 95, then the utility loss for that specific solution is .05 (i.e., $\frac{100-95}{100}$). The evaluation of an approximated PS Map is the average utility loss over all of the discrete locations in the map. Thus, the evaluation for a PS Map over all problem instances is $\frac{\sum_{i \in map} heuristic_i - approx_i}{\sum_{i \in map} heuristic_i}$ where $approx_i$ and $heuristic_i$ are the utility of the solutions given by the PS Map and a heuristic planner, respectively, for a given problem instance i. Lower utility loss is preferred; the best approximated solution will have a utility loss of zero.

[1] As previously mentioned, "high-quality" refers to solutions generated by heuristic search methods. As solutions to intractable problems they cannot be guaranteed to be optimal; therefore, I avoid the use of that term.

4.1 Traveling Salesman Problem

The traveling salesman problem (TSP) is a classic NP-complete problem in computer science in which cities must be ordered such that the length of the resulting route is minimized. The dynamic variant, the DTSP, allows for cities to be removed or added while the route is being traversed, creating a more challenging problem in which the route should be reoptimized in real time. I used the TSP as an initial domain for algorithm validation and development. I generated problem instances ranging from 5 cities to 100 cities, representing a range of problem complexity. The algorithms were developed based upon the insights from tests of each preceding algorithm, which will be reflected in some of the discussion. In addition to the inter-algorithm comparison, I also generated baseline results from typical online repair techniques.

4.1.1 High-Quality PS Map

For testing in the TSP domain, I generated three instances each of 5, 10, 20, 50, and 100-city DTSPs. One of the cities included with each of the DTSPs has a variable location. As the gold standard, high-quality PS Maps were generated via the Clark-Wright (Clarke and Wright, 1964) and Gillett-Miller (Gillett and Miller, 1974) algorithms, as implemented by the Drasys library.[2] I then removed errors stemming from heuristic-based solvers by executing the SSS algorithm over the PS Map. As described in Section 3.7, this process tests each unique solution against each problem instance, resulting in a more accurate PS Map.

[2] As of this writing, this library appears to no longer be publicly available. I have placed a copy of the original download at `http://www.umbc.edu/~holder1/or124.jar`

4.1.2 Online Repair Baseline

To compare how well PS Map approximation techniques perform against traditional online repair, I implemented the insertion approach (Psaraftis, 1988). This approach incorporates new cities into the route by finding the nearest city and inserting the new city into the route either before or after the nearest city. Although it is not the best repair technique, it is well suited for online repair due to its speed. In this case, the repair accuracy was within the expected loss of utility provided by other DTSP online repair algorithms as discussed by Larsen. This baseline will be discussed in more detail when presenting the experimental results.

4.1.3 Sampling-Classification Experiment

The sampling classification (SC) algorithm is a simple algorithm used as an initial exploration of the feasibility of the general approach of using classification techniques to match problem instances with solutions. The basic implementation accepts problem instances and their solutions as input, and uses nearest neighbor-like classification to assign solutions to unsolved problem instances.

SC Experiment Parameters The PS Map approximations were generated using 19 sample rates between .0001 and .01. The experimental configurations were drawn from the permutations created by the cross product of the DTSP problem, sample rate, and approximation technique. Each run was executed ten times.

Results The initial algorithm, sampling-classification (SC), solves a random sample of the problem instances and uses classification based on nearest neighbor to assign solutions discovered during the initial sample to each unsolved problem instance. In the initial experiment, all of the solutions of solved problem instances within a static radius of an

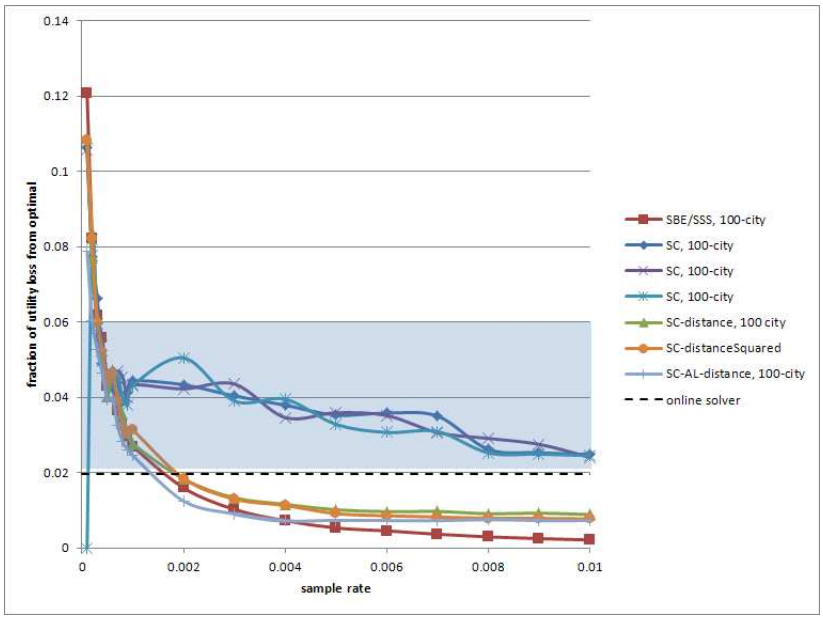

FIG. 4.1: Results of applying various approximation algorithms to the 100-city TSP domain. The dotted line represents the utility loss of the online planner. The shaded region is the expected loss of DTSP online repair algorithms as suggested by Larsen (2000).

unsolved problem instance were polled and the solution with the plurality was assigned to the unsolved problem instance. These results are included in Figure 4.1 as "SC, 100-city." Subsequent experiments weighted the solutions by the reciprocal of the distance or the distance squared, this giving more weight to the solutions of problem instances closer to the unsolved instance. These results, also in Figure 4.1, are labeled as "SC-distance" and "SC-distance squared," respectively. Figure 4.1 also includes the results of the SC experiments. In addition to showing fractional loss results, the graph highlights the range of fractional utility loss expected by online repair, as suggested by Larsen (2000). I also implemented a nearest-neighbor DTSP solver to insert the variable city into the route. The mean average loss from that online repair method was 1.97%, which is consistent with Larsen's range.

4.1.4 SC+Bias

Based on the results of the SC experiments, it became apparent that the larger solution regions tended to be represented in the approximated PS Map, but smaller regions tended to disapppear. This occurred due to the lower probability of the initial random sample choosing a problem instance in that region, resulting in either a particular region not being represented in discovered solutions or probable solutions being assigned. One attempt to mitigate this effect was inspired by observing that within the high-quality PS Map, more rapid changes in solutions and smaller solution regions tend to exist near city locations. The SC+bias algorithm attempts to take advantage of this observation by biasing samples towards the regions near cities. The city radius and bias parameters determine, respectively, the radius of the region around a city to apply the bias and how much to bias the samples. The near-city region is defined as a circle with the specified radius. The bias represents the odds that the near-city region will be sampled. For example, a bias value of three indicates that the near-city region will be sampled with odds 3:1 versus the non-near-city region.

Experiment Parameters This experiment approximated a PS Map for a 100-city DTSP. I assigned a bias factor as integers in the range from one to five, inclusive, and the city radius in the range from one to five, inclusive. The approximation algorithm was executed ten times for each combination of bias factor and city radius.

Results The results of this experiment are shown in Figures 4.2 and 4.3. These results do not appear to show an obvious pattern to determine which parameters are most promising. For example, the best performance are at the valleys (recall that lower utility loss is preferred) at bias values of one, four, and five, and radius values of two, four, and five. Looking at the graph, there is not an obvious gradient to suggest a generalized rule for setting these parameters.

4.1.5 SC+AL

Sampling classification with active learning (SC+AL) is another attempt to allow for smaller solution regions to be approximated effectively. SC+AL may be considered a generalization of SC+bias in that it allows more concentrated sampling in regions of the problem space in which the classification appears ambiguous rather than limiting the targetted samples to predetermined locations. For example, if two solutions are both strong candidates to be assigned to a specific problem instance, then SC+AL would solve the problem instance rather than risk assigning an incorrect solution. Similarly, if there are no strong candidates for a particular problem instance, then SC+AL would allow the problem instance to be solved rather than assign an arbitrary solution to it.

Experiment Parameters The alpha parameter was set to 0.5. Thus, half of the allotted problem instances solved were selected with random sampling. The other half were reserved for problen instances that the algorithm determines to be ambiguous.

Results The results of this experiment are shown in Figure 4.1. At the lower sample rates, the performance of the SC+AL algorithm appears to be slightly better than the SC results. This could suggest that at low sample rates, it is more critical to choose samples that convey the most information about the solution space. It's reasonable that as the sample rate increases, the probability increases of obtaining that same sample information through chance.

4.1.6 SSS

Experiment Parameters No algorithm-specific parameters were required for this experiment. As with the other experiments in this domain, the sample rate ranged from .0001 to .01 for a problem space consisting of 100-city DTSPs containing one variable city.

Results The results of this experiment are shown in Figure 4.1. The utility loss of SSS quickly drops, and at sample rates greater than .003 becomes the best performing algorithm. Intuitively, this seems reasonable: assuming that the initial sample discovers most solutions, then testing each of the solutions against the problem instance would result in the problem instance being assigned the optimal solution.

4.1.7 SBE

The solution border estimation algorithm (SBE) considers the mathematical features of the TSP. It calculates the border by recognizing that the border between any two solutions is represented by equating the distance functions of the two solutions. Unfortunately, at the time of this experiment, I did not find a Java library that could solve the complex equations that resulted from this technique. The SBE-trace technique is inspired by SBE; however, it finds borders between two solutions by searching the space between two problem instances with known solutions. Thus, a binary search can be employed. Assuming that the border between two solutions is continuous, then the remainder of the border can be found by comparing the utility of the two solutions at each problem instance.

Experiment Parameters No algorithm-specific parameters were required for this experiment. As with the other experiments, the sample rate ranged from .0001 to .01 for a problem space consisting of 100-city DTSPs, with one city having variable location.

Results The results of SBE-trace are shown in Figure 4.1. Note that SBE-trace is only suitable for two-dimensional PS Map approximation. Because of this limitation, it is not applicable to most domains, and thus I did not emphasize this algorithm in the subsequent experiments, which have PS Maps with higher dimenstions.

4.1.8 SVM

The support vector machine algorithm (SVM) uses a support vector machine to try to generalize the idea of SBE to multiple dimensions. Support vector machines calculate a maximum margin plane to separate different classes. The observations in this application are the sampled problem instances labeled with their solutions.

Experiment Parameters No algorithm-specific parameters were required for this experiment. As with the other experiments, the sample rate ranged from .0001 to .01 for a problem space consisting of 100-city DTSPs, with one city having variable location. I configured the SVM to use the radial basis function kernel.

Results The results of this approach are included in Figure 4.5. It demonstrates that at sample rates greater than about .01, the SVM-based algorithm performs better than the online repair baseline of fractional loss of 0.02 to 0.06 as mentioned earlier.

4.1.9 SVM+SBE

One disadvantage of the SVM-based approach is that it can misclassify problem instances. SVM determines the borders between two solution regions by calculating a hyperplane such that the gap between problem instances with differing solutions is as large as possible. This process results in a border that is approximately midway between differing solutions. SVM has been shown to be a good optimization technique in general; however, it does lead to misclassifications when the actual border does not conform to this approximation. By applying additional samples in key locations, the location of the hyperplane calculated by the SVM can be made more consistent with the actual borders. In this approach, the first step is an initial set of problem instances that are sampled and solved. The second step applies the binary search used in the SBE-trace algorithm to each distinct pair

of solutions, resulting in problem instances that represent solutions on the border between the distinct pair of solutions. Finally, those problem instances and the labeled solutions are added to the training set for the SVM.

Experiment Parameters The alpha parameter, which determines the fraction of the total allocated sample that will be used during random initial sampling, was set to 0.2 and 0.5 in separate runs. The SVM algorithm as described in the previous experiment uses all of its allocated samples during initial sampling, and is thus the equivalent of using an alpha parameter of 1.0. Choosing 0.2 and 0.5 values in this experiment results in testing of alpha values that span the most of the range of 0.0 to 1.0. I configured the SVM to use the radial basis function kernel.

Results The results of this approach are included in Figure 4.5. It is interesting to note that the performance of SVM+SBE using an alpha value of 0.2 performs better at lower sample rates, and that with an alpha rate of 0.5 performs better at higher sample rates. The crossover point is at a sample rate of approximately 0.02. It appears that at higher sample rates, the random sampling is sufficient to discover the border between solutions without targetting samples. At lower sample rates, the stucture of the solution space is not as explored, and thus it is valuable to discover key points where one solution becomes better than another. However, at lower sample rates fewer solutions are discovered. Thus, there is a tension between random sampling in order to discover the solutions that exist in the space, versus targeted sampling, which assists in accurately finding the borders between the discovered solutions. Revisiting the SVM algorithm, which is equivalent to SVM+SBE with an alpha value of 1.0, the trend continues: using fewer targeted samples results in worse performance at lower sample rates, but performs better at higher sample rates.

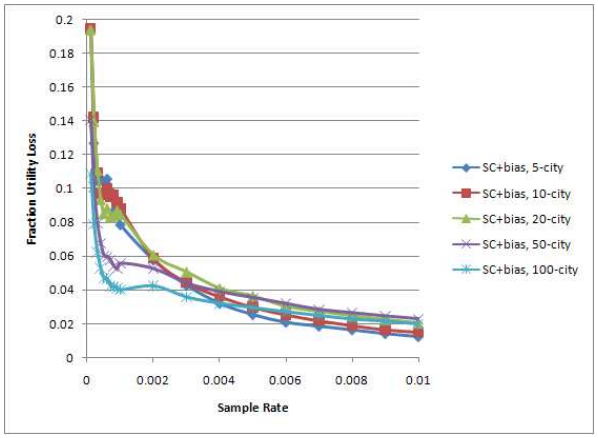

FIG. 4.2: Average utility loss of approximate PS Maps generated by SC+bias for DTSP problems of various sizes.

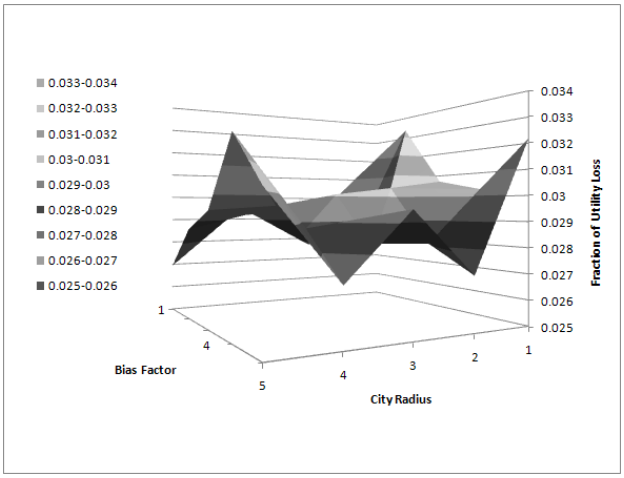

FIG. 4.3: Average utility loss of approximate PS Maps generated by SC+bias for 100-city DTSP problems at sample rate .005. SC-generated PS Maps generated under identical conditions have an average accuracy of .035.

4.1.10 Analysis

These results show that these algorithms are comparable to or better than online repair performance: all of the algorithms except for SC perform better than online repair at sample rates of .002 and above. It is quite reasonable that the alternate algorithms would perform better than SC, because SBE and SC+AL proactively attempt to find key problem instances that distinguish one solution from another, and SSS considers more information than SC during classification. The fact that the distance-squared version of SC performs better than the others suggests that solved problem instances that are closer to the instance being classified are more indicative of the proper solution than solved problem instances that are further away.

The results for SC+bias applied to TSP of various sizes are shown in Figure 4.2. Again, the results are comparable to online repair, but not as good as other techniques. SC+bias has *bias factor* and *city radius* parameters that can be modified and were set to various values within the experiment. Bias factor represents the degree to which to bias sampling to be near a city. The city radius indicates how close a problem instance has to be to a city to potentially benefit from the bias. Figure 4.3 shows utility loss results at sample rate .005 when SC+bias is applied with a range of parameter configurations. The results vary widely, and there does not appear to be any obvious correlation between specific parameter settings and the utility loss. This behavior also appears reasonable. The goal of this algorithm was to attempt to exploit city locations as indicators of boundaries between solution regions. However, there are many solution regions that are not near cities; thus, this algorithm has uneven and limited benefit.

The early experiments demonstrate that SSS and SBE have the best performance. SBE's performance is perhaps expected, as this algorithm most directly finds solution regions, thus exploiting the characteristic of this domain space in which similar problem

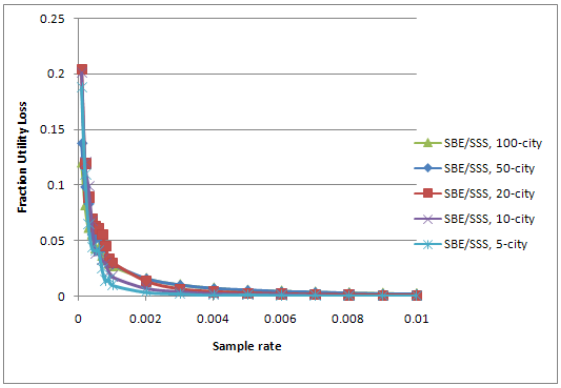

FIG. 4.4: Average utility loss of approximate PS Maps generated by SBE and SSS for DTSP problems of various sizes.

instances tend to have similar solutions. Alternatively, SSS's performance is best attributed to its brute-force approach of examining every problem instance and testing all known solutions. This would seem to continue to be feasible with a tractable number of problem instances and solutions, but may not scale well. Figure 4.4 explores SSS and SBE's potential with additional problem sizes. The performance continues to be good for all problem sizes, but appears to converge more rapidly for the smaller problem sizes. This behavior is expected due to the small number of unique solutions and larger homogeneous regions.

4.2 Knapsack Problem

The knapsack problem is a combinatorial optimization problem in which a subset of items of variable weight and value are chosen such that the total value is maximized and the total weight falls below a given threshold. For this experiment, I use the 0-1 knapsack problem variant, in which either zero or one copies of each item may be placed in the knapsack. The knapsack is prepopulated with a set of items that utilize 396 dekagrams (dag) of the total knapsack capacity of 400 dag, and one or more items of varying value

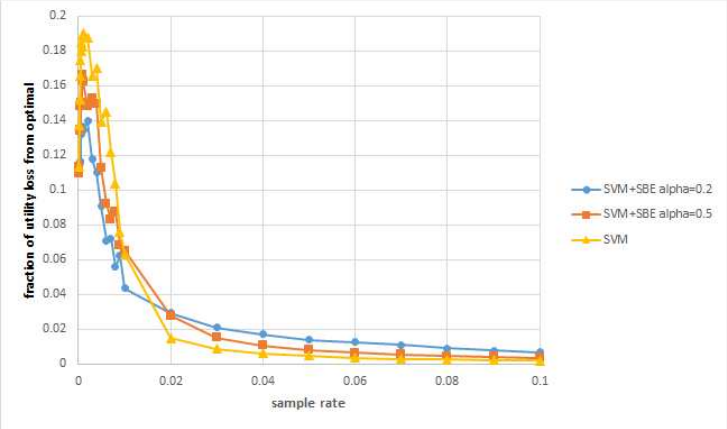

FIG. 4.5: Average utility loss of approximate PS Maps generated by SVM and SVM+SBE for 100-city TSP domain. Alpha refers to the fraction of samples used for random initial sampling.

and weight is added to the pool of items.

The knapsack domain demonstrates the applicability of the algorithms in a different domain. One difference between this domain and the TSP domain is that it entails a more abstract representation of distance, as an item's weight and value characteristics do not directly correspond to location and distance as do the cities within the TSP domain. The high-quality solution PS Map's characteristics also differ in this domain. For example, looking at the high-quality PS Map, one can see that, whereas the TSP domain had very circular homogeneous regions, the knapsack domain has rectangular homogeneous regions. I apply the same algorithms to this domain, with the exception of the SBE-trace algorithm, which is only suitable for problem spaces of two dimensions. I expect that performance of the algorithms could be worse in this domain, due to the greater number of solutions and smaller solution region size.

4.2.1 High-Quality PS Map

For the experiment, I defined a set of 22 items, each with known weight and value characteristics as shown in Table 4.1, from which to maximize the value of the knapsack while conforming to its maximum weight capacity. I defined one additional item, varying the weight and value from 1-100 inclusive to create 10,000 (100^2) problem instances. As a baseline, I solved all 10,000 problem instances to generate a high-quality PS Map. As with the TSP domain, I applied SSS over the PS Map to reduce errors from the heuristic solver. A visualization of the resulting two-dimensional PS Map is depicted in Figure 4.6.

I then generated more complex problem spaces by adding multiple items of varying weight and value characteristics to the pool. Solving each of the resulting problem instances – consisting of the static items and two additional items – resulted in a four-dimensional PS Map consisting of two weight and two value dimensions. I generated a high-quality PS Map, solving all 176,400 ($20^2 \times 21^2$) problem instances.

Continuing, I generated an eight-dimensional problem space consisting of a weight and value axis for each of four variable items. The range of the weight was 16 to 20 inclusive and the range of the value was 31 to 35 inclusive, resulting in a problem space of $5^8 = 390,325$ problem instances. For each of the problem instances in the problem space, I created the full problem instance by adding the variable items indicated by the problem instance to the knapsack. For example, if a problem instance in the problem space is $(w_0, v_0, w_1, v_1, w_2, v_2, w_3, v_3)$, then I solved a knapsack problem consisting of the pool of items in Table 4.1 plus items with weight and value scores of $(w_0, v_0), (w_1, v_1), (w_2, v_2)$, and (w_3, v_3). I solved each of the knapsack problems and created a mapping from each of the instances in the problem space to each of the calculated solution, thus composing the PS Map.

Finally, I generated a second PS Map of an eight-dimensional problem space as above,

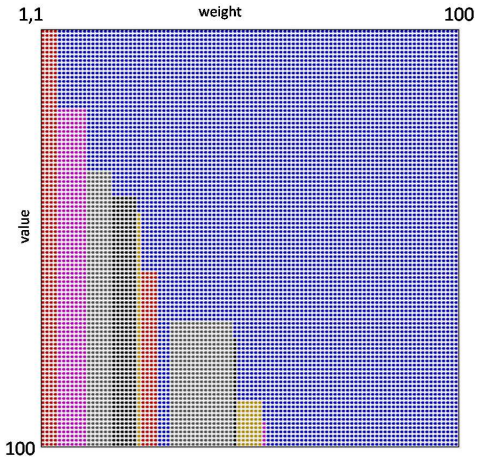

FIG. 4.6: High-quality PS Map for a knapsack problem. Best viewed in color.

but with the range of the weight expanded by one unit to 15 to 20, resulting in a problem space of $6^4 \times 5^4 = 810,000$ instances.

4.2.2 Online Repair Baseline

The online repair method is a greedy solver that selects the item with the highest value-to-weight ratio.

4.2.3 Experiment Parameters

The knapsack problem tested the sampling-classification (SC), sampling-classification with active learning (SC+AL), support vector machine (SVM), support vector machine with solution border estimation (SVM+SBE), and select from sampled solutions (SSS) methods. I did not perform experiments with SBE because, as previously mentioned, it is only applicable for two-dimensional domains, and, thus, is not as useful in general cases. The SC+Bias approximation algorithm is also omitted because its application is specific

to the TSP domain's city location parameters, and there is not a clear analog within the knapsack domain.

In my experiments, I found that large regions of the problem space were homogeneous, particularly as the values of the problem instances' variable features increase. To avoid positively skewing the results, I chose feature ranges to focus on the more heterogeneous regions of the problem instance space. For the two-dimensional experiment, I limited the problem space to problem instances with weights from 1-20, inclusive, and values from 50-70, inclusive. For example, when considering only one additional item, the first problem instance would consist of the static items plus an additional item with a weight and value (1,50); the second problem would consist of the static items plus an additional item with weight and value (2,50); and so forth, accounting for all possible combinations.

The approximation of all maps was done for sample rates ranging from .0001 to .001. For the SC+AL and SVM+SBE algorithms, the alpha rate was set to 0.5. Thus, the initial sample rate is half of the allocated samples, leaving half for active sampling. As before, the evaluation of the approximation is the fraction of the utility lost with respect to the heuristically calculated heuristic solution.

4.2.4 Results & Analysis

The result of generating a high-quality PS Map is displayed in Figure 4.6. The map confirms an intuitive estimation of solutions: for problem instances in which the variable item's weight falls within the slack of the original solution, it is always included in the knapsack. Once the variable item's weight exceeds the available slack, it is excluded from the knapsack until it becomes valuable enough to replace an item currently in the knapsack. Moving along the weight dimension, the variable item remains in the knapsack until it becomes too heavy for its value to contribute to an optimal solution and is excluded from the knapsack. This pattern repeats, creating a set of solutions resembling a staircase of

Object	Weight	Value
apple	39	40
banana	27	60
beer	52	10
camera	32	30
cheese	23	30
compass	13	35
glucose	15	60
map	9	150
note-case	22	80
sandwich	50	160
socks	4	50
sunglasses	7	20
suntan cream	11	70
t-shirt	24	15
tin	68	45
towel	18	12
trousers	48	10
umbrella	73	40
water	153	200
waterproof overclothes	43	75
waterproof trousers	42	70

Table 4.1: Knapsack static item pool

solutions, the edges of which represent a boundary in the solution space between where the variable item is included and excluded.

Figure 4.7 shows the results of applying the various PS Map approximation algorithms to a knapsack problem with one variable item. As one might expect, most of the algorithms trend towards zero utility loss as the sample rate increases. The notable exceptions are the SC and SSS algorithms. The SSS algorithm appears to provide somewhat of a theoretical best performance, with the other algorithms gradually converging. The SC algorithm appears to have a much slower convergence, as it still shows a loss of approximately 20% of the optimal utility at a 0.1 sample rate. Figure 4.8 highlights the turbulent region up to and including sample rate 0.01. Here it becomes apparent that the SC algorithm performs comparably to the other algorithms at this low sample rate, with the AL algorithm initially lagging behind. Figure 4.9 zooms in an additional time to the lower tenth of the sample rate range, up to and including 0.001, showing even more pronounced performance differences. The SC, SVM, and SVM+SBE algorithms are generally grouped together, and the AL and SSS algorithms show a utility loss at a fairly constant level at opposite ends of the performance range.

Figures 4.10, 4.11, and 4.12 show the relative rankings of the algorithms for the each of the preceding three figures. Although the quantitative difference in performance is lost in these graphs, it does notionally illustrate the preferred algorithm as the sample rate increases. We again see that the AL algorithms initially performs poorly, but converges quickly to become comparable to the SVM and SVM+SBE algorithms. Conversely, the SC algorithm performance degrades and quickly becomes the worst algorithm.

These results suggest that at very low sample rates, it is advantageous to use SC rather than AL, perhaps because SC's broader coverage of the space of problem instances is more useful than AL's targeted sampling for small sample rates. However, at higher sample rates, the higher number of samples available for AL's initial sample appears to provide

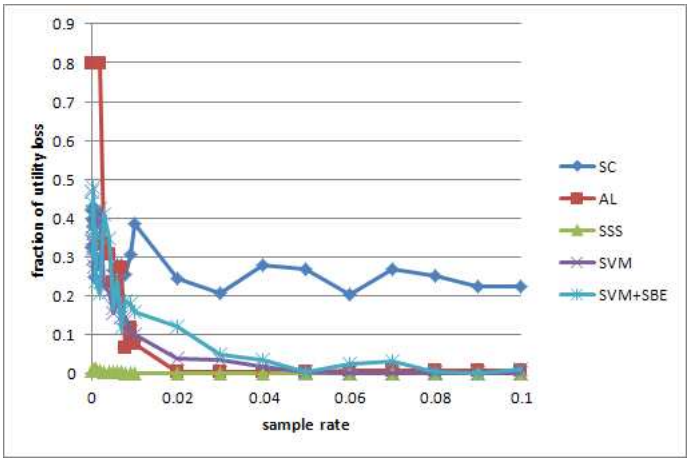

FIG. 4.7: Results of algorithms applied to a two-dimensional knapsack problem domain.

broad enough converage for the targeted sampling to outperform the SC algorithm. It is interesting that there is not the same level of distinction between SVM and SVM+SBE, perhaps because SVM's classifications methods permit the information gained from a sample to be applied more broadly, through the use of the maximum margin plane. SC and AL, on the other hand, limit the use of a sample's information to a very localized region. The advantage of SVM+SBE over SVM is that the targeted samples help to provide a more precise hyperplane location. However, in the knapsack domain, in which the utilities of the available solutions are similar, the benefit of the more precise hyperplane location is not as significant. Additionally, the number of samples available for targeted sampling may not provide enough information to create a more precise margin, particularly as the number of dimensions increases.

Figure 4.13 shows the results of applying the PS Map approximation algorithms to a knapsack problem with two variable items. Because each item has a weight and height characteristic, this results in a four-dimensional PS Map. In this experiment, I limited the range of the weight and value of the items to [14,24] and [30,40], respectively, due to the

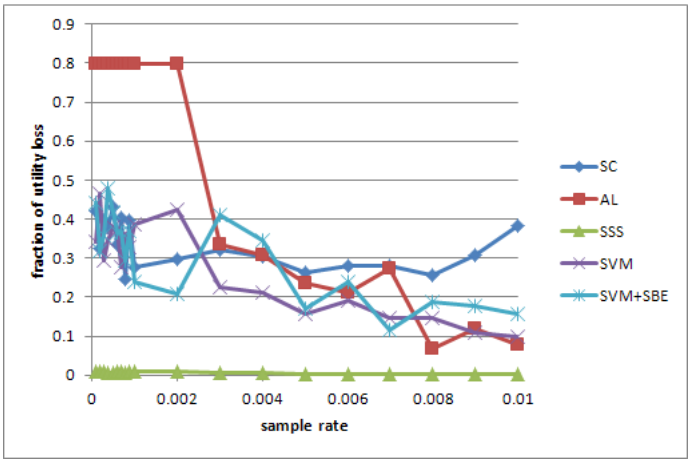

FIG. 4.8: Results of algorithms applied to a two-dimensional knapsack problem domain, focus on sample rate .01 and lower.

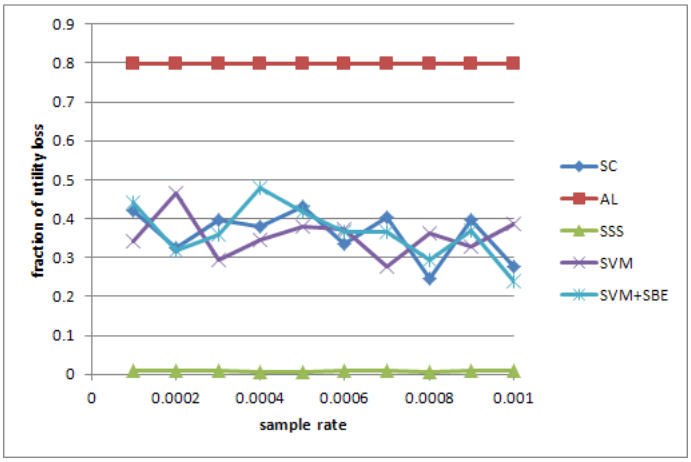

FIG. 4.9: Results of algorithms applied to a two-dimensional knapsack problem domain, focus on sample rate .001 and lower.

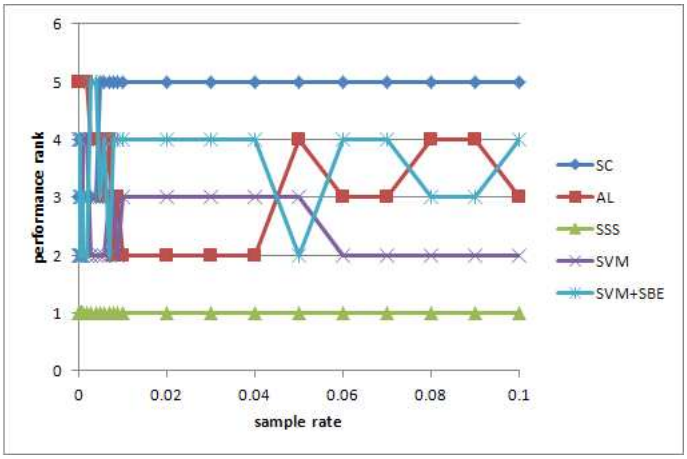

FIG. 4.10: Ranking of algorithms applied to a two-dimensional knapsack problem domain.

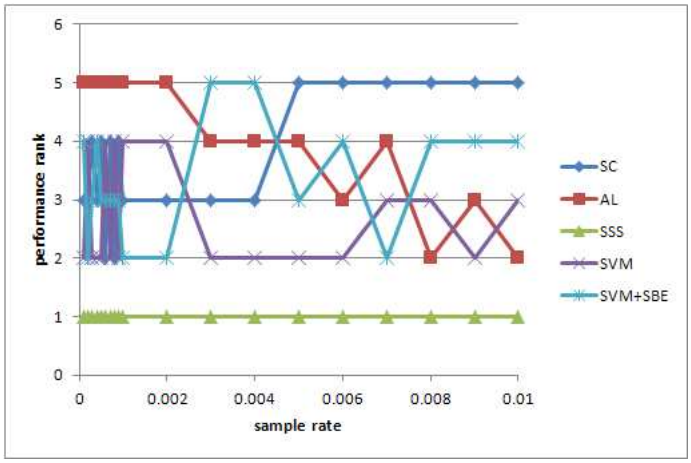

FIG. 4.11: Ranking of algorithms applied to a two-dimensional knapsack problem domain, focus on sample rate .01 and lower.

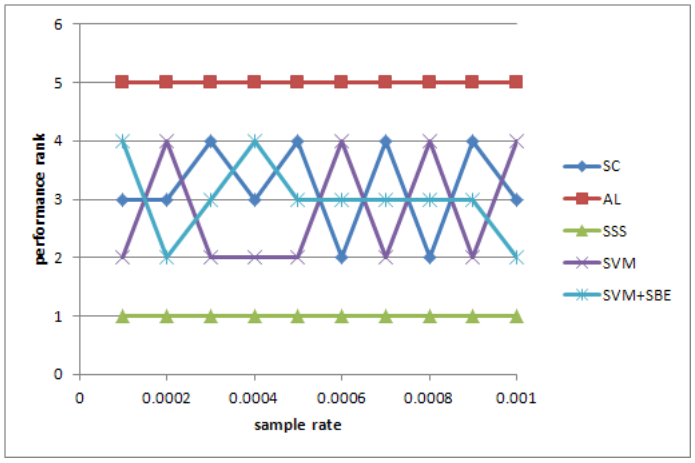

FIG. 4.12: Ranking of algorithms applied to a two-dimensional knapsack problem domain, focus on sample rate .001 and lower.

computation time required to complete the experiment. The graph shows a loss of utility well under 1% at low sample rates. In this domain, the algorithms appear to benefit from the higher dimensionality, because there is not a large increase in the number of unique solutions, leading to larger homogeneous solution regions that the algorithms can exploit.

The spikes in the SVM+SBE results are the effect of high variance that is a function of the manner in which SVM+SBE selects its sample points and the structure of the knapsack problem space. After the initial random sampling, SVM+SBE uses its additional samples to find problem instances that correspond to borders between pairwise solutions. As a result, all of SVM+SBE's subsequent samples will be in a region of the problem space that is bounded by the initial sample set. Therefore, if the initial sample does not bound a region that represents all solutions, then no subsequent samples will discover those solutions. In this test case, the region had a total of four solutions. If the initial sample discovered all four, then the average fraction utility loss was close to zero. If the initial sample discovered only three solutions and none were feasible with respect to the unrepresented problem

instances, then the average fraction utility loss rose to around 0.15. If the initial sample discovered only two solutions, the average fraction utility rose to around .45, indicating that the library did not have a feasible solution for almost half of the problem instances.

This effect is less pronounced in the other algorithms. For SC, SSS, and SVM, the probability of excluding a solution region at a particular sample rate is smaller because, unlike SVM+SBE, all of the samples are used in the initial sample, rather than a subset. For AL, which, like SVM+SBE, also reserves a fraction of its samples for targeted sampling, its subsequent sampling targets unrepresented regions, thereby reducing the probability that a region of the problem space would remain unsampled. The last factor is the domain, for which not all solutions are feasible for a given problem instance. In contrast to TSP, in which any solution can be applied to any problem instance, the knapsack problem domain defines a hard constraint – total weight – that if violated by a solution renders it inapplicable to the problem instance. This characteristic leads to large losses of utility because an infeasible solution has a utility close to zero,[3] whereas in a domain like TSP, a poor solution still does still contribute some portion of the optimal utility.

Figure 4.14 highlights the area of the graph where several of the algorithms appear to have similar performance. Upon closer inspection, the typical rapid convergence of the SSS algorithm is again visible. In this case, the other algorithms tend asymptotically towards zero utility loss as well.

Figure 4.15 puts these results in context against various baselines. The taller blue bars represent the fraction of utility lost if one were to assume a PS Map consisting of a single solution. Because the high-quality PS Map had seven solutions, there are seven cases represented in the graph. In this scheme, it is possible that the penalty for plan infeasibility could dominate the error results. The shorter red bar represents the result of applying a

[3]To avoid division-by-zero errors, the lowest utility in the knapsack problem domain is 1.

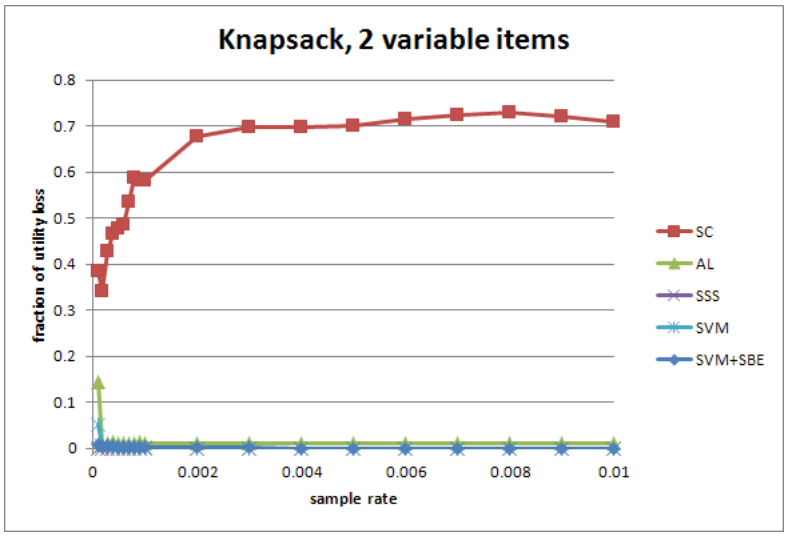

FIG. 4.13: Results of algorithms applied to a four-dimensional knapsack problem domain.

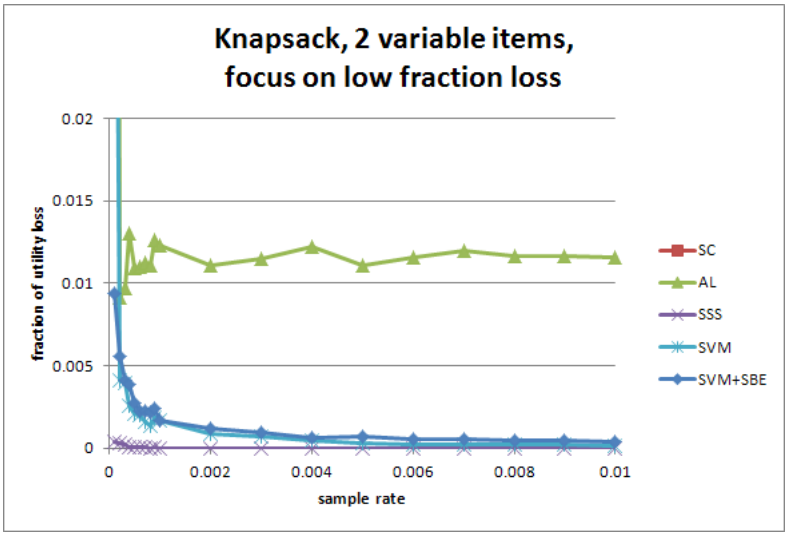

FIG. 4.14: Results of algorithms applied to a four-dimensional knapsack problem domain, highlighting low utility loss.

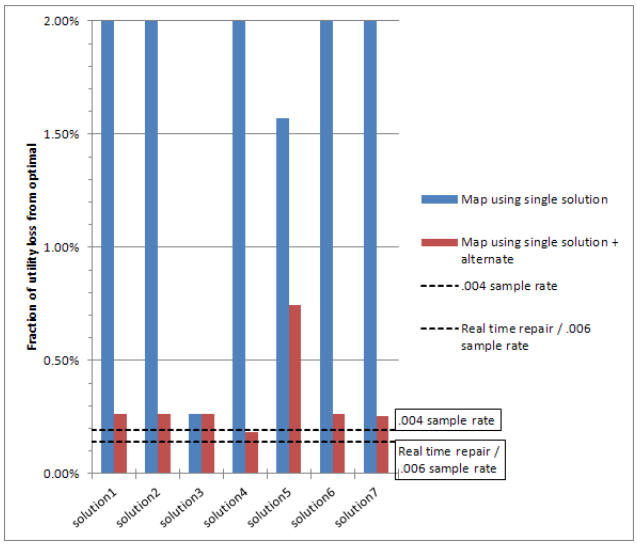

FIG. 4.15: Results of applying SVM+SBE to a knapsack with two variable objects. Dotted lines indicate fractional utility loss for online repair, .004 sample rate, and .006 sample rate, as labeled. Bars indicate the fractional loss when using a default map consisting of either a single default solution, or a default solution and the best found feasible solution.

default solution to a problem instance, but allows the system to choose an alternate solution if the default solution violates the weight threshold of the knapsack. In this case, a feasible plan is randomly chosen. The upper dotted line represents the fraction of utility loss of SVM+SBE at a sample rate of .004. The lower dotted line represents the identical loss of the online repair method as well as when sampling at a rate of .006 using SVM+SBE. The online repair method is a greedy solver that selects the items with the highest value to weight ratios. This demonstrates that the performance of the SVM+SBE algorithm when sampling at rate of .006 is roughly equivalent to that of the online repair technique.

4.3 Elevator Problem

The final domain, a elevator passenger transport problem, represents a more traditional planning domain. The TSP and knapsack domains can be considered optimization problems as well as planning problems. The elevator domain falls into the more traditional realm of planning, in which one has to find steps to accomplish a goal but there is no direct mathematical representation of the domain. Also, this domain is expected to be more challenging for the algorithms because the homogeneous regions are likely to be smaller and less regular. Lastly, this domain represents another level of abstraction, in that the solutions that are applied to the domain are not necessarily those that the algorithms will operate upon; the experiment parameters section describes this issue in detail. I also apply the algorithms to this domain, again with the exception of SBE-trace. I expect that this domain will the most challenging of the three, due to the possibility of changes in optimal plan being very sensitive to changes in the problem instance configuration.

The elevator domain is used by ICAPS in its International Planning Competition. It specifies several elevators, floors, and passengers, and requires the planner to deliver the passengers from their starting floor to their destination floor at the lowest possible cost. Each elevator is either “fast” or “slow.” The slow elevators incur little cost for movement, but more for stopping and starting. Conversely, the fast elevators incur more cost for movement, but little for stopping and starting. The planner specifies movements for the slow elevators, which may stop at any floor within a defined contiguous “block” of floors, and the fast elevators, which traverse the entire range of floors, but only stop at block centers and boundaries. For example, for a 12-floor problem with two slow elevators, one slow elevator will travel between the bottom six floors, and the other slow elevator will travel between the top six floors. The fast elevator will travel throughout the floors, but only stop at floors 0, 3, 6, 9, and 12. More formally, these features are specified with M and N param-

eters, which create a problem domain with M+1 total floors in blocks of N+1 floors, with fast elevators that may stop at floors that are multiples of $\frac{N}{2}$. Thus, in the example above, M is 12 (13 floors from 0 to 12) and N is 6 (two blocks each of seven floors, one from 0 to 6 inclusive, the other from 6 to 12 inclusive).

Typically, each planner submitted to the competition targets either the "optimal" or "sacrificing" track. The "optimal" track requires a planner to find the least costly means of transporting the passengers to their destinations. The "sacrificing" track does not require a planner to find the optimal plan, but only to find a feasible plan to deliver all of the passengers. My experiments focused on the optimal track and used one of the more successful planners, the LAMA Planner (Richter and Westphal, 2010).

4.3.1 High-quality PS Map generation

For this domain, I generated a 12-floor and two 24-floor elevator problems. The 12-floor problem contained two seven-floor blocks (M=12, N=6), two slow elevators, and one fast elevator. Each problem assumed two passengers with variable starting position, creating 169 problems to be solved with the LAMA planner. One 24-floor configuation consisted of six five-floor blocks (M=24, N=4), and the other contained four seven-floor blocks (M=24, N=6). The 24-floor problems vary the starting location of three passengers, thereby creating a high-quality map of 216 (i.e., 6^3) problem instances.

Similar to other hard problems, planners in this domain employ heuristics in order to solve these intractable problems, and thereby benefit from "smoothing" as described in Section 3.7: when generating the high-quality PS Map, each of the solutions is evaluated against each of the problem instances, and, if necessary, the problem instance is assigned a new solution. This prevents the odd phenomenon of the occasional approximate solution having better utility than the "optimal" solution, which may skew the results.

4.3.2 Experiment Parameters

My initial experiment used the 12-floor problem with three passengers, two slow elevators, and one fast elevator. I varied the starting positions of two passengers, resulting in a 169-instance problem space. In my initial experiment, there were too many unique plans, and the algorithms could not create classifications from the sampling. To make this domain appropriate for the algorithm, I abstracted the plans to transform a plan that moves elevators to a specific floor into a plan to move elevators to the location of specific passengers, thus creating a plan that could be applied to other problem instances. For example, consider a raw plan with the steps

```
(move-down-slow slow0-0 n6 n0)
(board p0 slow0-0 n0 n0 n1)
(move-up-slow slow0-0 n0 n3)
(leave p0 slow0-0 n3 n1 n0)
```

This plan specifies that, first, the slow elevator with id slow0-0 moves from floor 6 to floor 0. Next, the passenger with id p0 boards the elevator at floor 0, and the number of passengers increases from 0 to 1. Then the elevator moves from floor 0 to floor 3, and in the final step, the passenger leaves the elevator at floor 3 and the number of passengers in the elevator decreases from 0 to 1.

In order to make this plan reusable, it is transformed to be more general:

```
elevator slow0-0 picks up passenger p0
elevator slow0-0 drops off passenger p0
```

The first step specifies that the elevator with id slow0-0 moves to passenger p0's current location, and p0 boards the elevator. The second step then specifies that the elevator moves to passenger's desired destination and the passenger disembarks. This general plan

can be applied to problem instances in which the elevator and passengers are on floors other than those assumed by the raw plan.

In addition to abstracting the plans, I normalize the plan so that differences in the ordering of independent actions are not interpreted as distinct plans. For example, consider the plan below, annotated with action ids for ease of reference:

```
1: elevator slow0-0 picks up passenger p0
2: elevator slow1-0 picks up passenger p1
3: elevator slow0-0 drops off passenger p0
4: elevator slow1-0 drops off passenger p1
```

Note that the only dependencies are that action 1 must occur before action 3, and action 2 must occur before action 4. Thus, there are six potential plans[4] representing the same overall process. I normalize the plan by grouping together as many actions as possible that are performed by the same elevator. In this case, the resulting normalization is:

```
1: elevator slow0-0 picks up passenger p0
3: elevator slow0-0 drops off passenger p0
2: elevator slow1-0 picks up passenger p1
4: elevator slow1-0 drops off passenger p1
```

My subsequent experiments used a 24-floor elevator problem with six passengers, three of which had variable starting locations. One experiment used six fast elevators and three slow elevators, and the other used four slow elevators.

4.3.3 Online Repair Baseline

As a baseline, I implemented an online repair algorithm. van der Krogt and de Weerdt (2005) describe plan repair as consisting of removing actions from the original plan that

[4](1,2,3,4), (1,2,4,3), (1,3,2,4), (2,1,3,4), (2,1,4,3), and (2,4,1,3)

conflict with or impede achieving the new goal, followed by adding actions to the original plan that allow it to achieve the new goal. My baseline online replanning algorithm is consistent with this methodology. The new goal changes the initial location of the passenger, and thus I consider all actions that reference that passenger as candidates for deletion. van der Krogt and de Weerdt suggest that heuristics should be used to determine if a candidate action should be deleted. My heuristic is a simple one: I only remove the candidate action if it refers to a passenger whose starting position has moved outside the range of the elevator used by the action. For example, consider an abstracted action *elevator slow0-0 picks up passenger p0*. Elevator slow0-0's range is floors n0 through n6. If this action is applied to a problem instance in which p0's starting position is n7 or above, then the action would be removed.

In the event that an action is removed, I proceed with the second component of plan repair, in which I add actions to the original plan to achieve the new goal. There are two alternatives for continuation: either remove all subsequent actions that refer to the passenger and replan the entire route, or preserve the subsequent actions and replan the passenger route to comply with the constraints implied by the subsequent actions. In the case of the former, I generate a solution to transport the passengers whose actions were removed. In order to plan without the influence of the passengers whose actions have already been established, the initial starting conditions of those passengers is set to be equal to their destination location. In the case of the latter, the final condition is set to the location expected by the action that moves the passenger to its final destination. For example, if an action moves p2 from n6 to n2 to complete its journey, then the planner will set the final destination to n6.

Algorithm 8 Unrefinement

```
1: for each passenger p in plan P do
2:     if first action referencing p is invalid then
3:         remove all actions referencing p
4:     end if
5: end for
```

Algorithm 9 Refinement

```
1: actions ← generate plan for deleted passenger actions
2: parse and abstract actions
3: add actions to P
4: normalize P
```

4.3.4 Results

Results from the 12-floor elevator problem domain are displayed in Figure 4.16. The dotted lines represent the fractional utility loss of three independent runs of the online repair algorithm described in Algorithms 8 and 9. The solid lines represent the results of applying the SVM+SBE approximation algorithm with various SVM kernels and the SSS algorithm. The results demonstrate that the SSS algorithm has less fractional utility loss than the online algorithms, but the various SVM+SBE algorithms generally perform worse than the online repair algorithms.

Figures 4.17 through 4.19 show results of all the algorithms applied to the same 12-floor configuration mentioned above. Again, the utility loss is much greater in this domain than in other domains. This is due to the small number of unique solutions and the smaller size of homogeneous regions in the space. This effect can be observed more explicitly by examining the performance of the algorithms in two different 24-floor configurations.

Results from the 24-floor elevator domain experiments are shown in Figures 4.20 through 4.25. These results show that for each problem configuration, SSS performs better than SVM+SBE. Additionally, the algorithms perform better against the problem configu-

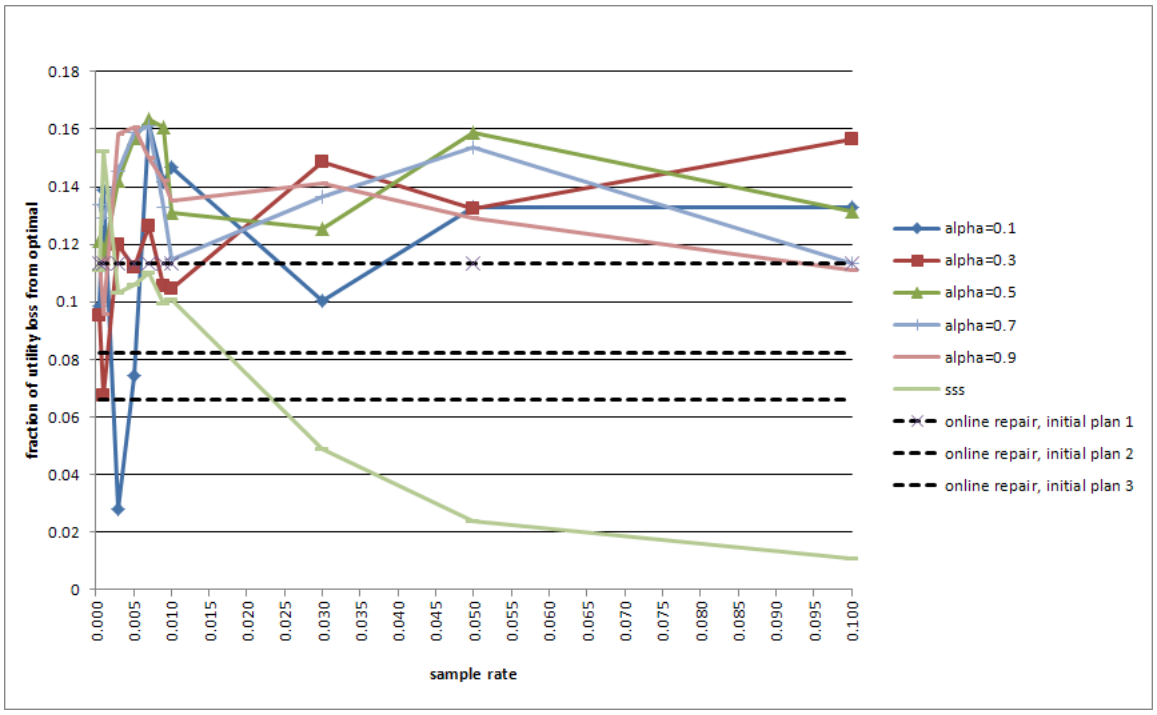

FIG. 4.16: Results of applying the SVM+SBE algorithm to a 12-floor elevator problem consisting of 2 slow elevators, 1 fast elevator, and 3 variable passenger starting locations. Dotted lines represent the utility loss of online repair. Solid lines represent approximations using various alpha values.

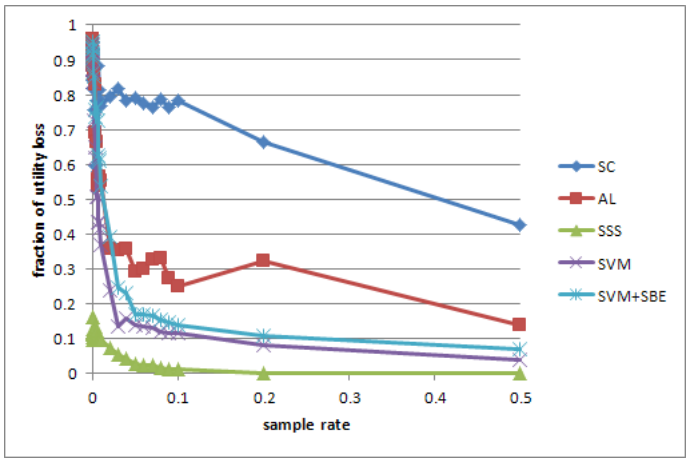

FIG. 4.17: Results of applying various approximation algorithms to a 12-floor elevator problem consisting of 2 slow elevators, 1 fast elevator, and 3 variable passenger starting locations.

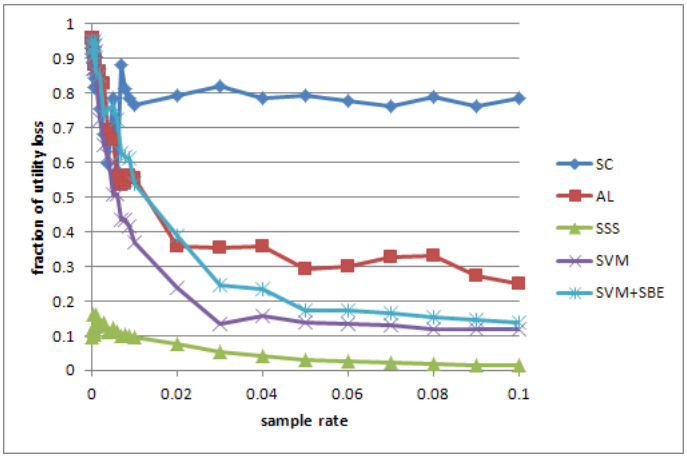

FIG. 4.18: Results of applying various approximation algorithms to a 12-floor elevator problem consisting of 2 slow elevators, 1 fast elevator, and 3 variable passenger starting locations, focus on sample rate 0.1 and lower.

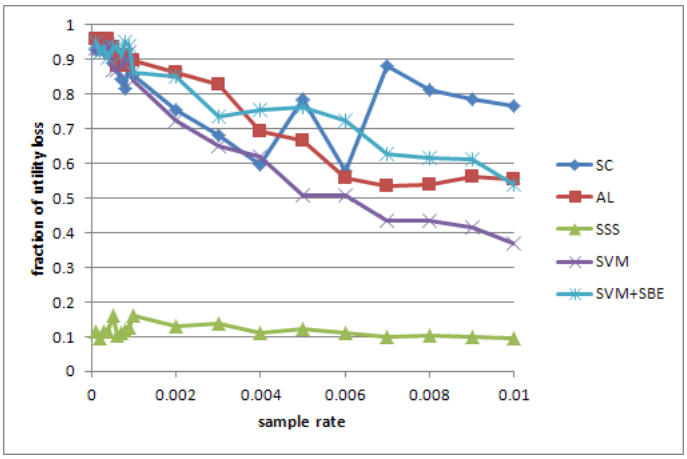

FIG. 4.19: Results of applying various approximation algorithms to a 12-floor elevator problem consisting of 2 slow elevators, 1 fast elevator, and 3 variable passenger starting locations, focus on sample rate 0.01 and lower.

ration with fewer elevators. This is not unexpected, given the nature of the problem space of each configuration and the algorithms used. In all of the configurations, the problem spaces have homogeneous regions, but they are small, which can make it difficult for an SVM-based algorithm to converge and find the appropriate boundaries. However, those small regions are not a disadvantage for the SSS algorithm, because it chooses a solution for each unsolved problem instance, rather than attempting to find groupings like SVM+SBE. This same logic is applicable to the generally better results for the problem configuration with fewer elevators. In the configuration with four slow elevators, the homogeneous regions are larger than in the problem space with six slow elevators, and thus the SVM+SBE algorithm performs better. Because there are fewer total solutions in the configuration with fewer elevators, the SSS algorithm performs better as well.

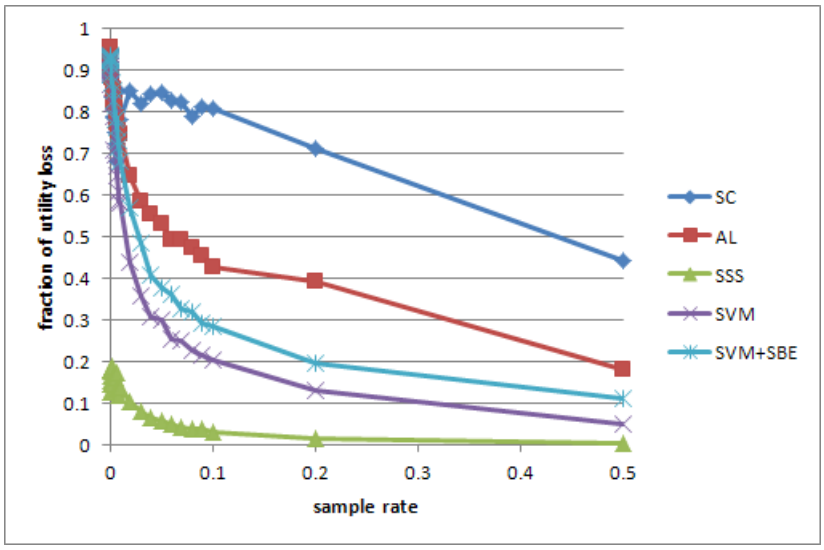

FIG. 4.20: Results of applying various approximation algorithms to a 24-floor elevator problem consisting of 6 slow elevators, 3 fast elevators, and 3 variable passenger starting locations of 6 total.

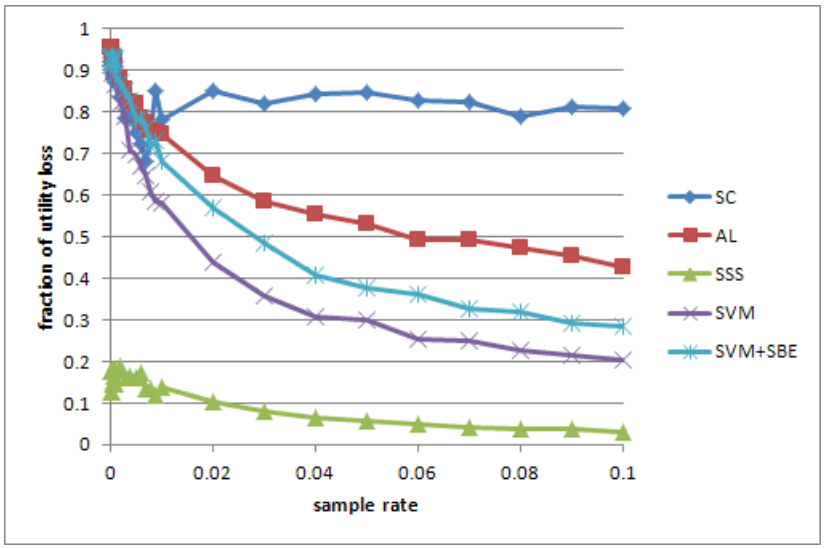

FIG. 4.21: Results of applying various approximation algorithms to a 24-floor elevator problem consisting of 6 slow elevators, 3 fast elevators, and 3 variable passenger starting locations of 6 total, focus on sample rate 0.1 and lower.

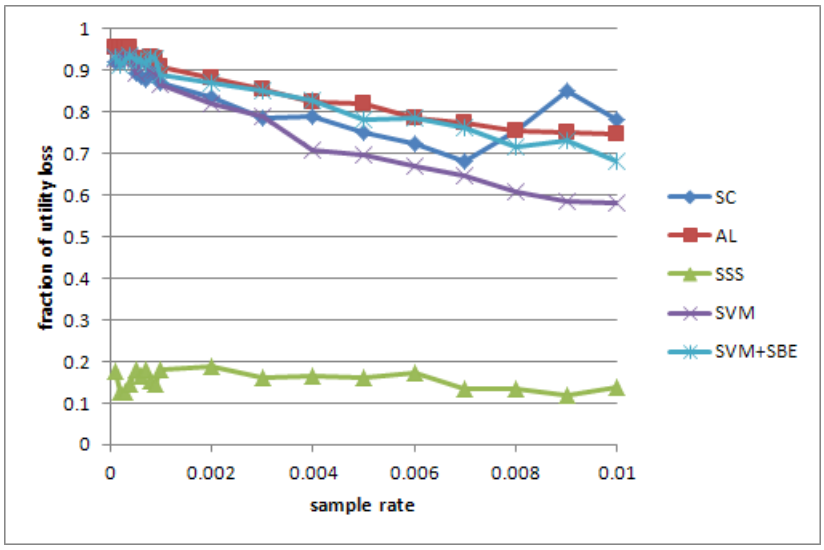

FIG. 4.22: Results of applying various approximation algorithms to a 24-floor elevator problem consisting of 6 slow elevators, 3 fast elevators, and 3 variable passenger starting locations of 6 total, focus on sample rate 0.01 and lower.

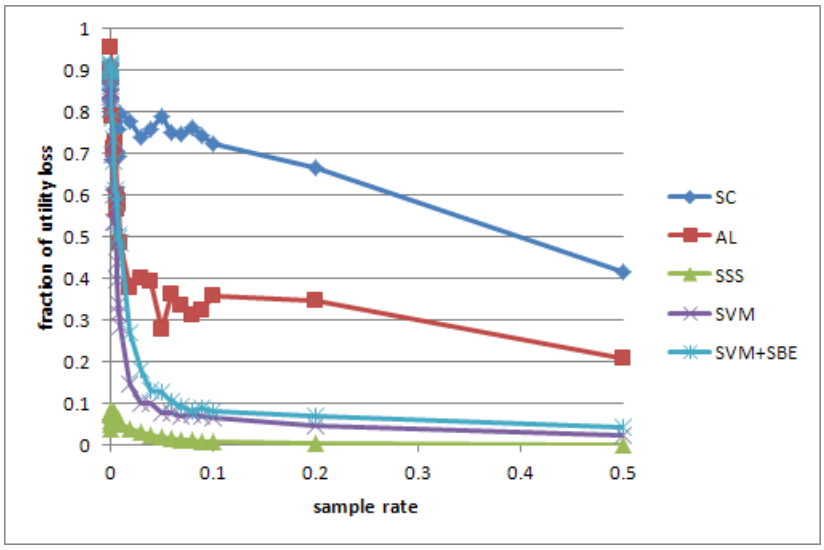

FIG. 4.23: Results of applying various approximation algorithms to a 24-floor elevator problem consisting of 4 slow elevators, 0 fast elevators, and 3 variable passenger starting locations of 6 total.

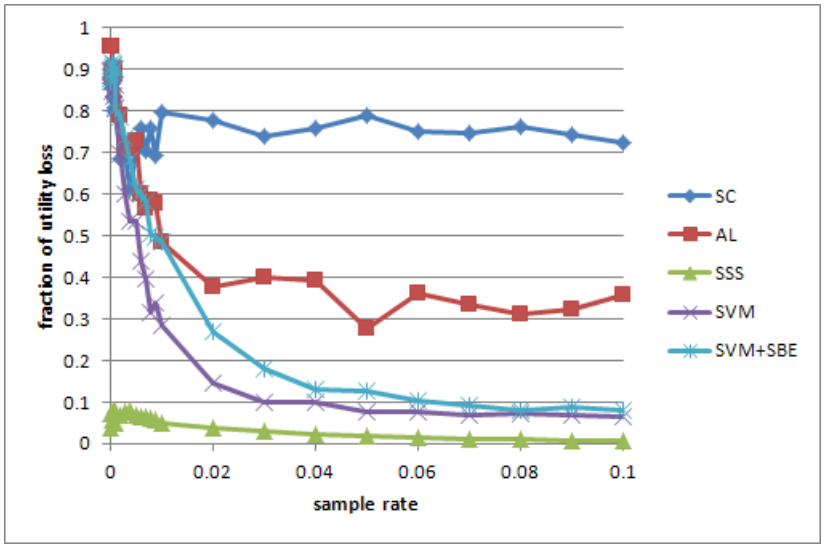

FIG. 4.24: Results of applying various approximation algorithms to a 24-floor elevator problem consisting of 4 slow elevators, 0 fast elevators, and 3 variable passenger starting locations of 6 total, focus on sample rate 0.1 and lower.

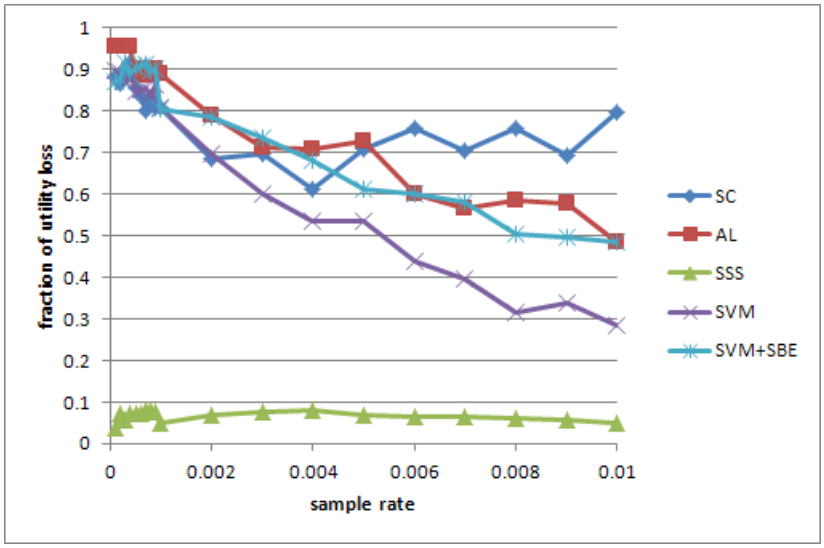

FIG. 4.25: Results of applying various approximation algorithms to a 24-floor elevator problem consisting of 4 slow elevators, 0 fast elevators, and 3 variable passenger starting locations of 6 total, focus on sample rate 0.01 and lower.

4.4 Overall Analysis & Discussion

The results generally show little difference between approaches at low sample rates. In fact, the results tend to be highly volatile, perhaps due to the high dependence on a small number of samples, which leads to large variances in the solutions available for the algorithms to consider.

SC tends to be a reasonable approach, with the added benefit that it is very simple to implement. SC+bias shows the ability to improve on SC; however, it is not clear how to tune its parameters to achieve consistently good results. SBE clearly is the best performer in the domains in which it was applied. However, my implementation of SBE is limited to two dimensions. SSS tends to have the same results as SBE, but SSS is more computationally intensive, potentially leading to scaling issues in large problem spaces.

The use of the SVM approach was intended to mimic the idea of SBE, but adds the ability to apply it higher-dimensional domains. This approach tended to yield reasonable results. Augmenting SVM with additional points to constrain the margin plane in SVM+SBE did not appear to have the significant impact one might have expected. This may be because the test domains are fairly forgiving when applying a less optimal solution to a problem instance. A domain in which there is a larger penalty for less optimal solutions may require an approach like SVM+SBE, which would provide a more accurate classification of solutions to the problem instances. Although SVM and SVM+SBE did not achieve the results of SSS or SBE, they do have the advantage of being less computationally intensive and being applicable to domains of more than two dimensions.

The results of all algorithms appear to be sensitive to problem domain characteristics. In the case of TSP, for example, larger size and a smaller quantity of homogeneous solutions regions generally resulted in better performance by the algorithms. This was also apparent when comparing the performance of the algorithms in the various elevator domain config-

urations. The algorithms were able to perform well when tested against configurations that resulted in large homogeneous spaces in the solution space.

Most importantly, the results do demonstrate that these techniques are useful as an alternative to online plan repair. At the appropriate sample rate, performance tends to be comparable, and sometimes better, than the online solution. The benefit of my approach is that, assuming the ability to compute the necessary library before the environment changes, the new plan can be accessed much more rapidly than the online repairer can calculate a new plan. These tradeoffs are discussed in more detail in the next chapter.

Chapter 5

DISCUSSION

This chapter presents considerations when using the problem space analysis (PSA) algorithms described in the previous chapters. It details some of the implicit criteria for effective use of the algorithms, discusses the tradeoffs between using online repair and PSA, and describes other potential applications.

5.1 Algorithmic Assumptions

The effectiveness of the algorithms described in the previous chapters requires the existence of regions in the problem space with identical solutions. Fewer regions and larger region size allow the algorithms to be more effective. This was demonstrated through the experiments in which TSPs with fewer cities created fewer, larger homogeneous solution regions and had better PS Map approximation. Likewise, elevator domains with larger N values – that is, larger blocks of floors – tend to be more readily approximated by the algorithms. Conversely, increasing the number of fast elevators potentially increases the number of regions, and, consistent with the results, becomes less amenable to approximation by the algorithms.

Thus, plan solution spaces have homogeneous solution regions that the algorithms attempt to exploit. These regions could be considered a function of a problem domain's

objective function, as demonstrated by the justification for SBE. Recall that the goal of SBE is to mathematically discover boundaries between solution regions by equating the objective functions of problem instances with differing variable features. In this way SBE discovers problem instances for which two solutions have equal utility, thus constituting a boundary between two solution regions.

As seen with SBE and its skeletal generation of solution region boundaries, a TSP's solution boundaries are defined by each pairwise set of unique solutions discovered by an initial sample. Fewer unique solutions increases the number of problem instances per solution; that is, it increases the size of the solution regions. At a given sample rate, the larger solution regions create a greater likelihood that a random sample will include the points necessary to identify the unique solutions within the problem space.

Other problem domains, such as the elevator domain, do not have explicit objective functions, but do have problem configurations that can serve the same purpose. Within the TSP domain, the number of fixed cities affects the number of possible unique solutions, a fraction of which are represented in the problem space as solution regions. In the knapsack domain, the set of static items affects the number of possible unique solutions as well. As the value of a variable item increases, it eventually supersedes a static item, resulting in a new solution. For example, every problem instance in which the variable item is "worse" than the "worst" static item will have a solution that includes the static item rather than the variable item.[1] The problem instances for which the variable item is "better" than the static item will result in a distinct solution. Each static item presents an opportunity for a new solution. Thus, increasing the number of static items that are present in the domain results in more distinct solutions and thereby more solution regions will exist.

In the elevator domain, the number of blocks of floors affects the number of solution

[1]The evaluation of "worse" and "worst" depends on the heuristic used by the solver, but one example is the ratio of weight to value.

regions. In general, the number of steps for a passenger to move from its starting to final destination is a function of the floor block that contains its starting location. If abstracted as previously described, there is potentially little difference in the solution regardless of where in the floor block the passenger starts, which itself suggests an identical solution for several starting locations. The greater number of floor blocks thereby leads to a greater number of solution regions.

The general conclusion is that the static characteristics have a direct impact on the number of homogeneous solution regions. This can be observed in the previous examples in which the static characteristics tend to serve as an indicator of a threshold that variable features may cross and create a distinct solution.

In order to create these homogeneous regions, the axes used in the PS Map must be chosen appropriately, such that the problem instances with similar solutions are grouped together. In the TSP domain, indexing by the x- and y-coordinates of the variable location resulted in homogeneous regions; in the knapsack domain, indexing by the variable item's weight and value results in homogeneous regions; and in the elevator domain, indexing by the starting passengers' starting location resulted in homogeneous solution regions. In other domains, the surface attributes may not provide a natural grouping. For example, I briefly investigated the problem space of a game, *Alien Frontiers*. This game falls in the category of worker placement, in which a player rolls at least three and sometimes up to seven dice, and may choose to place dice of meeting certain criteria in a "docking station." For example, two or three of a kind is required for some docking stations; others require three dice of consecutive increasing value (e.g. 3,4,5); and others merely require a total value of greater than seven. Particularly in the early game, a pair is a valuable roll, and my solver would generally create one class of plan for rolls containing a pair, and another for rolls not containing a pair. In this domain, indexing by the value of the dice did not result in homogeneous regions. Rather, a better indexing scheme in this case would have been a

derived boolean attribute, indicating “pair” or “not pair.”

In addition to appropriate indexing, the plans must be abstracted enough to create similar plans that can form homogeneous regions. This is demonstrated in the elevator domain in which the raw plans were abstracted to more generic plans. If indexing and abstraction result in homogeneous clumps, then an either an SBE approach, in which objective functions are equated, or an SVM+SBE approach could be appropriate. If not, SSS could be a viable alternative.

5.2 Tradeoff with Online Repair

These techniques allow a system to find solutions for large numbers of similar problem instances, providing useful information in domains that do not allow for large amounts of replanning time once an incident occurs, but in which there is some time before such an incident.

Given that the sample rate determines the accuracy of the approximated map, a system would want to use the highest sample rate possible. In the case where the system knows the expected time until a disruptive event occurs, then this technique could be used as a contract algorithm (Zilberstein et al., 1999) – an algorithm that is given a specific amount of time with which to find a solution – with a sample rate:

$$rate = \frac{time_{\textit{offline}}}{time_{inst} * n_{inst}},$$

where $time_{\textit{offline}}$ is the estimated time preceding the disruptive event, $time_{inst}$ is the time required to solve a single problem instance, and n_{inst} is the total number of problem instances in the space. (More intuitively, it is the amount of offline time divided by the amount of time that would be required to solve every problem instance.) In the case where there is no knowledge of the length of time until the disruptive event, then the system can

define $time_{\textit{offline}}$ as a periodic "refresh" interval that triggers the generation of a new PS Map, or use a real-time algorithm approach in which PS Maps are generated with successively larger sample rates until the time of the event.

Considering the test domains of the previous chapter, the tradeoff can be made more concrete. The typical time to solve a 100-city TSP with the heuristic solver is three seconds on a laptop and approximately 0.4 seconds on a high-performance machine. The knapsack problem required .016 seconds on a high-performance machine, and the elevator domain required 30-60 seconds on the same machine. Knowing that the online repair for a 100-city TSP has a fractional utility loss of approximately .02 and mapping that to a SVM+SBE approximation sample rate of .002 in Figure 4.1, one can insert these values to the equation.

$$.002 = \frac{time_{\textit{offline}}}{0.4sec * 10000}$$

This results in a $time_{\textit{offline}}$ of eight seconds. Thus, if the system comparable to the laptop's capability has eight seconds or more with which to preplan, then it is advantageous to use PSA. Otherwise, plan repair is probably a better option. For the knapsack domain, using the online repair results from Figure 4.15, I obtain the equation

$$.006 = \frac{time_{\textit{offline}}}{.016sec * 10000}$$

This results in a $time_{\textit{offline}}$ of 0.96 seconds. Of course, determining whether investing the required lead time or the online repair time is preferable would be application-specific. Figures 5.12 and 5.13 show the quickly increasing solver time required as the problem sizes grow larger, which would imply that the time required to generate a PS Map would also

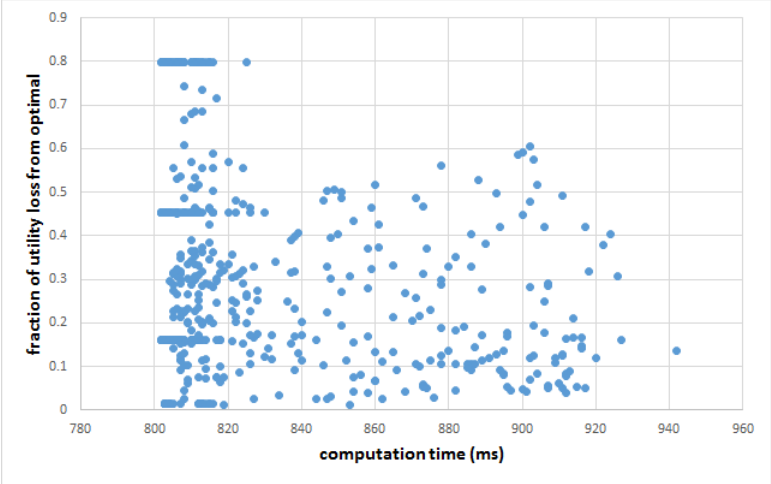

FIG. 5.1: Relationship between SC approximation computation time and map quality for a two-dimensional knapsack domain.

increase. In the same way, online repair time for increasing problem complexity would also increase. This again points to a tradeoff between the increasing solution time needed for PSA and the expected decline in the performance of online repair.

A more comprehensive view of this tradeoff is shown in Figures 5.1 through 5.5. Looking at the knapsack results, there does not seem to be an especially strong correlation between the computation time and the utility loss. Given the random nature of the SC algorithm, it is not surprising that there is a lot of variation in the results. The other algorithms show a stronger relationship between computation time and performance. This is not surprising given the more directed nature of these algorithms.

Similar to the knapsack problem, the elevator domain shows a strong correlation between computation time and utility loss when the more sophisticated algorithms are employed, as shown in Figures 5.6 through 5.10. However, even the random SC algorithm in this domain seems to show a tendency towards better performance at high sample rates.

It is worth noting the discrete characteristic of several of the maps. This results from

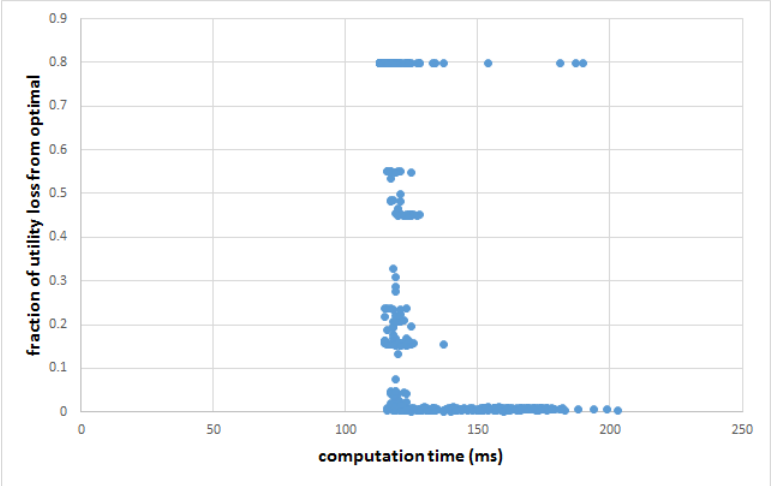

FIG. 5.2: Relationship between SC+AL approximation computation time and map quality for a two-dimensional knapsack domain.

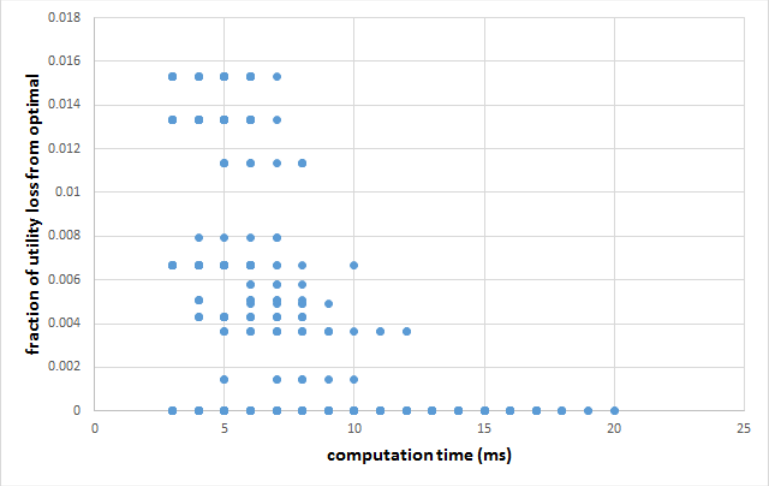

FIG. 5.3: Relationship between SSS approximation computation time and map quality for a two-dimensional knapsack domain.

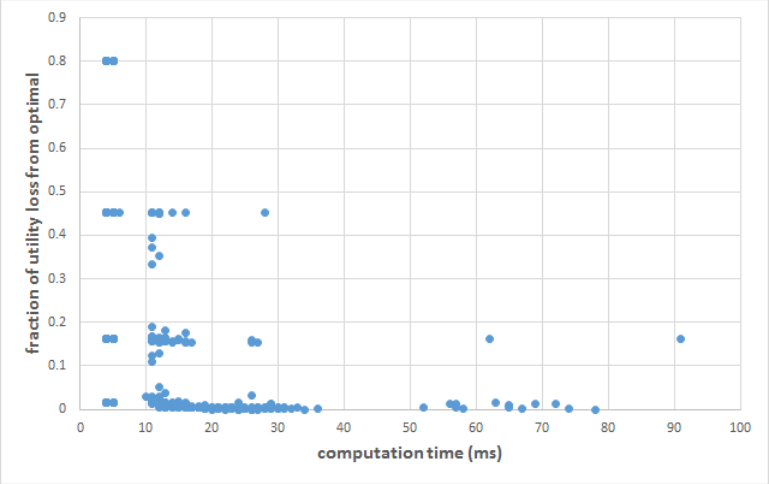

FIG. 5.4: Relationship between SVM approximation computation time and map quality for a two-dimensional knapsack domain.

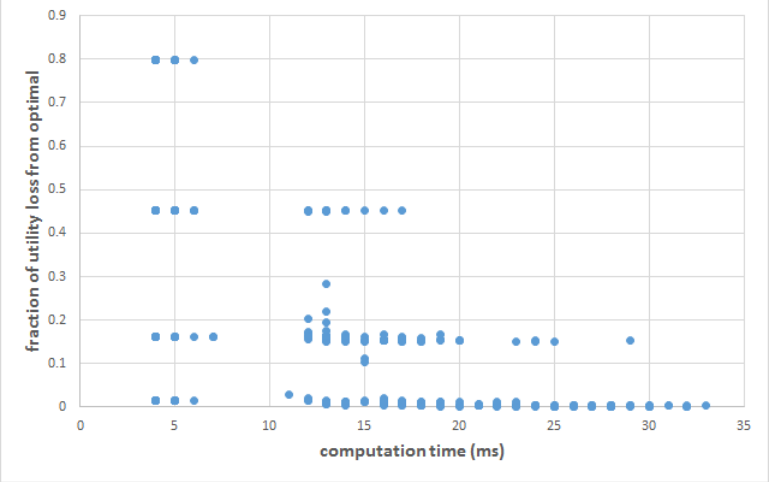

FIG. 5.5: Relationship between SVM+SBE approximation computation time and map quality for a two-dimensional knapsack domain.

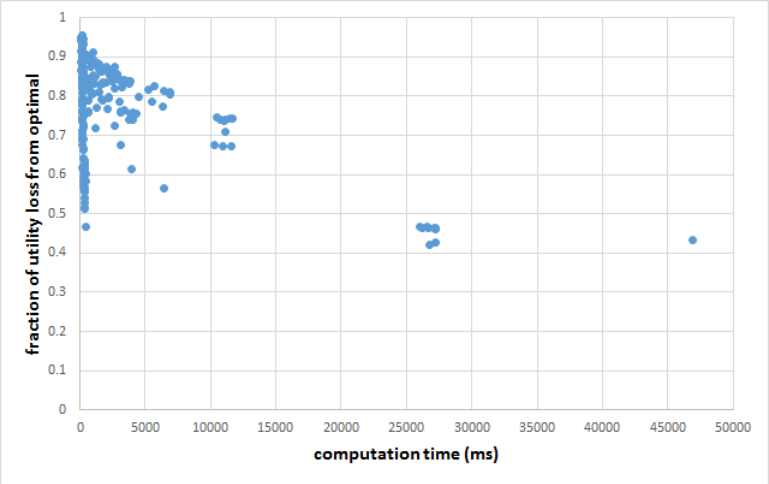

FIG. 5.6: Relationship between SC approximation computation time and map quality for a three-dimensional elevator domain.

very low variance in utility loss as a function of the discovered solutions. That is, the set of solutions that the initial sampling discovers tends to determine the overall performance of the approximation algorithm. This is most evident in the SSS algorithm, in which the assignment of the solutions to unsolved problem instances is most directly determined by the set of discovered solutions. Recall that in SSS, each unsolved problem is assigned a solution by testing each previously discovered solution. Other approximation algorithms attempt to avoid testing all discovered the solutions, but, outside of SC, these algorithms still tend to use the information from the set of discovered solutions in a consistent, although not deterministic, manner.

This discrete characteristic appears less frequently in the elevator domain, likely due to the larger number of solutions available in the domain. Thus, the set of discovered solutions is more varied.

This leaves the question of how to accurately estimate the expected performance is for a particular sample rate. Recalling that the effectiveness of the algorithms appears to

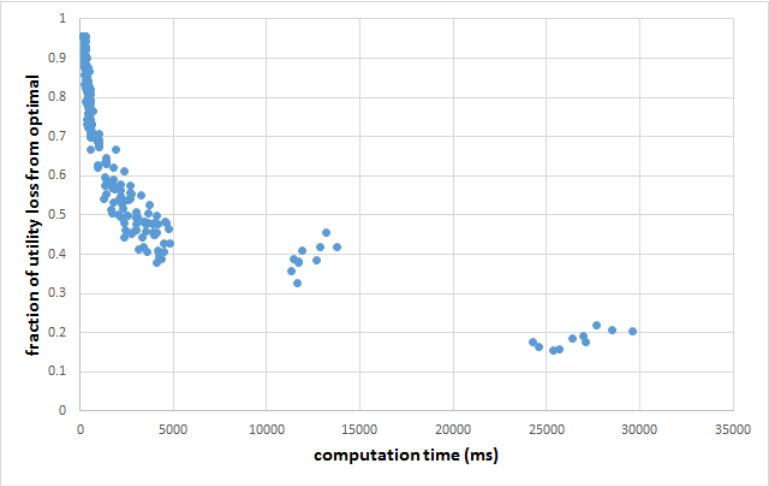

FIG. 5.7: Relationship between SC+AL approximation computation time and map quality for a three-dimensional elevator domain.

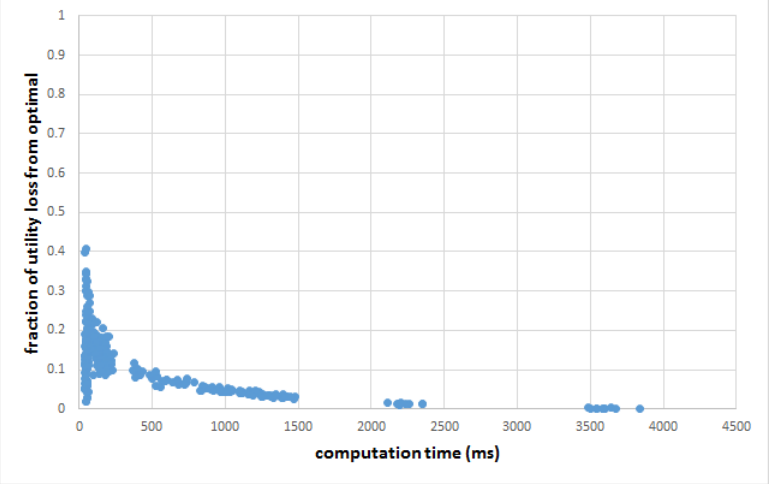

FIG. 5.8: Relationship between SSS approximation computation time and map quality for a three-dimensional elevator domain.

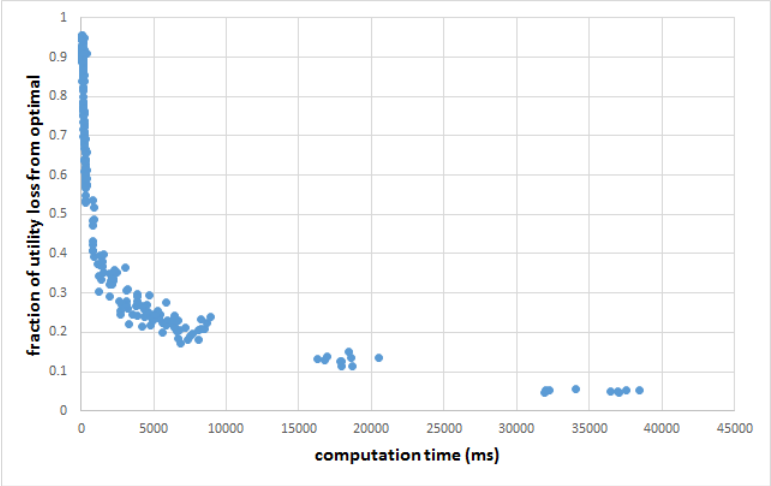

FIG. 5.9: Relationship between SVM approximation computation time and map quality for a three-dimensional elevator domain.

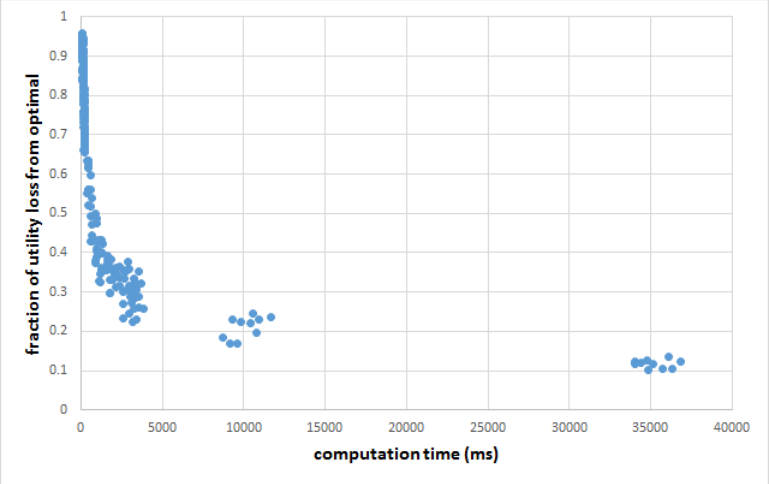

FIG. 5.10: Relationship between SVM+SBE approximation computation time and map quality for a three-dimensional elevator domain.

be a function of the number of size of homogeneous solution regions, it may be possible to estimate the number of homogeneous solution regions by examining the characteristics of the objective function or the problem configuration. For example, an elevator domain with configuration M=12, N=6, with one slow elevator per block, has approximately two solution regions: one in which the elevator in the first block picks up the passenger, and one in which the elevator in the second block picks up the passenger. One might then speculate that the number of solution regions is approximately $\frac{M}{N}$, assuming one slow elevator per block and zero fast elevators. Determining the number of solution regions in other domains is potentially less straightforward. For example, Figure 5.11 shows PS Maps for several randomly configured DTSPs. The unique solutions vary from eight to eleven.

As one experiment shows, the number of unique solutions in a knapsack PS Map is approximated by the number of unique item weights in the knapsack prior to the consideration of the variable item. In the first experiment, I started with a knapsack of static items and generated a PS Map for several weight threshold values. For each weight threshold, I found the knapsack solution and recorded the number of unique item weights. I then introduced the variable item, generated the PS Map, and recorded the number of unique solutions.

In the second experiment, I kept the weight threshold constant and generated a PS Map for several static item configurations. As in the first experiment, I recorded the number of unique weight values in the static item set, added the variable item, generated the PS Map, then recorded the number of unique solutions in the PS Map.

The results of the first experiment demonstrate that the number of unique solutions per number of unique weights is approximately 1.02.

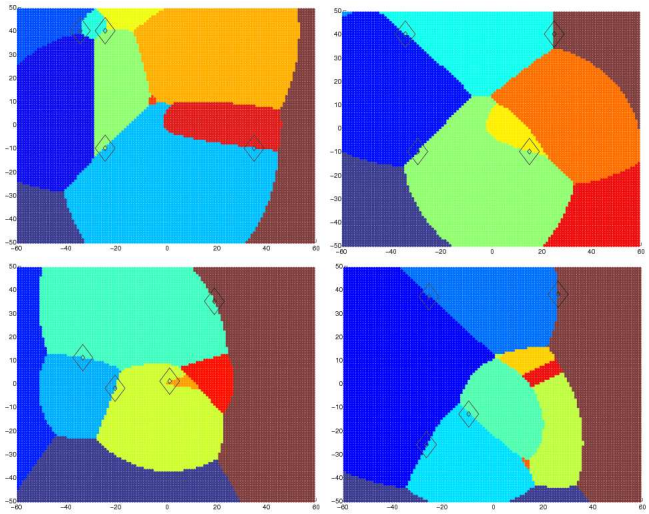

FIG. 5.11: Various high-quality PS Maps of five-city TSPs. Total number of unique solutions varies from eight to eleven.

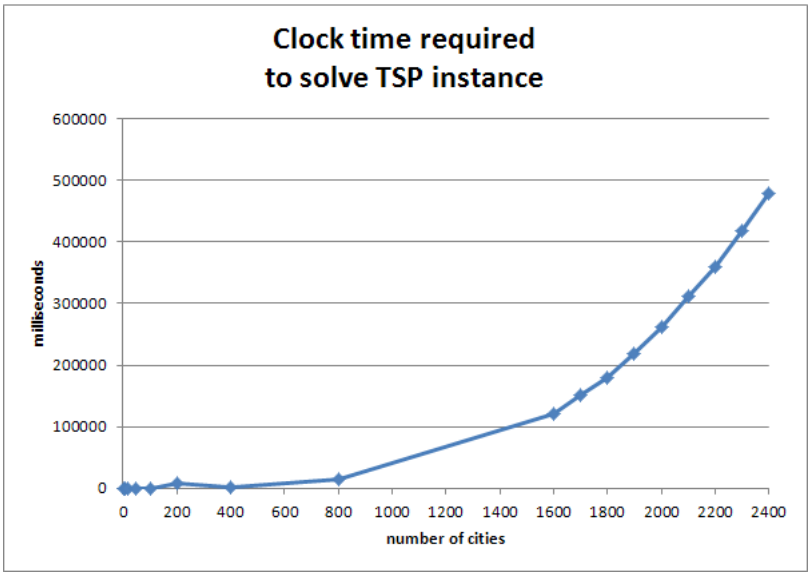

FIG. 5.12: Time required to solve TSP problems of various sizes. The average time to solve 400-city TSPs is less that required to solve 200-city TSPs.

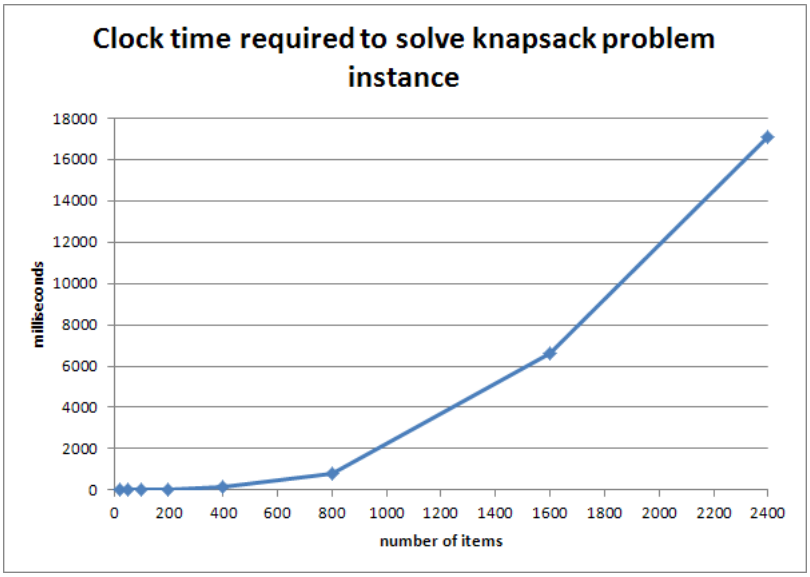

FIG. 5.13: Time required to solve knapsack problems of various sizes.

Chapter 6

CONCLUSION & FUTURE WORK

This dissertation introduced the concept of *plan space analysis* (PSA), specifically the use of Problem-Solution Maps to rapidly allow a system to adjust its plan when it encounters a change in the environment. Ideally, a system would have a library of plans for numerous possible changes in the environment, thereby being able to select one at runtime, rather than replanning from scratch or engaging in online repair. Chapter 3 provided examples of PS Maps, noted that a brute-force approach to creating a PS Map is not feasible, and presented seven algorithms to approximate a PS Map. The chapter also presented a complexity analysis of the algorithms. Chapter 4 described the traveling salesman problem (TSP), knapsack problem, and elevator problem as test domains and presented the results of approximating PS Maps within those domains. The results demonstrate that the utility of the plans given by the approximated PS Map are frequently comparable to the utility of the plans generated with online repair. Chapter 5 discussed approaches to determining the tradeoff between using PSA versus online repair, particularly when considering the time available for PSA calculations. In addition to considering tradeoffs related to timing, the chapter also detailed requirements for representing the problem instances within the problem space. Namely,

- The problem space axes should be selected such that problem instances with identical

solutions are adjacent.

- The solutions should be abstracted in order to create similar solutions, thus allowing for homogeneous solution regions.

In the TSP, knapsack, and elevator domains, using the variable features of the problem instance as axes for the problem space was sufficient to create homogeneous solution regions. However, one could imagine other domains in which there are homogeneous solution regions, but only with respect to a more complex function of the variable features, such as with the example of the game described in the chapter. In that case, axes that considered higher-level or derived features, such as whether the numbers rolled on the dice constitute a pair or straight, would be much more useful than axes based on the numbers themselves.

Abstraction was not required for raw solutions to the TSP and knapsack problems. However, the elevator domain did require abstraction and canonicalization for the solutions to be appropriate for the algorithms.

Chapter 5 also described the tradeoff between expected performance and offline planning time based on examination of problem characteristics.

Some thoughts for extending this work follow.

6.1 Suboptimal Plans

The PS Map map assists in plan library creation by showing the minimum number of solutions required for optimal competency across the problem space. In the case of a 5-city DTSP map, as few as only eight solutions are required, representing fewer than 7% of the 120 (5!) possible solutions. However, for large problems, storing even 7% of the possible solutions may not be feasible. One alternate approach is to accept suboptimal solutions in the library, particularly when one suboptimal plan may replace multiple one or more optimal plans. In this case, this map gives hints about regions in which tolerating a

suboptimal plan over a large region, in place of several plans from smaller regions, may be beneficial in reducing the number of plans in the library.

6.2 Automated Plan Abstraction

Srivastava et al. (2008)'s work describes the process of transforming a plan specific to a single problem instance into a generalized plan that is applicable to more than one problem instance. This is similar to the transformation done within the elevator domain testing, although Srivastava et al. present a general approach in which operation preconditions are examined, thus formalizing the conditions that can be generalized. This approach would be likely be applicable when applying my work to additional planning domains. Within the elevator domain, my approach was to transform steps such as

```
move elevator slow0-0 to floor 2
board passenger p1 into elevator slow0-0
```

to

```
elevator slow0-0 picks up passenger p1
```

This allows the plan to be valid for any passenger location within elevator slow0-0's range. Srivastava et al.'s abstractions would include this level of transformation, and might also consider a further generalization such as

```
elevator slow0-0 picks up a passenger within range
```

or even introduce loops such as

```
for each passenger p within range
  elevator slow0-0 picks up passenger p
```

The primary result would be to create similar solutions, for which the appropriate axes could create homogeneous solution regions. This would also assist with reducing the number of plans to store in the library.

6.3 Analysis of Problem Configuration and Sample Rate

As mentioned in Chapter 5, the ability to estimate the configuration of the solution region would be helpful in determining the appropriate sample rate to increase the effectiveness of the approximation algorithms. Future work could entail finding a correlation between problem domain configuration and the sample rate that should be targeted for a good approximation.

6.4 Sampling-based Motion Planning

In robot motion planning, one way to reduce the computational complexity of path planning is to represent the area of operations as a set of discrete cells and points, called *C-space*. Sampling the operations area will provide a subset of the obstacles that the plan must have the robot avoid, effectively creating a plan with relaxed constraints. A plan that is not feasible with the relaxed constraints can be discarded, and plans that are feasible can be further refined.

The sampling in my algorithms is across full problem instances; the sampling in sampling-based planning is across the constraints of a domain, thus always generating a partially defined problem instance. This approach would be equivalent to adding an additional index to the solution space that represented the constraint. Because the obstacles are simply binary – either the plan will consider the obstacle or it will not – it may be more efficient to use sampling-based planning to sample the 2^n binary combinations rather than adding n additional dimensions. This would support rapid replanning in cases in which an

obstacle appears or disappears during the course of plan execution.

6.5 NASA

Smith (2012) describes a challenge that the Mars Rover scientific team faces in which they must decide on a set of goals for a planner to consider. There are many constraints to consider that would make for a challenging planning problem; however, the key issue is that the scientists do not have a way to evaluate the tradeoffs between the goals they may consider. Smith proposes a solution in which scientists are able to consider a variety of plans from which they could get a sense of what goal combinations are feasible. My work could be suitable for this initial need. However, the second need that Smith describes is plan explanation, in which scientists could ask why one goal is included in the plan and not another, as well as what-if questions that allow them to explore tradeoffs between their goals.

A PS Map for a planning domain shows the set of solutions available for a set of potential changes in the problem space. An interesting extension may be a PS Map that gives information about the set of solutions two steps removed from the current environment. In principle, this could be accomplished by adding axes to the problem space representing all two-hop changes, similar to a TSP PS Map that considers more than one new location. However, in more traditional planning domain, it may be possible to exploit the temporal relationship between two-hop changes to create the map more efficiently.

6.6 Solver Validation

In addition to library generation, the SBE techniques suggest a mathematical framework that proves the solution similarity of groups of problem instances. When comparing approximate maps to the high-quality maps, I found instances of solution variety in re-

Configuration	Unsmoothed	Smoothed
M=24, N=4, 6 slow, 3 fast, 3 variable	1072	506
M=24, N=6, 4 slow, 0 fast, 3 variable	590	198

Table 6.1: Effect on smoothing on PS Maps created by a heuristic solver. "Configuration" refers the elevator domain's M and N parameters, the total number of elevators, and the number of passengers with variable starting positions. "Unsmoothed" and "smoothed" is the number of unique solutions prior to and after smoothing.

gions of the problem space that the SBE technique indicated should be homogeneous. This led me to develop a "smoothing" technique in which I run SSS over specific groups of instances to increase the accuracy of the high-quality maps. This approach could also be used to compensate for the flaws inherent to a heuristic solver based on search. Future work could examine confirming the solution of a given problem intance by also solving problem instances that are similar to it and returning the best solution. How to best mutate the given problem instance to maximize the chance of finding a better solution may be an interesting research question.

6.7 Concluding Thoughts

The challenge of rapidly finding good solutions to complex problems is a theme common to many projects in my workplace. During the course of this work, I have been happy to discover numerous potential applications for some of the ideas presented here. I find myself particularly interested in related problems within the Smart Grid and energy management, and hope to explore solutions to problems in that domain. I hope that the approaches I have developed here may be of some use or inspiration to others encountering these types of problems.

Bibliography

L. Bottou and C.-J. Lin. *Large Scale Kernel Machines*, chapter Support Vector Machine Solvers. MIT Press, 2007.

Justin Boyan and Andrew W. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112, 2000.

J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. *Intelligent Robots and System, IEEE/RSJ International*, 2002.

Blazej Bulka. Analyzing, learning, and shaping planning spaces. In *ICAPS Doctorial Consortium 2006*, 2006. URL `http://www.plg.inf.uc3m.es/icaps06/dc-papers/paper4.pdf`.

Blazej Bulka and Marie desJardins. Useful topological features of planning state spaces. In *The Third North East Student Colloquium on Artifcial Intelligence (NESCAI08)*, Ithaca, NY, May 2 2008.

Ethan Burns, J. Benton, Wheeler Ruml, Sung Wook Yoon, and Minh Binh Do. Anticipatory on-line planning. In *ICAPS*, 2012.

David Chapman. Penguins can make cake. *AI Mag.*, 10(4):45–50, 1989. ISSN 0738-4602.

Thomas M. Cioppa. *Efficient Nearly Orthogonal And Space-Filling Experimental Designs For High-Dimensional Complex Models*. PhD thesis, Naval Postgraduate

School, September 2002. URL http://www.nps.navy.mil/or/thesis2.asp?offset=20&id=34.

G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

Amanda Coles, Andrew Coles, Angel García Olaya, Sergio Jiménez, Carlos Linares López, Scott Sanner, and Sungwook Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33(1):83–88, 2013. URL http://www.aaai.org/ojs/index.php/aimagazine/article/view/2392.

Patrick R. Conrad, Julie A. Shah, and Brian C. Williams. Flexible execution of plans with choice. In *ICAPS*, 2009.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20 (3):273–297, 1995. ISSN 0885-6125. URL http://dx.doi.org/10.1007/BF00994018.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(2):21–27, 1967. ISSN 0018-9448.

Carmel Domshla, Erez Karpas, and Shaul Markovitch. To max or not to max: Online learning for speeding up optimal planning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 1071–1076, 2010.

D. Ferguson, N. Kalra, and A. T. Stentz. Replanning with RRTs. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2006.

Maria Fox and Derek Long. Hybrid STAN: Identifying and managing combinatorial optimisation sub-problems in planning. In *Proceedings of the Seventh International Joint*

Conference on Artificial Intelligence (IJCAI-01), pages 445–450, Seattle, Washington, August 2001.

B. E. Gillett and L. R. Miller. A heuristic algorithm for the vehicle dispatch problem. *Operations Research*, 22:340–349, 1974.

M. L. Ginsberg. Ginsberg responds to schoppers and chapman: Universal planning: an (almost) universally bad idea. *AI Mag.*, 10(4):61–62, 1989a. ISSN 0738-4602.

M. L. Ginsberg. Universal planning: an (almost) universally bad idea. *AI Mag.*, 10(4): 40–44, 1989b. ISSN 0738-4602.

Ramesh Gopal and George Starkschall. Plan space: Representation of treatment plans in multidimensional space. *International Journal of Radiation Oncology, Biology, Physics*, 53(5):1328–1336, 2002.

Jörg Hoffmann. Local search topology in planning benchmarks: An empirical analysis. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 453–458, Seattle, Washington, August 2001.

R. Holder, C. Pascale, M. Dale, R. Daley, E. Chong, V. Shestak, H.J. Siegel, and D. Marinescu. Company Resource Management (CRM) Algorithm Description Document. ARMS Final Report, December 2006.

Robert Holder. Improving a Plan Library for Real-time Systems Using Nearly Orthogonal Latin Hypercube Sampling. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1804–1805. AAAI Press, July 2008. (Student abstract).

Peter Jonsson and Christer Bäckström. On the size of reactive plans. In *AAAI-96 Proceedings*, 1996.

George Konidaris. Autonomous Robot Skill Acquisition. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1855–1856. AAAI Press, July 2008. (Student abstract).

George D. Konidaris and A. G. Barto. Sensorimotor abstraction selection for efficient, autonomous robot skill acquisition. In *Proceedings of the 7th IEEE International Conference on Development and Learning*, August 2008.

Y.-K. Kwok, A. A. Maciejewski, H. J. Siegel, I. Ahmad, and A. Ghafoor. A semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 66:77–98, 2006.

Gilbert Laporte, Michel Gendreau, Jean-Yves Potvin, and Frédéric Semet. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operations Research*, 7:285–300, 2000.

Allan Larsen. *The Dynamic Vehicle Routing Problem*. PhD thesis, Technical University of Denmark, 2000.

S. LaValle and J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5), 2001.

Nikolas List and Hans Ulrich Simon. Svm-optimization and steepest-descent line search. In *In Proceedings of the 21st Annual Conference on Learning Theory (COLT 2009*, 2009.

Brian Logan and Riccardo Poli. Route planning in the space of complete plans. In *In Proceedings of the Fifteenth Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG-1996*, pages 233–240, 1997.

Sean R. Martin, Steve E. Wright, and John W. Sheppard. Offline and online evolutionary bi-directional RRT algorithms for efficient re-planning in environments with moving

obstacles. In *Proceedings of the 3rd annual IEEE Conference on Automation Science and Engineering*, New York, September 2007. IEEE Press.

Stewart Massie, Susan Craw, and Nirmalie Wiratunga. What is cbr competence? In *Poster presentation at the twenty-third Annual International Conference of the British Computer Society's Specialist Group on Artificial Intelligence*, 2003.

M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000. ISSN 0040-1706.

David McSherry. The case-recognition problem in intelligent case-authoring support. In *11th Irish Conference on Artificial Intelligence and Cognitive Science*, pages 180–189, 2000.

David McSherry. Similarity and compromise. In *In Proceedings of the Fifth International Conference on Case-Based Reasoning*, pages 291–305. Springer, 2003. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.97.8116`.

Nilufer Onder and Martha E. Pollack. Contingency selection in plan generation. In *Plan Execution: Problems and Issues: Papers from the 1996 AAAI Fall Symposium*, pages 102–108. AAAI Press, Menlo Park, California, 1996. ISBN 1-57735-015-4. URL `http://citeseer.ist.psu.edu/onder97contingency.html`.

Harilaos N. Psaraftis. Dynamic vehicle routing problems. In B. Golden and A. Assad, editors, *Vehicle Routing: Methods and Studies*. North-Holland, 1988.

Harilaos N. Psaraftis. Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, 61:143–164, 1995.

A. Ram and J. C. Santamariá. Continuous case-based reasoning. *Artif. Intell.*, 90(1-2): 25–77, 1997. ISSN 0004-3702.

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research (JAIR)*, 39:127–177, 2010. URL `http://www.informatik.uni-freiburg.de/~srichter/papers/richter-westphal%-jair10.pdf`.

M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In John McDermott, editor, *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 1039–1046, Milan, Italy, 1987. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA. URL `citeseer.ist.psu.edu/schoppers87universal.html`.

Marcel Schoppers. In defense of reactions plans as caches. *AI Mag.*, 10(4):51–60, 1989. ISSN 0738-4602.

Marcel Schoppers. Estimating reaction plan size. In *Proceedings of AAAI'94*, pages 1238–1244, Menlo Park, CA, USA, 1994. ISBN 0-262-61102-3.

Burr Settles. Active learning literature survey. Technical Report 1648, University of Wisconsin-Madison, January 2010.

David E. Smith. Planning as an iterative process. In *AAAI*, 2012.

Barry Smyth and Elizabeth McKenna. Competence models and the maintenance problem. *Computational Intelligence*, 17:235–249, May 2001.

Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Learning generalized plans using abstract counting. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 991–997, 2008.

Roman van der Krogt and Mathijs de Weerdt. Plan repair as an extension of planning. In *In Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, pages 161–170, 2005.

K. Q. Ye, W. Li, and A. Sudjianto. Algorithmic construction of optimal symetric Latin hypercube designs. *Journal of Statistical Planning and Inference*, 90(1):145–159, 1998.

Shlomo Zilberstein, François Charpillet, and Philippe Chassaing. Real-time problem-solving with contract algorithms. In *Proceedings of IJCAI-99*, pages 1008–1013, 1999.

M. Zucker, J. Kuffner, and M. Branicky. Multipartite RRTs for rapid replanning in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.