**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 less cities created fewer, larger homogeneous solution regions and had better PS Map approximation. I would expect, in general, elevator domains with larger N values, that is, larger blocks of floors, 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, be less amenable to approximation by the algorithms.

In general, plan solution spaces have homogeneous solution regions that the algorithms attempt to exploit. These regions could be considered a function of a problem do-

main's objective function, as demonstrated by the justification for SBE. As we've seen with SBE and its skeletal generation of solution region boundaries, a TSP's solution boundaries are defined by each pairwise set of unique solution 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, it is more likely 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 impacts 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 impacts the number of possible unique solutions as well. As the features of the variable item increase, they eventually supersede a new static item, resulting in a new solution. The more static items there are, the more often this occurs, resulting in more solutions, and therefore also increases the number of solution regions. In the elevator domain, the number of blocks of floors impacts the number of solution regions. In general, the steps for a passenger to move from its starting to final destination are a function of 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, which can be shown by observing the incremental changes in the problem instance variable features results in new solutions.

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, this may not be the case. 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 other 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 boolean "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 SBE approach in which objective functions are equated, or SVM+SBE approach could be appropriate. If not, SSS could be a viable alternative.

## 5.2 Tradeoff with Online Repair

These techniques allows 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), with a sample rate:

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

where $time_{prec}$ 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. In the case where there is no knowledge of the length of time until the disruptive event, then the system can define $time_{prec}$ 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 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 fraction utility loss of approximately .02 and mapping that to a SVM+SBE approximation sample rate of .002 in Figure 4.2, we can insert these values to the equation.

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

This results in a $time_{prec}$ 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 4.9, I obtain the equation

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

This results in a $time_{prec}$ 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 5.2 and 5.3 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 increase. In the same way, online repair time for increasing problem complexity would also increase. This again points to a tradeoff between the increasing solve time needed for PSA and the expected decline in performance of online repair.

This leaves the question of how to know what the expected performance is for a particular sample rate. Recalling that the effectiveness of the algorithms appear to be a function of the number of size of homogeneous solution regions, there may be able to consider characteristics of the objection function or the problem configuration to estimate. 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 approximate 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 5.1 shows PS Maps for several randomly configured DTSPs. The unique solutions vary

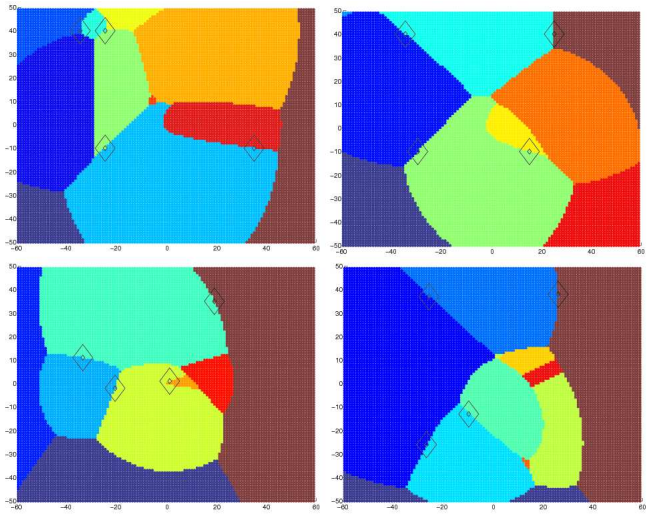

FIG. 5.1: Various high-quality PS Maps of five-city TSPs. Total number of unique solutions varies from eight to eleven.

from eight to eleven.

### 5.3 Solver Validation

In addition to library generation, the SBE techniques suggest a mathematical framework that proves the solution similarity of groups of problem instances. This approach could compensate for the flaws inherent to a heuristic solver based on search. When comparing approximate maps to the high-quality Maps, I found instances of solution variety in regions 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.

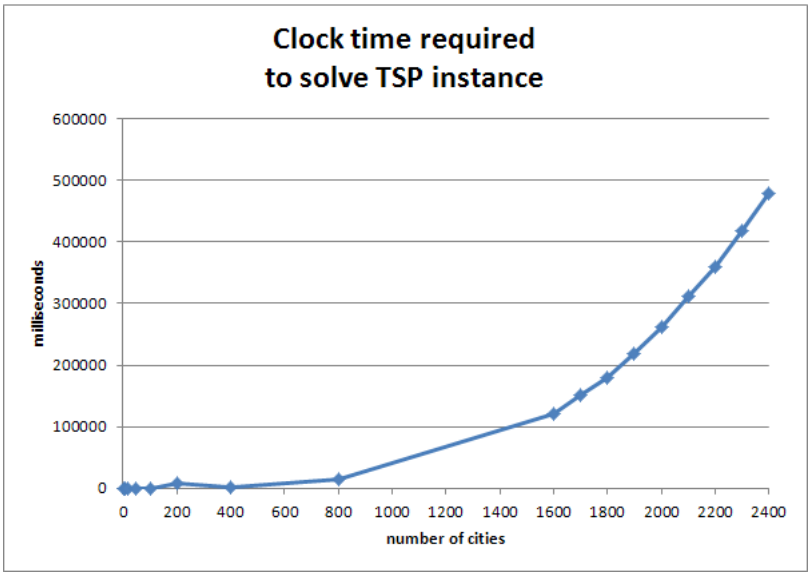


FIG. 5.2: Time required to solve TSP problems of various sizes. The average time to solve 400-city TSPs is less that required to solve 200-city TSPs.

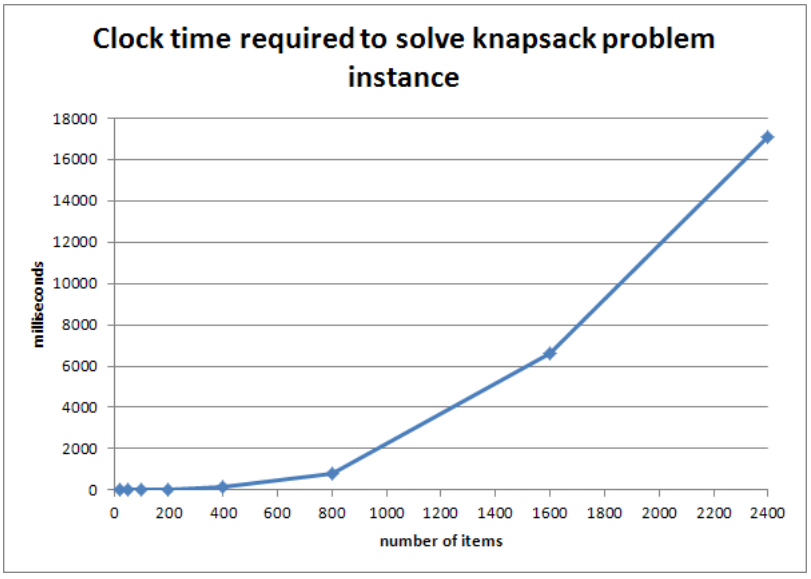


FIG. 5.3: Time required to solve knapsack problems of various sizes.

| **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 5.1: 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.