Problem Space Analysis for Library Generation and Algorithm Selection in Real-time Systems

Anonymous Author
Anonymous Affiliation

Abstract

Computing solutions to intractable planning problems is particularly problematic within real-time domains. Many visitation planning problems can be mapped to a Dynamic Vehicle Routing Problem (DVRP), in which a system uses a set of vehicles to visit a set of potentially changing locations. One approach to this problem includes off-line computation of contingency plans. However, because complex domains preclude creating a comprehensive library, a system must choose a subset of all possible plans to include. Strategic selections will ensure that the library contains an appropriate plan for encountered situations.

This work proposes a scheme in which problem space analysis drives the creation of an efficient plan library. Problem space analysis can also inform the selection of an appropriate algorithm and its configuration during initial solution generation, and later during any necessary adaptation.

For complex problems, an exact analysis of the problem space is not feasible, and an efficient means of creating an approximation is required. Thus, this work proposes the development of algorithms to both efficiently generate and leverage the problem space analysis of complex planning problems.

Introduction

Computing solutions to intractable planning problems is particularly problematic within real-time domains. Many visitation planning problems can be mapped to a Dynamic Vehicle Routing Problem (DVRP), in which a system uses a set of vehicles to visit a set of potentially changing locations. One approach to this problem includes off-line computation of contingency plans. However, because complex domains preclude creating a comprehensive library, a system must choose a subset of all possible plans to include. Strategic selections will ensure that the library contains an appropriate plan for encountered situations.

Proposed is a scheme in which problem space analysis drives the creation of an efficient plan library. By analyzing the problem space, a minimal library can be created. Minimizing the solutions required to achieve competent coverage of a problem space is well studied within Case-Based Reasoning (CBR) literature. However, here the library size is reduced further by allowing suboptimal solutions for some problem instances. In other situations, solutions that the system can infer at runtime can be removed.

In addition to creating an efficient plan library, problem space analysis can also support algorithm selection and configuration. Algorithms are generally tailored to suit assumptions of particular domain. For example, based on the "regular world" assumption that similar problems have similar solutions, hill-climbing algorithms iteratively modify a given solution to find a better solution. Like hill-climbing, genetic algorithms also improve upon previous solutions, but do so by combining multiple previous solutions to create new solutions that are related, but generally not similar to the old solutions. Specific algorithms may be configured such as modifying the hill-climbing step size or the number of genes a genetic algorithm will mutate. Problem space analysis can inform the selection of an appropriate algorithm and configuration during the initial solution generation, and later during any necessary solution adaptation.

Finally, problem space analysis can assist with problem decomposition. Figure 1 shows the Problem-Solution Map of a one-vehicle VRP, known as a Traveling Salesman Problem (TSP), in which four of the cities have been fixed. Each point represents the possible location of an unknown fifth destination. The color of the point represents the optimal ordering of cities to achieve the shortest path. Figure 6 shows several other configurations of fixed cities. It seems reasonable that the characteristics of some maps would facilitate planning. One approach to DVRPs is to assign sets of cities to a vehicle, creating several TSPs. Problem space analysis could assist with decomposing a large DVRP problem into smaller TSPs in such a way to facilitate planning for unknown future destinations.

For complex problems, an exact analysis of the problem space is not feasible. Thus, in addition to designing algorithms to use problem space analysis, an efficient means of approximating the problem space analysis must be found. Possibilities include uninformed sampling, as well as informed sampling techniques that leverage information acquired in previous samples, such as importance sampling and active learning. Domain-based hints can also assist with generating the problem space analysis. For example, the problem space analysis of DRVPs seems to exhibit a pattern in which the locations of the known destinations suggest boundaries between solution regions. Figure 1 illustrates

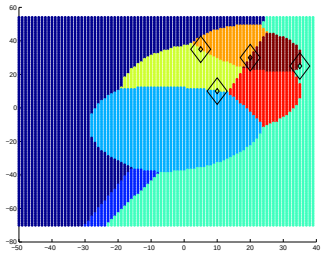

Figure 1: Problem-Solution Map with marked city locations

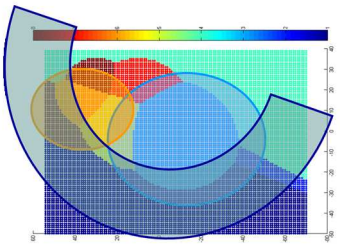

Figure 2: Solution-Problem-Utility Map for 5-city TSP

this correlation between the city locations and the boundaries of the solution regions.

Thus, the proposed research consists of two parts: algorithms that utilize problem space analysis and algorithms to efficiently create the problem space analysis.

Utilization of Problem Space Analysis

The problem space analysis will consist of three maps: the Problem-Solution Map, the Solution-Problem-Utility Map, and the Solution-Similarity Map. The combination of the maps allows the system to generate a plan library, choose appropriate algorithms and configurations, and strategically decompose a problem.

Problem-Solution Map (PS Map)

An example of a PS Map is shown in Figure 1. This map shows the solutions for a set of 5-city TSP problems. Four of the cities are fixed, as indicated by diamonds, and the axes represent the potential locations of the fifth city. The path starts in the middle at (0,0). For each of the possible locations of the fifth city (assuming integer coordinates), the shortest route is generated as the solution. Finally, each unique solution, consisting of a sequence of city identifiers, is assigned a color and plotted. For example, instances of the problem with the solution 0-1-5-3-2-4 might all be colored yellow, as in Figure 4.

This map assists in plan library creation by showing the minimum number of solutions required for optimal competency across the problem space. We see that only eight solutions are required, representing just under 7% of the 120 possible solutions. This is encouraging, but, for large problems, storing 7% of the possible solutions is not feasible. One approach is to accept suboptimal solutions in the library, particularly when one suboptimal plan may replace one or more optimal plans. This map provides information that may be useful to determine regions in which replacing several smaller optimal solution regions with fewer large suboptimal regions may be beneficial in reducing the number of plans in the library, while preserving reasonable plan quality. For example, targeting the smaller regions, such as the narrow blue region and the small orange and brown regions, may be a good heuristic to find solutions to replace.

Solution-Problem-Utility Map (SPU Map)

A notional example of an SPU Map is shown in Figure 2. This map assists in plan library creation by showing where

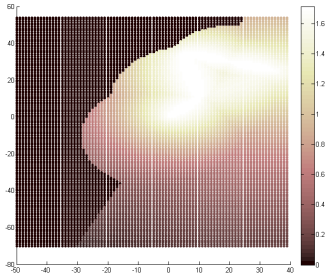

Figure 3: Solution-Problem-Utility Map showing the global competency of one solution. Shade becomes lighter as competence decreases.

tolerating solution utility degradation can reduce the number of plans that the system must store. The expanded regions represent the problem space for which the solution is not optimal, but still tolerable. This example shows that the yellow solution in the middle can be removed from the library, and replaced by components of surrounding solutions. This map is a notional aggregation of a set of SPU Maps. Figure 3 shows an actual SPU map associated with one particular solution. It is generated by coloring each problem instance based upon the loss of competence between the particular solution and the optimal solution. For each problem instance, the system divides the difference in length of the solution path and the optimal path by the length of the optimal path, i.e.

$$competence = \frac{length_{solution} - length_{optimal}}{length_{optimal}} \quad (1)$$

Thus, for any solution-problem combination, the best quotient is 0.0, which is colored black. Higher numbers indicate that the solution path is not optimal, and are displayed by successively lighter colors. In this case, the map suggests that the solution's tolerable scope extends throughout a large part of the solution space. This type of analysis may be particularly useful in reducing the number of stored solutions in high-dimensional problem spaces where large numbers of relatively similar solutions may exist.

Solution-Similarity Map (SS Map)

A notional SS Map is shown in Figure 4. Each solution consists of a city ordering, as indicated in the figure. The length

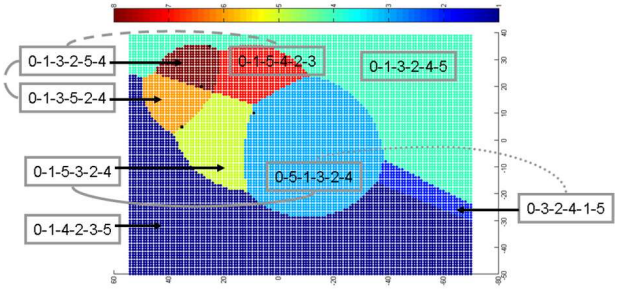

Figure 4: Solution-Similarity Map for 5-city TSP

of the connecting dashes represent the similarity of two solutions, with longer dashes indicating more similarity. This map helps to determine whether a solution can be removed from the plan library in favor of dynamically generating it a runtime. For example, the light blue region (representing solution 0-5-1-3-2-4) and the yellow solution (representing solution 0-1-5-3-2-4) are similar enough (in terms of edit distance) that we may permit a hill-climbing algorithm to adapt one to the other at runtime. Alternatively, these two similar solutions might be combined into one parameterized solution that determines the ordering of city 5 and city 1 as a function of the problem instance.

Algorithm Selection and Configuration

Problem space analysis can give hints to which types of algorithms are appropriate for a particular region of the problem space. When the system encounters a problem that is not explicitly in the solution library, the system can choose the solution of a nearby problem. The system may then choose the type of adaptation based upon its understanding of the local solution region configuration. For example, if the unknown problem is within a homogenous region, the system may decide that no adaptation is necessary. If the unknown solution is in a region of multiple solutions, the system may choose to iterate through each of the nearly solutions to find the best one. If there are too many solutions to make iteration feasible, then the system may choose a solution and hill-climb if nearby solutions tend to be similar to each other, or may use a more perturbing algorithm if the neighboring solutions tend to be dissimilar.

An analogous option is available during the sampling state. If the current solution map suggests a homogeneous solution region, then the system can use that region's solution as a heuristic choice, perhaps followed by adaptation based on neighboring solution similarity as described above. In a heterogeneous solution region, the system can decide to use a more accurate heuristic, or configure an algorithm to be more conservative, such as configuring a hill-climbing algorithm to have a smaller step size.

Generating Problem Space Analysis Maps

Generating maps by solving each instance in the problem space is not practical, so solutions will be generated only for a subset of the problem space. Both informed and uninformed sampling schemes are under consideration. Either type of scheme may be biased towards areas suggested by domain-based hints.

After sampling is completed, PS Map approximations will utilize classification techniques, and SPU Map approximations will utilize regression techniques.

Sampling the Problem Space

The ability to create a generalization from a sample greatly depends upon the sampling scheme. If an area is oversampled at the expense of undersampling a different area, then the generalization has gained no precision from the excess samples in one area, thus they are wasted. At the same time, the approximation of the area that was undersampled will not as precise. Thus, the challenge is to sample the problem space such that the chosen interpolation algorithm will create a reasonable approximation of a complete Problem-Solution Map.

Preliminary work using random sampling schemes show promising results. However, using a technique such as rapidly expanding random trees (RRT) would guarantee a uniform sample of the problem space. Other techniques, such as stratified sampling (McKay, Beckman, & Conover 2000) or nearly orthogonal Latin hypercube sampling (NOLHS) (Cioppa 2002), are also worth considering. In any case, since generating solutions for each sample can be expensive, any technique should consider biasing its sampling technique towards areas of higher interest.

Domain-informed sampling

Additional means of choosing the most useful samples are to take into account known problem characteristics. Figures 1 and 6 show preliminary results of different cases of 5-city TSP problems. In all cases, the fixed city locations hint at borders between region solutions. Given the large amount of heterogeneity in regions near these fixed cities, the ability to identify these regions as candidates for additional sampling seems a promising means of increasing the approximation accuracy.

Interpolating Solutions

PS Map Approximating the Problem-Solution (PS) Map from a sampling of problem instances and their known solutions can be crafted as classification problem: given a set of samples with characteristics and "classified" with a solution, the set of unknown samples can be heuristically associated with a solution. This relies on a "regular world" assumption that similar problems have similar solutions. Early work, using a simple classification scheme based on polling the classified neighbors of an unclassified problem instance, shows promise. However, more sophisticated techniques should be considered. For example, a Bayesian network, support vector machine, or neural network classifier may suffice. Also, blending the sampling and solution generation steps through techniques such as active learning or importance sampling techniques is another consideration. As previously mentioned, exploiting domain information should also increase the accuracy of the map approximation.

SPU Map Approximation of the SPU Map will require a different type of interpolation. In this case, we are attempting use regression to approximate a function representing the utility degradation of a solution when the solution is applied throughout the problem space. Although a non-linear regression could create a single function approximation for the problem space, it is worth considering that the form of the degradation function, the "kernel," may vary in the problem space. For example, the degradation may be linear near the space where the solution is optimal, but degrade exponentially within other regions. Thus, determining the appropriate kernel for the regression as a function of the problem space may be a relevant topic to investigate.

Related Work

Much work relating to solution libraries can be found in Case-Based Reasoning (CBR) literature. Typically, a 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 & Santamariá 1997). In most cases, CBR does not truly preplan; rather, all its solutions are generated during runtime.

Contingency planning generates plans for situations in which a given plan may fail. One classic approach to contingency planning is Schopper's universal plans (Schoppers 1987) in which a solution to every possible situation is stored in a plan library. However, the potential drawback for this technique is the sheer number of states that must be considered. One alternative is to determine the necessary contingencies to plan for by calculating an expected *disutility* for an action that fails (Onder & Pollack 1996). Like CBR, contingency planning is typically used in discrete domains.

Preliminary Work & Results

Preliminary work using random sampling demonstrates that an accurate map can be generated with relatively few samples. In one experiment, the PS Map from figure 1 is sampled at various rates ranging from .0001 to .9, and solution interpolation is done by polling neighbors within a 15-unit radius. Finally, the percentage of correct solutions when compared to the original PS Map is plotted against the sample rate. With a sample rate of .01, the approximated PS Map matches the actual PS Map with .804 accuracy. Sampling at a .0001 rate yields a .544 accuracy. There is a large jump between the accuracy achieved with between very small sample rates and the .913 accuracy at a .05 sampling rate. The results of a follow up three-trial experiment focusing on the sampling rates between .0001 and .01 is shown in figure 5. As promising as these initial results are, even a .001 sampling rate will be unwieldy for a domain with 10^6 problem instances. Hopefully, with intelligent sampling, good accuracy could be achieved with a .0001 sampling rate.

Further exploration of the idea of domain-specific hints informing sampling schemes was done by generating several PS Maps with various fixed city locations as shown in figure 6. In all cases, every fixed city was on a border between two solution regions. Although it is an open question how this might generalize to a higher dimensional problem

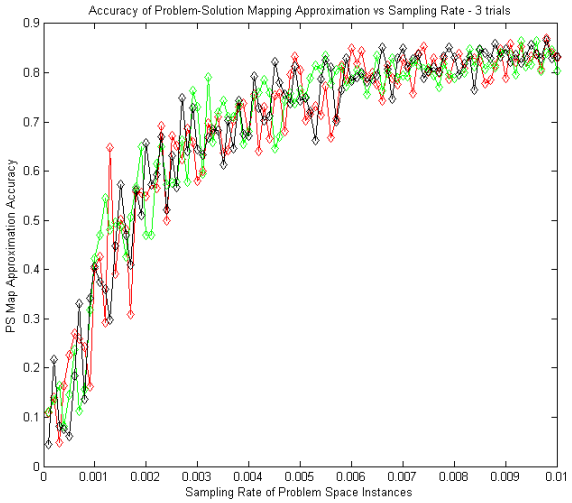

Figure 5: Three trials demonstrating accuracy of PS Map approximation using random sampling and nearest neighbor classification

spaces, obviously clues such as these would be a significant hint towards locations where to focus additional sampling.

Open questions

In spite of the promise of preliminary results, much work needs to be done to extend these techniques to higher dimensions. As problems become more complicated, the dimensionality will explode. Finding an efficient means to accurately approximate the problem space analysis at a low sample rate becomes critical. Although a number of options exist, the appropriate sampling and interpolation schemes for large-scale maps is unclear. Initial results suggest that domain-based hints do exist to inform useful sampling bias, however these hints have not been proven to be consistent or extend to higher dimensions.

Once problem space analysis maps can be generated efficiently, applying them to algorithm selection will require a characterization of algorithms and the impact of various configurations within the chosen domain. It is not clear which features will be most useful for this task. In order to apply these techniques to problem decomposition, a way of comparing the ease of planning with various PS Maps is needed. One approach would be to introduce a problem space complexity metric.

The application of these techniques to other domains is another necessary consideration. For example, a wireless sensor network (WSN) must reconfigure itself to maintain a communications topology as targets move and sensor energy levels fluctuate. A predetermined library consisting of mappings from environment conditions to network configuration could reduce the amount of coordination necessary within the network, thus reducing communications needs and extending the life of the network. The interdependence of individual sensor policies may result in irregularities in

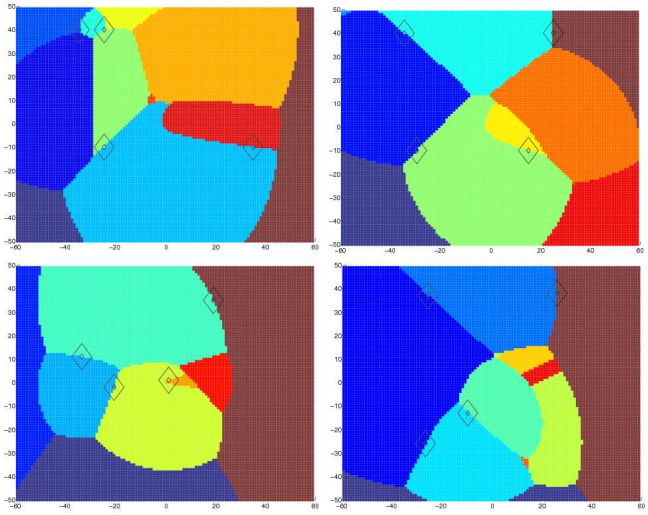

Figure 6: Demonstration of hypothesis of TSP city location indicating solution region boundaries

the problem space, creating challenges when approximating the problem space analysis maps.

Summary

I propose that problem space analysis of real-time planning problems will lend insight to creating plan libraries to be used at runtime, drive algorithm selection and configuration, and supply hints for problem decomposition. It is not practical to generate this analysis by computing solutions to all problem instances in the space. However, it is expected that clever interpolation from a sample of solved problem instances will result in a close approximation to an ideal set of problem space analysis maps.

References

Cioppa, T. M. 2002. *Efficient nearly orthogonal and space-filling experimental designs for high-dimensional complex models*. Ph.D. Dissertation, Naval Postgraduate School.

McKay, M. D.; Beckman, R. J.; and Conover, W. J. 2000. 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.

Onder, N., and Pollack, M. E. 1996. Contingency selection in plan generation. In *Plan Execution: Problems and Issues: Papers from the 1996 AAAI Fall Symposium*, 102–108. AAAI Press, Menlo Park, California.

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

Schoppers, M. J. 1987. Universal plans for reactive robots in unpredictable environments. In McDermott, J., ed., *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, 1039–1046. Milan, Italy: Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.