Chapter 4

EVALUATION IN TEST DOMAINS

I evaluated the approximation algorithms using the Dynamic Traveling Salesman Problem (TSP), a dynamic variant of the 0-1 Knapsack Problem, and an elevator passenger transport domain. The TSP and knapsack domains are NP-Complete problems and translate to a wide variety of practical problems. The elevator domain is a NP-Hard challenge problem developed by Association for the Advancement of Artificial Intelligence (AAAI) for its International Planning Competition (Coles et al., 2013).

Traditional TSP problems consist of a set of unordered locations, sometimes referred to as "cities," that must be ordered such that length of the route 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 initial and desired locations 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 high quality[1] solution. 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 (the result of $\frac{100-95}{100}$). The evaluation of a approximated PS Map is the average utility loss over all its solutions. Thus, the evaluation for a PS Map over all problem instances is $\frac{\sum_{i \in map} optimal_i - approx_i}{\sum_{i \in map} optimal_i}$ where $approx_i$ and $optimal_i$ are, respectively, the utility of the solutions given by the PS Map and a heuristic planner for a given problem instance i. Lower utility loss is preferred; the best approximated solution will have a utility loss of zero.

4.1   Traveling Salesman Problem

For testing in the TSP domain, I generated three instances each of 5, 10, 20, 50, and 100-city DTSPs with one of the cities having variable location. As the gold standard, high-quality PS Maps were generated via the Clark-Wright **?** and Gillett-Miller **?** algorithms, as implemented by the Drasys library **?**. I then removed errors stemming from heuristically-based planning by executing the SSS algorithm over the PS Map. As described in section 3.7, this tests each unique solution against each problem instance, resulting in a more accurate PS Map. The PS Map approximations were generated using nineteen 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

---

[1]As previously mentioned, "high-quality" refers to solutions generated by heuristic search methods. Technically, they are not guaranteed to be optimal and therefore I do not use that term.

technique. Each run was executed ten times.

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 expected of online repair algorithms. This will be discussed in more detail when presenting the experimental results.

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 variant, in which an item may be used a maximum of once. 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 and weight is added to the pool of items.

For the first 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 (100^2) problem instances to generate a high-quality PS Map. A visualization of the resulting two-dimensional PS Map is depicted in Figure 3.2. This represents a best available set of alternate solutions for a system to consider.

The experiments generally focused on the inclusive ranges 1-20 for the weight char-

acteristic and 50-70 for value characteristic[2]. For example, when considering only one additional item, the first problem instance would consider the static items plus an additional item with a (weight, value) pair (1,50), the second problem would consider the static items plus an additional item with weight and value (2,50), and so forth accounting for all possible combinations.

I then generated more complex sets 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 ($20^2 * 21^2$) 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, but with the range of the weight expanded by one unit to 15 to 20, resulting in a problem space of $6^4 * 5^4 = 810,000$ instances.

---

[2]I chose these ranges to avoid large regions of homogeneous solutions that positively skew the experiment results.
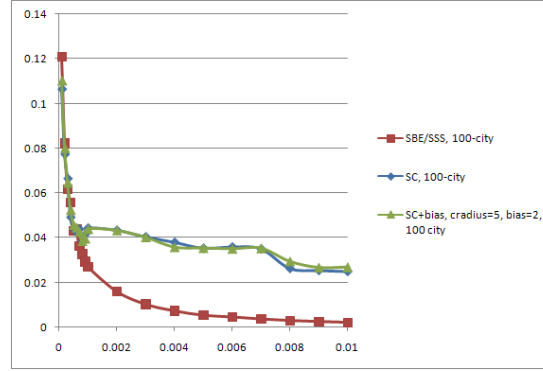
FIG. 4.1: Average utility loss of approximate PS Map generated by various approaches for 100-city DTSP problems

The approximation of all maps was done with the SVM+SBE approximation algorithm with an alpha value of 0.5 with sample rates ranging from .0001 to .001. With the alpha rate of 0.5, 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 optimal solution.

4.3   Elevator Problem

The elevator domain was developed for the 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 with as little cost as possible. 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 are assigned to specific blocks of floors, and the fast elevators, which traverse the entire range of floors, but only stop at block centers and boundaries. For example,

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 over-clothes	43	75
waterproof trousers	42	70

Table 4.1: Knapsack static item pool

42
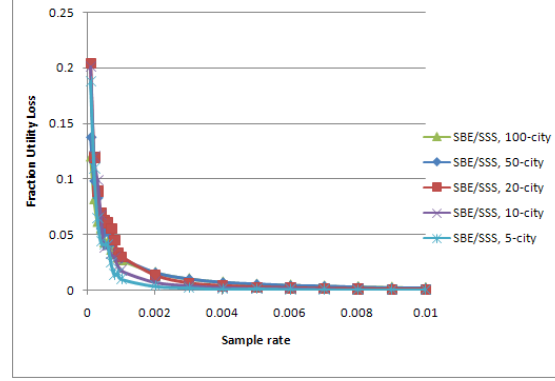
F<small>IG</small>. 4.2: Average utility loss of approximate PS Maps generated by SBE and SSS for DTSP problems of various sizes
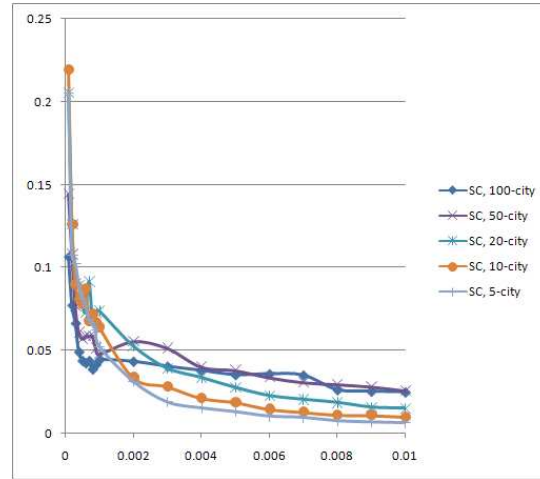

F<small>IG</small>. 4.3: Average utility loss of approximate PS Maps generated by SC for DTSP problems of various sizes
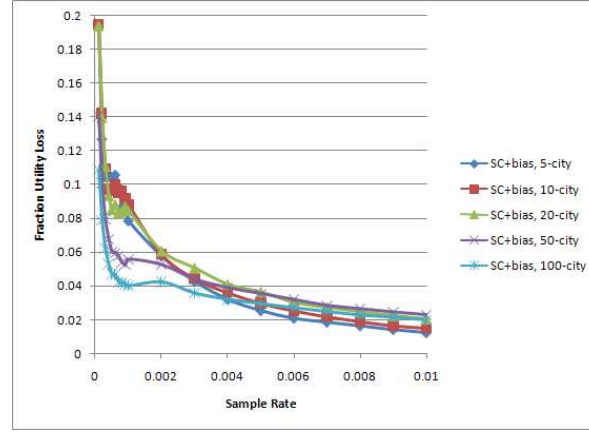
FIG. 4.4: Average utility loss of approximate PS Maps generated by SC+bias for DTSP problems of various sizes
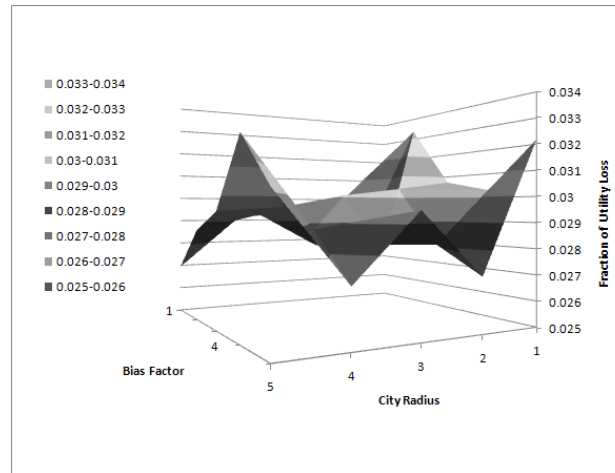

FIG. 4.5: Average utility loss of approximate PS Maps generated by SC+bias for 100-city DTSP problems at sample rate .005. SC-generated PS Maps generated under identical conditions have an average accuracy of .035.
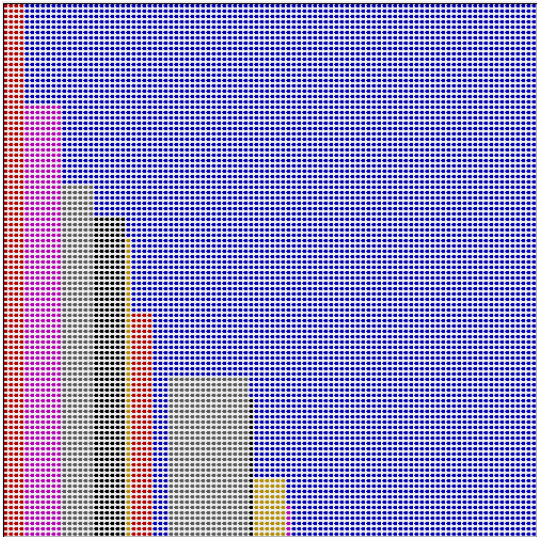
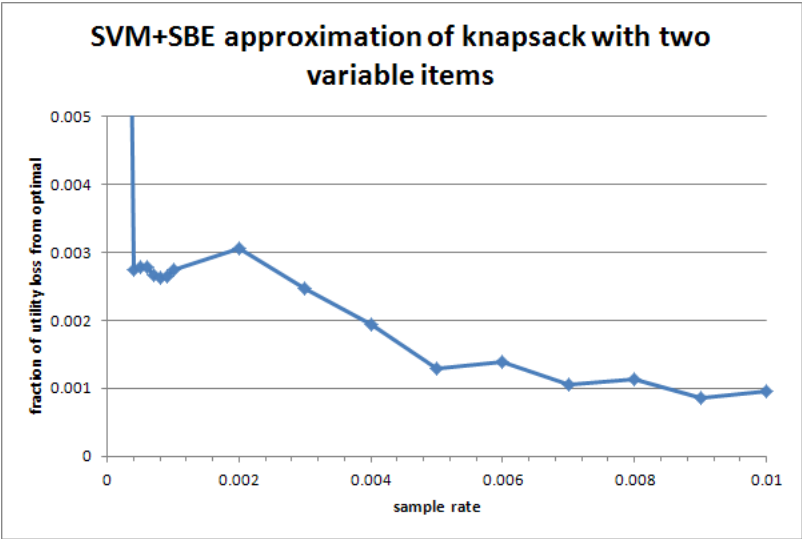FIG. 4.6: High-quality PS Map for a knapsack problem
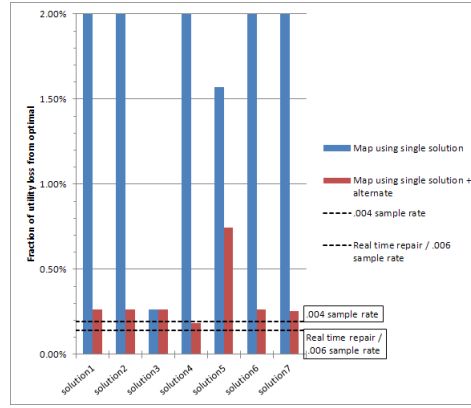

FIG. 4.7: placeholder

FIG. 4.8: 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.
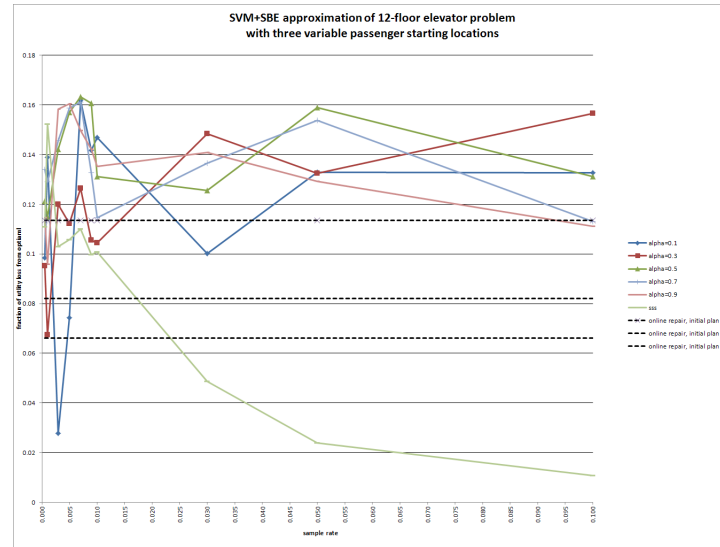

FIG. 4.9: Results of applying the SVM+SBE algorithm to a 12-floor elevator problem consisting of 2 slow elevators, 1 fast elevator, and 3 variable passenger starting locations. Dotted lines represent the utility loss of online repair. Solid lines represent approximations using various alpha values.

FIG. 4.10: Results of applying the SSS algorithm to a 24-floor elevator problem consisting of 4 slow elevators, 6 passengers, and 3 variable passenger starting locations
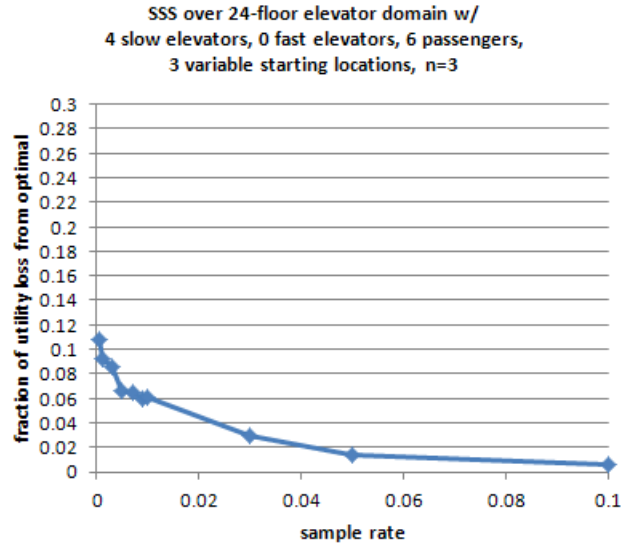

FIG. 4.11: Results of applying the SVM+SBE algorithm to a 24-floor elevator problem consisting of 4 slow elevators, 6 passengers, and 3 variable passenger starting locations

FIG. 4.12: Results of applying the SSS algorithm to a 24-floor elevator problem consisting of 6 slow elevators, 3 fast elevators, 6 passengers, and 3 variable passenger starting locations
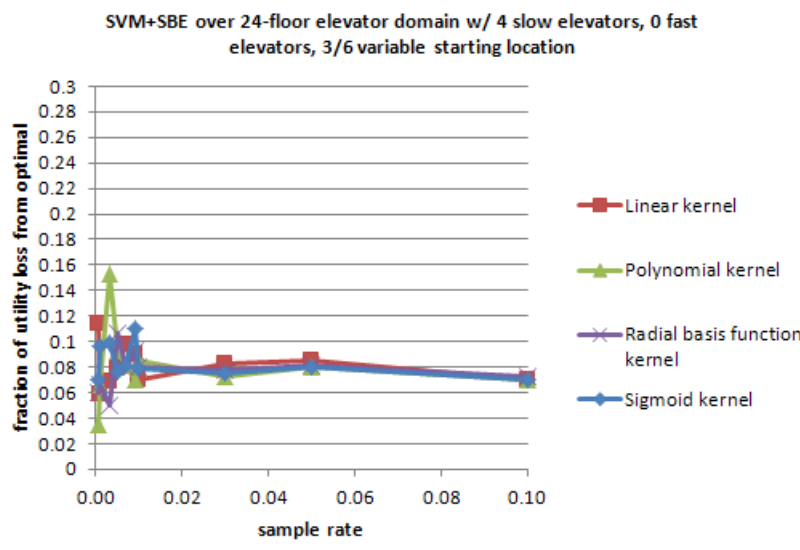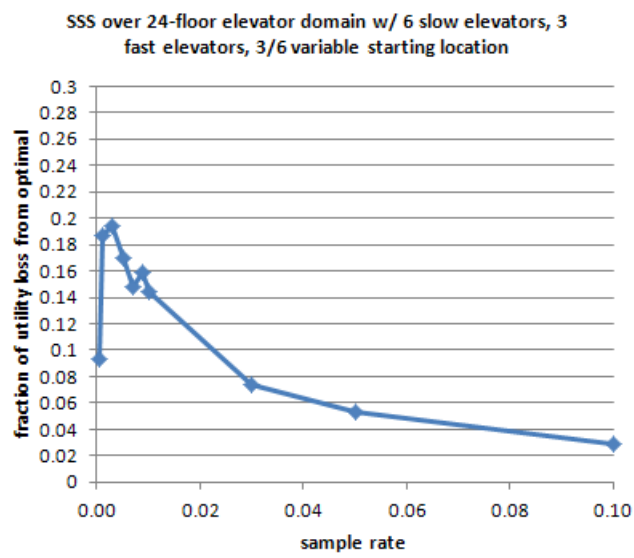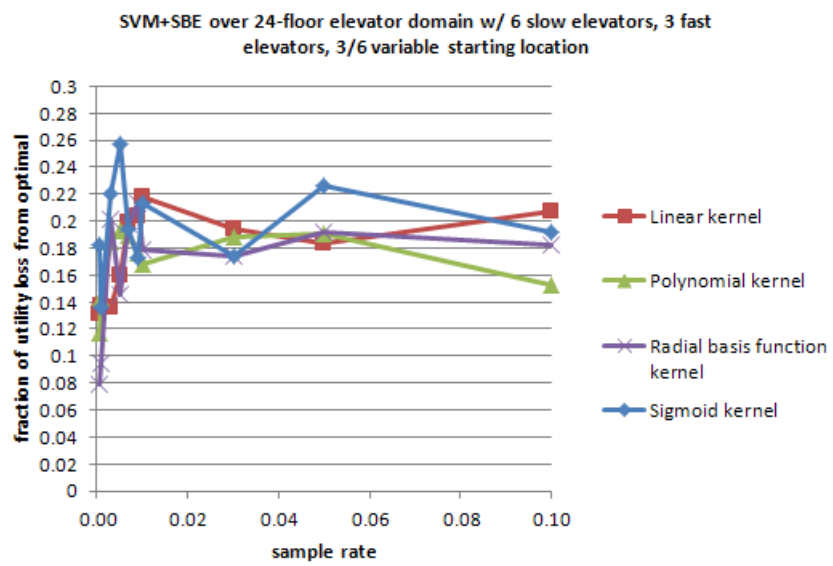
FIG. 4.13: Results of applying the SVM+SBE algorithm to a 24-floor elevator problem consisting of 6 slow elevators, 3 fast elevators, 6 passengers, and 3 variable passenger starting locations

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 an M and N parameter which create a problem domain with M+1 total floors in blocks of N+1 floors, with fast elevators 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 the passengers. My experiments focused on the optimal track and use one of the more successful planners, the LAMA Planner (Richter and Westphal, 2010).

Similar to other hard problems planners in this domain employ heuristics in order to solve these intractable problems, and thereby benefit from "smoothing." This prevents the odd phenomenon of the occasional approximate solution having better utility than the "optimal" solution which may skew the results.

My initial experiment used a 12-floor problem with three passengers, two slow elevators, and one fast elevator. I found that, at least in our initial experiment, that there were a lot of plans for merely a 169-instance problem space (I varied the starting position of two passengers over the thirteen floors). To make this domain appropriate for the algorithm, I generalized the plans such that a plan that moved elevators to a specific floor was transformed into a plan to moved elevators to the location of specific passengers, thus making a plan that could be applied to other problem instances. For example, 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)
```

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.

In order to make this plan reusable, it is transformed into the more general

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

thereby allowing the plan to 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 generalizing 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[3] representing the

---

[3](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)

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.

As a baseline, I implemented an online repair algorithm. **?** describe plan repair as consisting of removing actions from the original plan that 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. **?** 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 refered to in the action. For example, consider an abstracted action "elevator slow0-0 picks up passenger p0"[4] Elevator slow0-0's range is floors n0 through n6, thus if p0's starting position changed to n7 or greater, then the action would be removed.

In the event that an action is removed, then 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

---

[4]The unabstracted actions, for example, "(move-down-slow slow0-0 n6 n0) (board p0 slow0-0 n0 n0 n1)", are less roboust and would require more repair.

two alternatives for continuation: one may either remove all subseqent actions that refer to the passenger and replan the entire route, or one may preserve the subsequent actions and replan the passenger route comply with the constraints implied by the subsequent actions. In the case of the former, I generate a PDDL file indicating a problem that solely focuses on the passengers whose actions were removed. In order to plan without 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 normalize actions
3: Add actions to P
4: normalize P

4.4   Results

4.4.1   TSP

The results for the 100-city TSP runs are displayed in Figure 4.1. Each accuracy measure is the average over the problem space of the fraction of utility lost by using the

solution retrieved from the approximate PS Map instead of the solution from the ideal PS Map. For example, an accuracy measure of .02 indicates that, on average, the approximated map returns a solution that is 2% worse than the ideal map. As previously mentioned, SBE and SSS generate identical PS Maps within this domain, and are represented with the same line.

I focus primarily on 100-city problem instances since they are most representative of a complex domain. However, Figures 4.2, 4.3, and 4.4 display the accuracy of the PS Map generated by SBE/SSS, SC, and SC+bias, respectively, for DTSPs of various route lengths.

To compare our approaches with traditional online plan repair, I used a nearest neighbor DTSP solver to insert the variable city into the route. The average difference between the heuristic and ideal plan was 1.97%, approximately equal to the SBE performance at a .002 sample rate. Other work comparing heuristic and optimal plan repair Larsen (2000) suggests that a 2-6% utility loss is typical.

Lastly, I tested the accuracy of SC+bias approximations of 100-city DTSP PS Maps at sample rate .005 with varying bias factor and city radius parameters, the results of which are shown in Figure 4.5. As comparison, the average accuracy of SC-generated approximate PS Maps for 100-city routes at a .005 sample rate is .035.

4.4.2   Knapsack

The results of generating a high-quality PS Map are 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 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 SVM+SBE 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, for the purposes of computation time required to complete the experiment. The graph shows a loss of utility well under 1% at low sample rates.

Figure 4.8 puts these results in context against various baselines. The taller blue bars represent the fraction of utility lost assuming a PS Map consisting of a single solution. Because the high-quality PS Map had seven solutions, there are cases represented in the graph. The shorter red bar represents applying a 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. The upper dotted line represents the fraction of utility loss 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. The online repair method is a greedy solver that selects the items with the highest value:weight ratios. This demonstrates that the performance of SVM+SBE algorithm when sampling at at rate of .006 is roughly equivalent to that of the online repair technique.

4.4.3 Elevator

Results from the 12-floor elevator problem domain are displayed in Figure 4.9. 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.

Results from the 24-floor elevator domain experiments are shown in Figures 4.10 through 4.13. These results show that for each problem configuration SSS performs better than SVM+SBE. Additionally, the algorithms perform better against the problem configuration with four slow elevators (4s) than the configuration with six slow and three fast elevators (6s-3f). Both of these results seem reasonable given the nature of the problem space of each configuration and the algorithms used. In all the configurations, the problem spaces have homogenous regions, but they are small, which can make it difficult for an SVM-based algorithm to converge and find the appropriate boundaries. However, this is 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 four slow elevators. In the 4s configuration, the homogenous regions are larger than in the 6s-3f problem space, and thus the SVM+SBE algorithm performs better. Because there are less totxal solutions in 4s, the SSS algorithm performs better as well.

4.5   Analysis & Discussion

Our results show little difference between approaches at low sample rates, but whereas SC plateaus at just above .02 accuracy, SBE/SSS continues asymptotically towards perfect accuracy. In all approaches, there is high volatility at low sample rates, likely due to the PS Map approximation being sensitive to a small number of random samples.

The empirical performance as a function of route size show that SBE and SSS are much less sensitive to the complexity of the problem instance than other approaches. This is

reasonable, given that DTSPs with more locations have more solution borders, and the limitation of SC methods is that they have more difficulty determining the location of solution borders. At greater sample rates, SBE and SSS provide significantly better performance than nearest-neighbor solutions.

The results for the SC+bias approach suggest that the attempt to mitigate the limitations of SC is generally successful. The accuracy of SC for 100-city routes at a sample rate of .005 is .035, and the accuracy of PS Maps generated with various SC+bias parameters varies from .027 to .033. These are positive results, but there is no obvious pattern to determine the optimal parameters.