An Effort Bias and Cost Pruning for Anytime Sampling-based Motion Planning

Tianyi Gu

Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA

Abstract

We present a new anytime motion planning approach called B-SST. B-SST first runs BEAST, an effort-aided planner, to find a first solution as quickly as possible, then switch to another motion tree growth process called SST-with-cost-pruning, which adopt both idea from SST and cost pruning algorithms. We first introduce several related work that we build upon. Then B-SST is described in detail. Results with a variety of vehicles and environments showed that B-SST is competitive compared to A-BEAST and other successful anytime planners. We also discussed a more sophisticated idea on how to create a better anytime motion planner in the end.

Introduction

Given a robot with a description of its dynamics, a description of the obstacles in the workspace, motion planning is the problem of taking that robot from a starting state to a goal region without intersecting obstacles. Recent advances in sampling-based optimal motion planning build on decades of work in the topic of feasible motion planning, in which costs are ignored. However, solutions from these methods can be unnecessarily long due to the sampling process. As robots become more mobile, it is important to find not only a feasible plan but also improved ones when there is enough computational time. The strategy most planners take in this situation is to find a solution as quickly as possible and improve that solution as time permits.

Recently, an algorithm called Bayesian Effort-Aided Search Trees (BEAST) (Kiesel and Ruml 2016), is developed in UNH, which biases tree growth through regions in the state space believed to be easy to traverse. Beast is shown that it can find solutions fast than other planners. An anytime beast algorithm A-BEAST (Kiesel 2016) is built upon beast but it is really complicate and also performs not good enough compared to existing anytime motion planning algorithms. In this project, we simplify current A-BEAST algorithm and invent a new algorithm called B-SST, which rely on beast to find the first solution and then improve the current solution by the idea of a general cost pruning algorithm (Hauser and Zhou 2015) and SST (Li, Littlefield, and Bekris 2015). We will introduce these related algorithms in the next section and present our work in section 3.

Experiments with 4 type of vehicles including kinematic car, dynamic car, hovercraft, and quadrotor operating in 5 environments including 3-ladder, parking lot, intersection, single wall, and forest demonstrate the efficiency of B-SST. Results show that B-SST is competitive compared to A-BEAST and other successful anytime planners.

Related Work

There has been much related work on sampling-based motion planning. Some of them are focusing on feasibility, meaning trying to find a feasible solution, while other of them are working on optimality, meaning trying to find a optimal solution. Since our work is built on both of this two type of algorithms, we introduce several leading algorithms of each type.

RRT

Rapidly-exploring random tree (RRT)(LaValle and Kuffner 2001) is the ancestor for all sampling-based motion planning algorithms. It grows a motion tree rooted at the starting point by using random samples from the whole work space. When the tree touch the goal region, it then can reconstruct the motion trajectory. With uniform sampling of the search space, the tree may eventfully fill up the whole area which is not a efficient strategy. So there are many researches working on helping RRT focus on useful exploration by increasing the probability of sampling states from a specific area, e.g., KPIECE(Şucan and Kavraki 2009) and P-PRM(Le and Plaku 2014).

P-PRM

P-PRM(Le and Plaku 2014) combines sampling-based motion planning in the state space with discrete search over a probabilistic roadmap abstraction. Probabilistic roadmap abstraction is constructed over a low-dimensional configuration space obtained by considering relaxed and simplified representations of the robot model and its feasible motions. The edges in abstract graph are collision-free. It then run a Dijkstra algorithm to find the shortest path from the starting point to goal region and use this path as a heuristic to bias sampling.

It searches by maintaining a queue of abstract vertices in the graph sorted by increasing scores. At each search itera-

tion a abstract vertex c is selected based on the corresponding heuristic cost and the number of times it has been selected for expansions in the past. An abstract vertex along the cheapest precomputed path from c to abstract goal vertex is chosen. This vertex is then used to create a random concrete state p within some pre-specified state radius. This is now the "target" state used similarly to when plain RRT chooses a state uniformly at random. The only difference is that instead of choosing in the whole motion tree, it choose the nearest state in the state group of c as the root for the new propagation which is steered (if possible) toward the random state. Any new abstract vertices touched by the propagation attempt are added to the queue if not previously enqueued.

P-PRM tries to pursue the completion of low cost paths by following its heuristic estimates in the abstract space. It tries to avoid getting stuck during planning by penalizing the score of abstract states after they are examined.

BEAST

Bayesian Effort-Aided Search Trees (BEAST) (Kiesel and Ruml 2016) is a novel method that tries to find solutions as quickly as possible by constructing solutions which it estimates require the least effort to build. Building a motion tree will vary in difficulty across different regions of the workspace. Recognizing this, BEAST reasons about effort using a discrete abstraction of the state space, in the form of a directed graph of regions. As it grows the motion tree, it maintains for each edge in the graph an estimate of the effort that would be required to grow the tree between the corresponding abstract regions of the state space. At each iteration, it allocates its effort to growing the tree through the region of the state space that is estimated to allow the least total effort (propagation attempts) before the motion tree reaches the abstract goal region.

Cost Pruning

A general cost pruning method (Hauser and Zhou 2015) is a simple anytime motion planning algorithm which can find a optimal solution asymptotically. They use a new state-cost space formulation to transform optimal motion planning problems into feasible kinodynamic motion planning problems. Then a meta-algorithm, AO-x, is introduced to adapt any feasible kinodynamic planner x into an asymptotically-optimal motion planner by generating a series of feasible trajectories in state-cost space with progressively lower costs. Based on this idea, it is able to prune any extensions to the motion tree that exceed the cost of current incumbent.

SST

Stable sparse RRT(SST) (Li, Littlefield, and Bekris 2015) is another motion planning algorithm that find optimal solution. It try to detect and remove duplicate in motion tree as the tree growing. When a new tree branch enters a duplicate detection scope, it is checked against other previous branches residing in that scope to determine dominance. The dominant branch inside a scope is the only branch propagated from during tree growth.

B-SST($Abstraction, Start, Goal, \delta_s, \delta_v$)
1. Incumbent=**INFINITY**
2. MotionTree.Insert(Start)
3. IsFindFirst=**FALSE**
4. AbstractStart = Abstraction.Map(Start)
5. AbstractGoal = Abstraction.Map(Goal)
6. Abstraction.PropagateEffortEstimates()
7. Open.Push(AbstractStart.GetOutgoingEdges())
8. **While** Have More Time
9. **If** IsFindFirst = **FALSE**
10. BEAST($Open, MotionTree, IsFindFirst$)
11. **Else**
12. SSTWithCostPruning($MotionTree$, $Incumbent, \delta_s, \delta_v$)
13. **If** Reach The Goal
14. Incumbent = Solution.Cost

Figure 1: Pseudocode for the B-SST algorithm.

A-BEAST

A-BEAST (Kiesel 2016) is the anytime version of BEAST. The author add three additional components to BEAST. First a cost pruning function (idea from (Hauser and Zhou 2015)) is used to ensure it will not add any state that exceed the cost of current incumbent in to the motion tree. The second modification is instead of growing a simple motion tree, it maintain a spares motion tree by filtering duplicate states in same region, using the idea of (Li, Littlefield, and Bekris 2015). By doing this, as the tree growing, it can continually reduce cost in the motion tree. Lastly, for each edge, it not only consider an effort estimate but also a cost estimate. Then the idea from Anytime EES (Thayer, Benton, and Helmert 2012) is emulated to searching for the lowest effort solution with a cost estimate lower than the current incumbent. However, it is really complicate to understand the way they reasoning about the cost and hard to implement and also performs not good enough compared to existing anytime motion planning algorithms. So our work here is based on A-BEAST and to provide a much simpler algorithm called B-SST which is still competitive compared to A-BEAST.

B-SST

B-SST is a new algorithm that tries to combine the idea of BEAST, SST and cost pruning. At beginning, We use BEAST to propagate the motion tree, just like what A-BEAST do. The difference is that after it find the first solution, B-SST switch to another motion tree growth process called SST-with-cost-pruning and try to keep improving the solution.

The pseudocode for B-SST is present in Figure 1. The algorithm begins by initializing a cost incumbent to infinity (line 1), inserting the start state to motion tree (line 2), and propagating effort estimates through the abstract graph outward from the region containing the concrete goal state (line 6). The algorithm then pass the edge open list and the motion tree to BEAST algorithm in line 10 and do exactly

```
BEAST(Open, MotionTree, IsFindFirst)
15.    Edge = Open.Pop()
16.    StartState = Edge.Start.Sample()
17.    EndState = Edge.End.Sample()
18.    ResultState = Steer(StartState, EndState)
19.    MotionTree.Insert(ResultState)
20.    Success = Edge.End.Contains(ResultState)
21.    If Success
22.       Edge.UpdateWithSuccessfulPropagation()
23.       If Edge.End == AbstractGoal
24.          Open.Push(GoalEdge)
25.    Else
26.       Edge.UpdateWithFailedPropagation()
27.    Abstraction.PropagateEffortEstimates()
28.    Open.Push(Edge)
29.    If Success
30.       Open.Push(Edge.End.GetOutgoingEdges())
31.    If Goal(ResultState)
32.       IsFindFirst=TRUE
```

Figure 2: Pseudocode for the BEAST algorithm.

```
SSTWithCostPruning(MotionTree, Incumbent, δs, δv)
33.  ResultState = SST(MotionTree, δs, δv)
34.  If ResultState.Cost < Incumbent
35.     MotionTree.Insert(ResultState)
```

Figure 3: Pseudocode for the SSTWithPruning algorithm.

what BEAST do. After BEAST find the first solution, it will switch to another motion tree growth process called SST-WithCostPruning (line 12). Every time it find a new solution, the cost incumbent will be updated (line 14).

The pseudocode for BEAST is present in Figure 2. A D* Lite (Koenig and Likhachev 2002) algorithm is used to compute the effort-to-go value for each vertex. BEAST considers which edge in the abstract graph represents the best way to attempt to grow the motion tree. It maintains a open list of edges whose source regions have been touched by the motion tree and is kept sorted in increasing order of total estimated effort. In each iteration, BEAST pops the edge off open with the lowest estimated effort. Then an existing concrete state in the edge's source region is chosen to propagate from. A new concrete target end state is generated from the edge's abstract end region. Finally, an attempt is made to grow the tree from start sate to end state using a steering function. In line 19, the result state is inserted into the motion tree. After the propagation, it update the effort estimate for the edge by maintenance of a beta distribution. When BEAST find a solution it will update the global IsFindFirst boolean variable to TRUE (line 32).

The pseudocode for SSTWithCostPruning is present in Figure 3. It first try to propagate by SST (Li, Littlefield, and Bekris 2015) in line 33. After getting a new result state, it check the state by current incumbent (line 34). If its cost is exceed the current incumbent, we just throw it away, else it will be inserted into the motion tree.

Experiments

We compared B-SST with BEAST, A-BEAST, Restarting-RRT-with-Pruning, and SST.Implementation of A-BEAST, Restarting-RRT-with-Pruning, and SST are from A-BEAST paper (Kiesel 2016). Experiments also used OMPL's implementation of a Kinematic Car, Dynamic Car, and Quadrotor vehicle, as detailed in A-BEAST paper. All the vehicle meshes are presented in Figure 4 panels (a) and (b). The environment meshes used for the experiments are presented in Figure 4 panels (c)-(g).

For each vehicle, 5 start and goal pairs were used, and for each start and goal pair 25 different random number generator seed values were used. This provided 125 runs for each of the domains. The start states were biased toward the center of the workspace while the goal was biased toward the lower center of the workspace. This set-up tends to generate problems in which the optimal solution threads its way carefully between the obstacles, but it is much easier to take a more costly route around the obstacles. This wide diversity of planning time/solution cost trade-offs directly tests the ability of anytime motion planner to estimate planning effort and adjust its behavior accordingly. A motion planner that explicitly tries to find plans quickly ought to exhibit superior performance. A planning timeout of 300 seconds was used.

Results

The first plot in each series is instance coverage as a function of time. This is a very important metric to exemplify how quickly algorithms are able to solve all instances within the set.

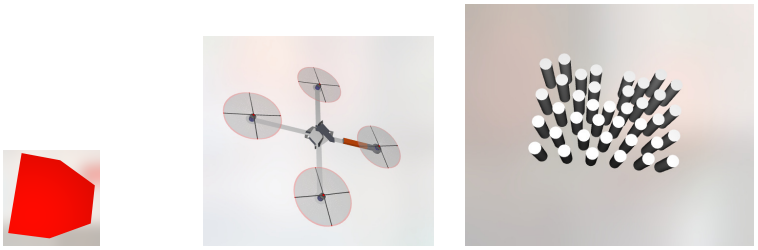
The next plot is solution cost as a function of time. At a given time point, if an algorithm has not solved an instance, the solution cost for that instance is considered to be a large finite constant: 100,000. Using a large finite constant rather than infinity allows us to compute a finite average across the x-axis. This leads to algorithms not being shown on the plot until they have provided at least one solution for every instance in the set.

The next plot is Goal Achievement Time (GAT) as a function of time (Kiesel, Burns, and Ruml 2015). This metric is how quickly a goal is actually achieved after the instance was issued. It is the sum of planning time and execution time. This is a very practical metric when a solution will actually be executed. Minimizing planning time can result in very long solutions, while minimizing solution cost can result in very long planning times. These plots are targeted to reward a balance between these two quantities.

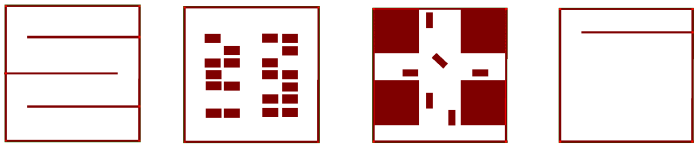
The last plot in the set is an anytime solution quality plot which is the IPC Anytime Metric. These plots show solution quality as a function of time. They attempt to reward coverage and low solution cost by rolling these values together. This type of plot is used by the International Planning Competition and has become their traditional anytime plot.

Kinematic Car

Figure 5 through Figure 9 present the results of B-SST in the kinematic car domain over each of workspace in Figure 4.



(a) cars and hovercraft (b) quadrotor (c) forest



(d) 3 ladder (e) parking lot (f) intersection (g) single wall

Figure 4: The vehicles and maps used in the experiments.

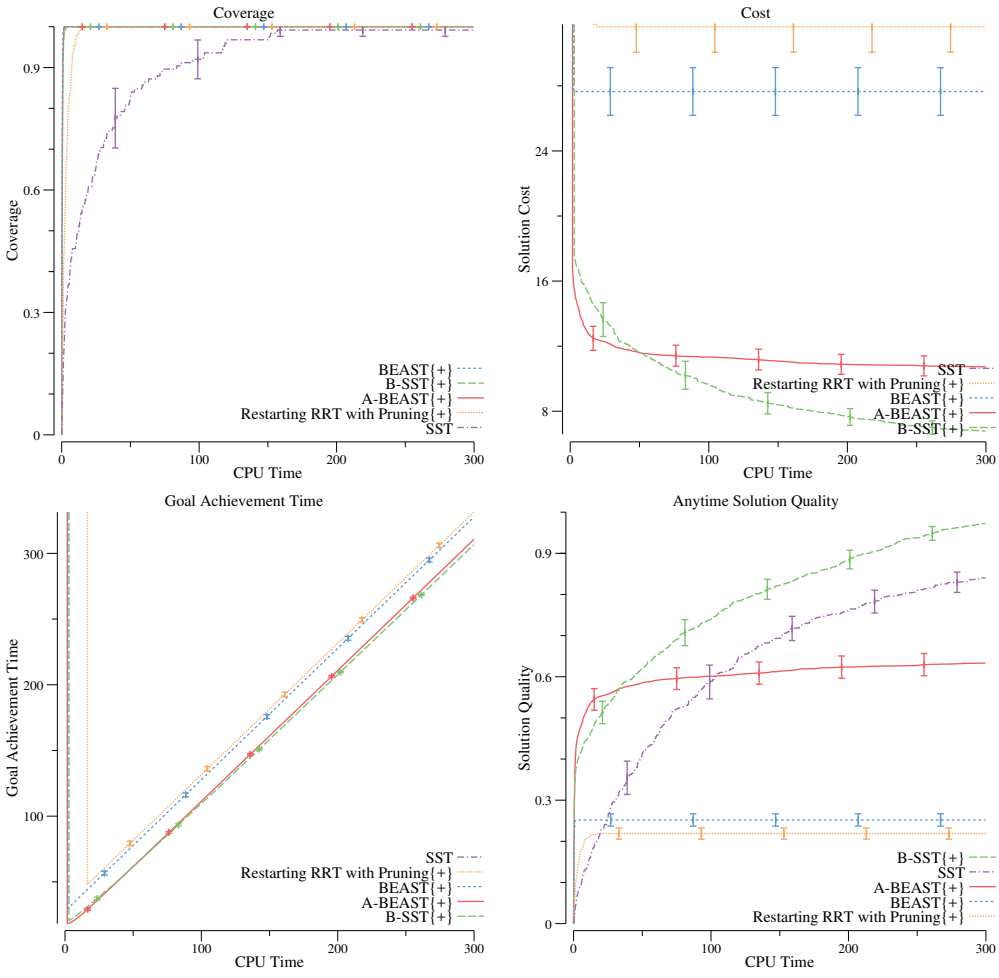


Figure 5: Kinematic car results in the forest workspace.

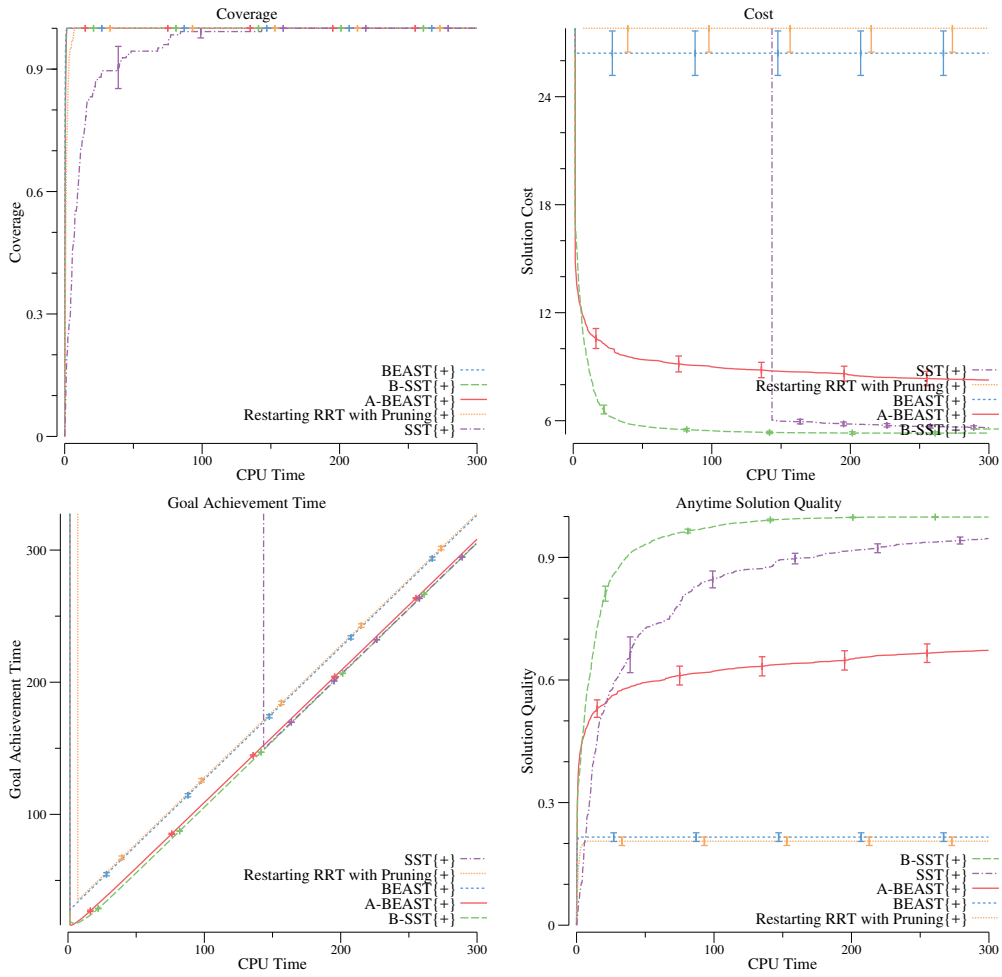


Figure 6: Kinematic car results in the single-wall workspace.

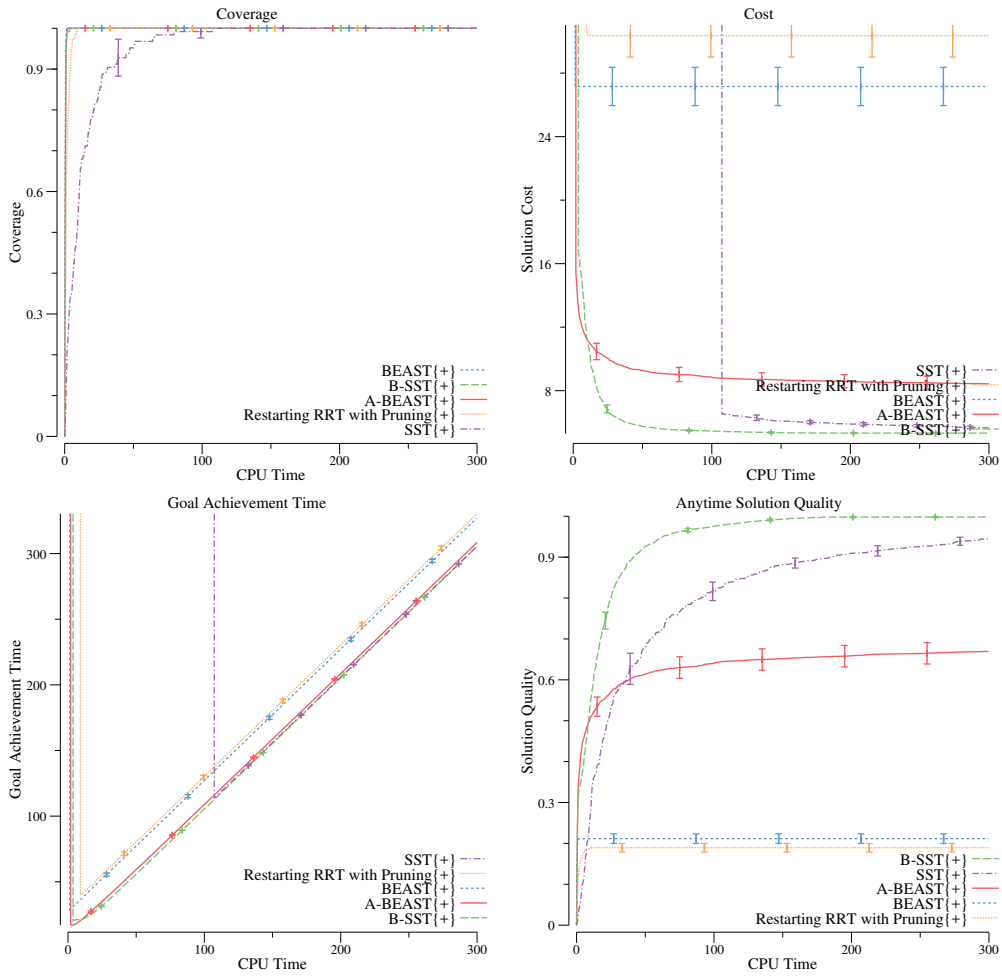


Figure 7: Kinematic car results in the 3-ladder workspace.

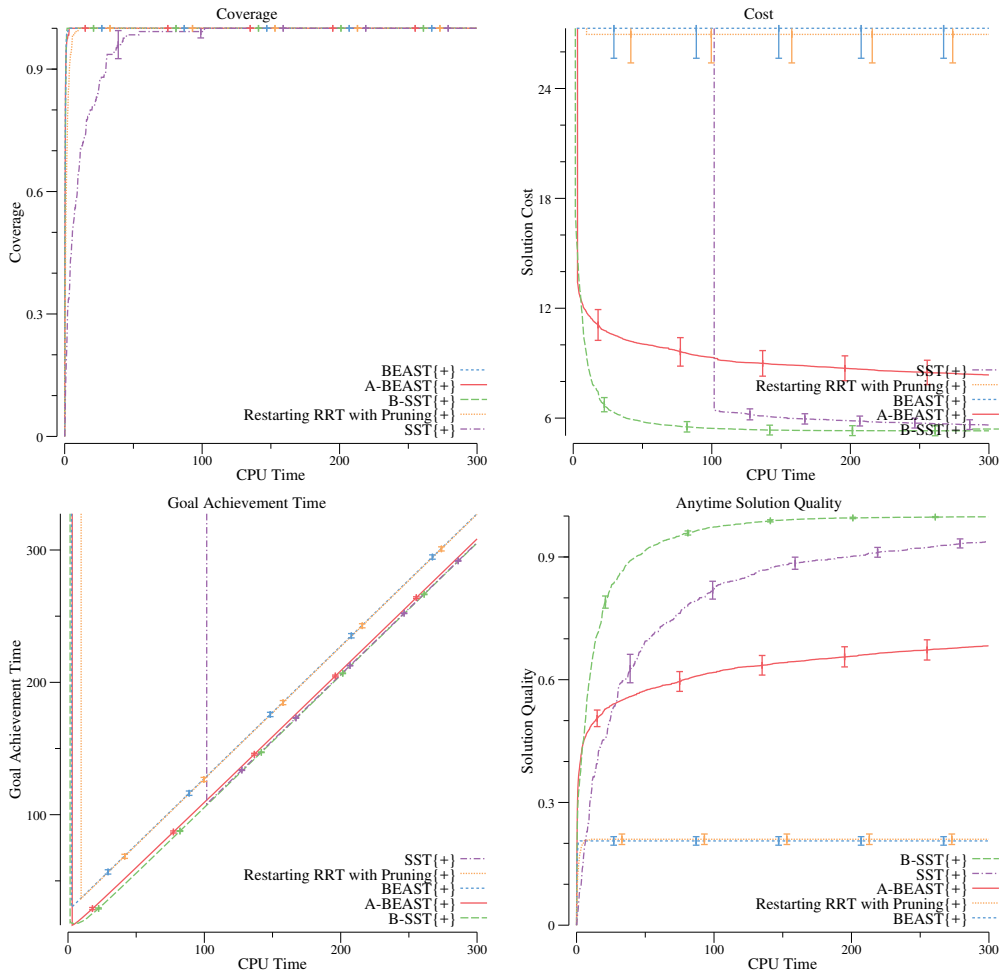


Figure 8: Kinematic car results in the parking-lot workspace.

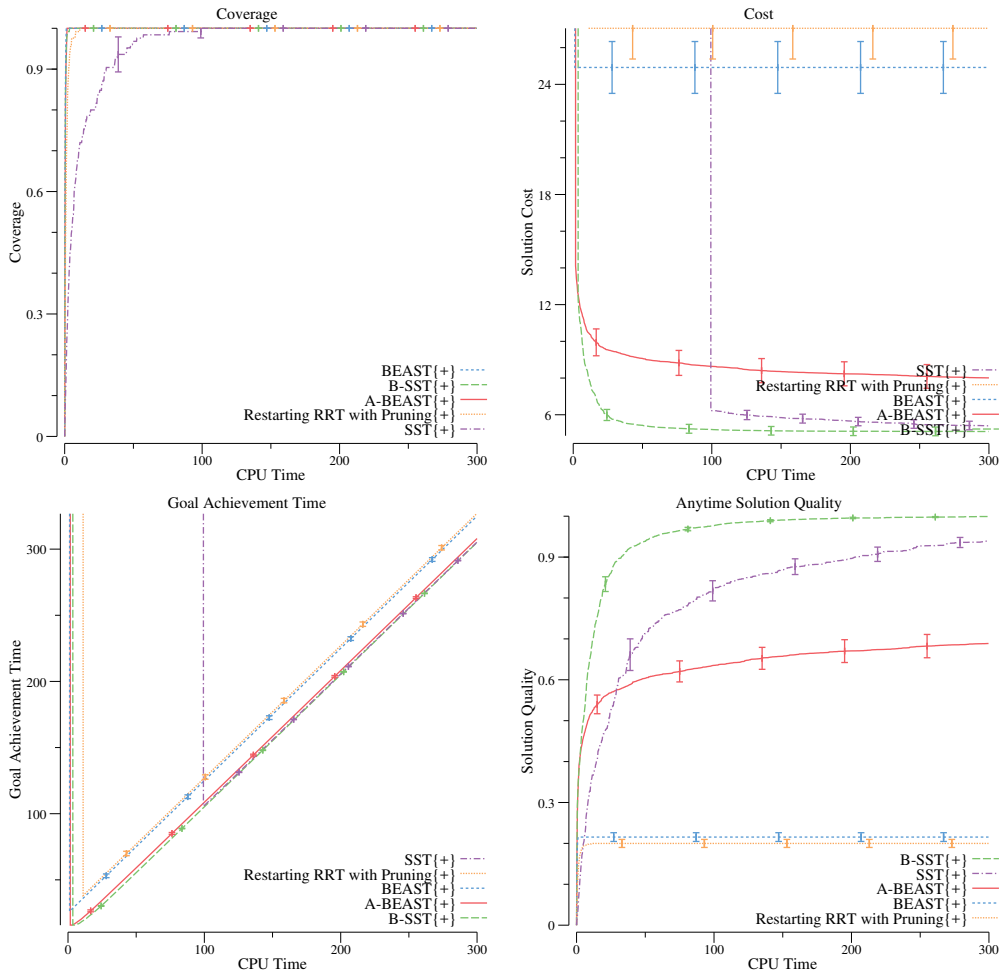


Figure 9: Kinematic car results in the intersection workspace.

As shown in coverage plot, B-SST immediately solve all instances within teeny time. BEAST and A-BEAST are also able to achieve this. SST fail to solve all the instances in forest environment.

As shown in the cost plot, B-SST is always able to find cheapest solution very fast. While A-BEAST is as fast as B-SST, its solutions are more expensive. Although SST is also able to find cheap solution close to B-SST's, but it need significantly more time.

In terms of the average goal achievement time across the planning duration, the B-SST as well as A-BEAST are the two best performers.

In the Anytime Solution Quality plots, B-SST perform best, followed by SST and A-BEAST. This is not too surprising because the solution quality metric has to do with factor of best solution found and B-SST eventually find the cheapest solutions in the planning duration.

Dynamic Car

Figure 10 through Figure 14 present the results of B-SST in the Dynamic car domain over each of workspace in Figure 4.

It is very embarrassed that all algorithm perform surprisingly bad in both forest work space and 3-ladder workspace. Our explanation to it is either the machines are interrupted or we messed up some parameters in problem setting. We will look into this in the future. In despite of this, we can still look at rest of the workspaces.

As shown in all of the 4 plots, B-SST performs as well as A-BEAST do. However in Dynamic car domains, SST is able to find cheaper solution as we give it more time.

Hovercraft

Figure 15 through Figure 19 present the results of B-SST in the Hovercraft domain over each of workspace in Figure 4.

It happened again that all the algorithm perform very bad in forest workspace, let's just skip it.

The rest of hovercraft results are very similar to the dynamic car results. B-SST performs as well as A-BEAST do. However in Hovercraft domains, SST is able to find cheaper solution as we give it more time.

Quadrotor

Figure 20 present the results of B-SST in the quadrotor with forest environment.

A-BEAST performs best in this domain. B-SST is the second best algorithm according to GAT plot. As we can see from the plots, both Restarting RRT with Pruning and SST performs bad. Since B-SST directly switch to SST-with-cost-pruning, it is reasonable that it performs not good as well.

Discussion

As you can see from the result plots, there still some performance gap could be filled between B-SST and other motion planning algorithms in some problems, e.g., Figure 11, Figure 20. So we are now keep working on a new anytime motion planning algorithm, in which we try to combine effort and cost reasoning along each edge. The idea is that for each edge, we can maintain a beta distrubution that reflect the probabilty of both success in propagation and resulting in a path cheaper than incumbent.

First, to estimate the cost of a solution path that an edge will participate in, we introduce the following notations. For every abstract edge, we maintain a cost coefficient ϵ_e to record the ratio of the average cost of the motions across the edge over the abstract edge cost. Then for those edges containing motions, we can calculate the cost estimate $\hat{c}_e$ by averaging on its motions' cost. For those edges without a single successful propagation, we do this cost estimate by computing the product of edge cost and a global average coefficient $\bar{\epsilon}$ which is just an average on all ϵ_e.

$$\epsilon_e = \frac{\sum_{s \in e} c_s / e.totalmotions}{c_e}$$

$$\hat{c}_e = \begin{cases} \epsilon_e \cdot c_e & e \in \text{edges with successful propagation} \\ \bar{\epsilon} \cdot c_e & e \in \text{edges without successful propagation} \end{cases}$$

With a cost estimate for each edge, we can compute the total cost estimate $\hat{f}_e$ of a solution path that a particular edge will participate in by summing up the estimated cost-to-come $\hat{g}_e$ and the estimated heuristic cost-to-go $\hat{h}_e$and the edge cost estimate $\hat{c}_e$.

$$\hat{f}_e = \hat{g}_e + \hat{h}_e + \hat{c}_e$$

Recall we want to pop a best edge from open that some how make a good trade off between effort estimate and cost estimate. We can image that for each edge to the goal, there must exist a trade off curve like a Pareto frontier that describe the relation between these two quantities. Figure 21 shows an example curve. By looking up this curve, we can find the best effort estimate e_{best}which ensure its estimate cost less then incumbent. Then we re-sort open by e_{best} and pop the first edge to execute a new propagate trial. Afterword, we update all trade off curves of all edges based on latest beta distribution. We are now making this idea more practicable by improving the details and will try to implement it in the near future.

Conclusion

We present a new anytime motion planning approach called B-SST. B-SST first runs BEAST to find a first solution as quickly as possible, then switch to another motion tree growth process called SST-with-cost-pruning, which adopt both idea from SST and cost pruning algorithms. Results with a variety of vehicles and environments showed that B-SST is competitive compared to A-BEAST and other successful anytime planners. We also discussed a more sophisticated idea on how to create a better anytime motion planner.

Acknowledgments

The authors would thank the members of the UNH AI Group and CS830 for their insightful comments.

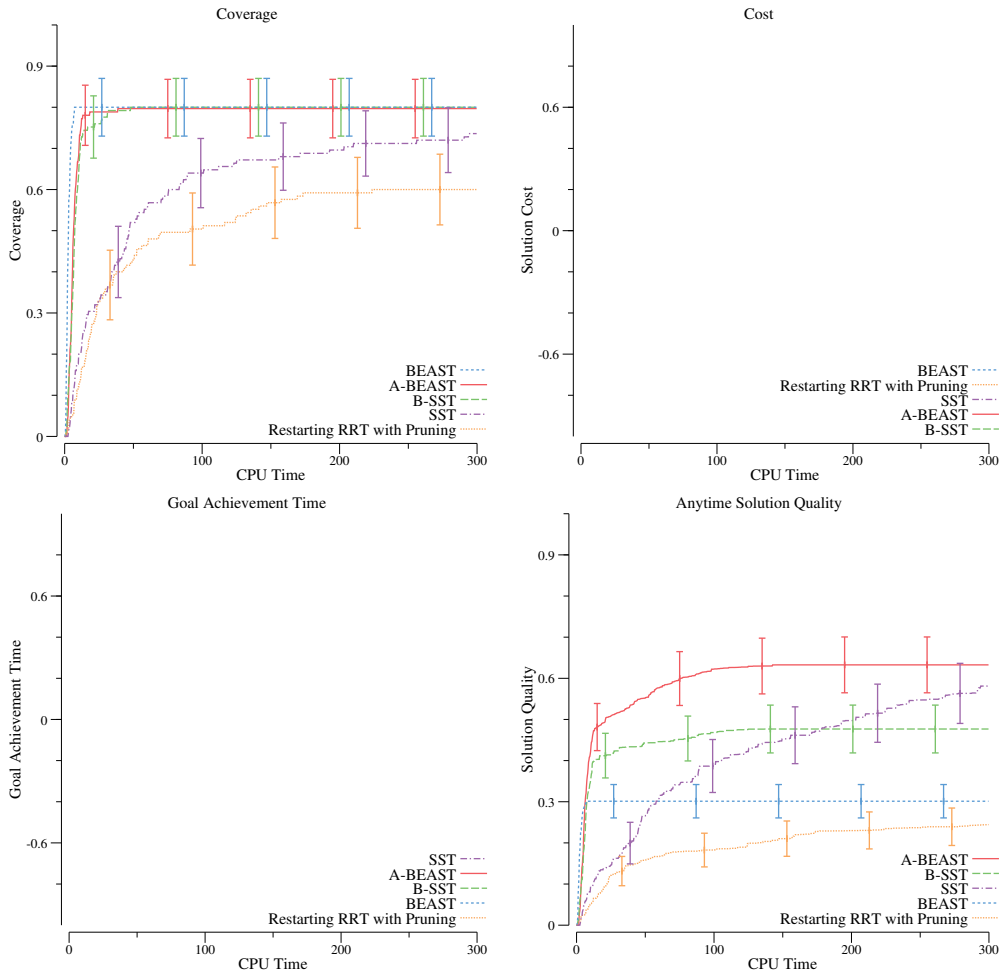


Figure 10: Dynamic car results in the forest workspace.

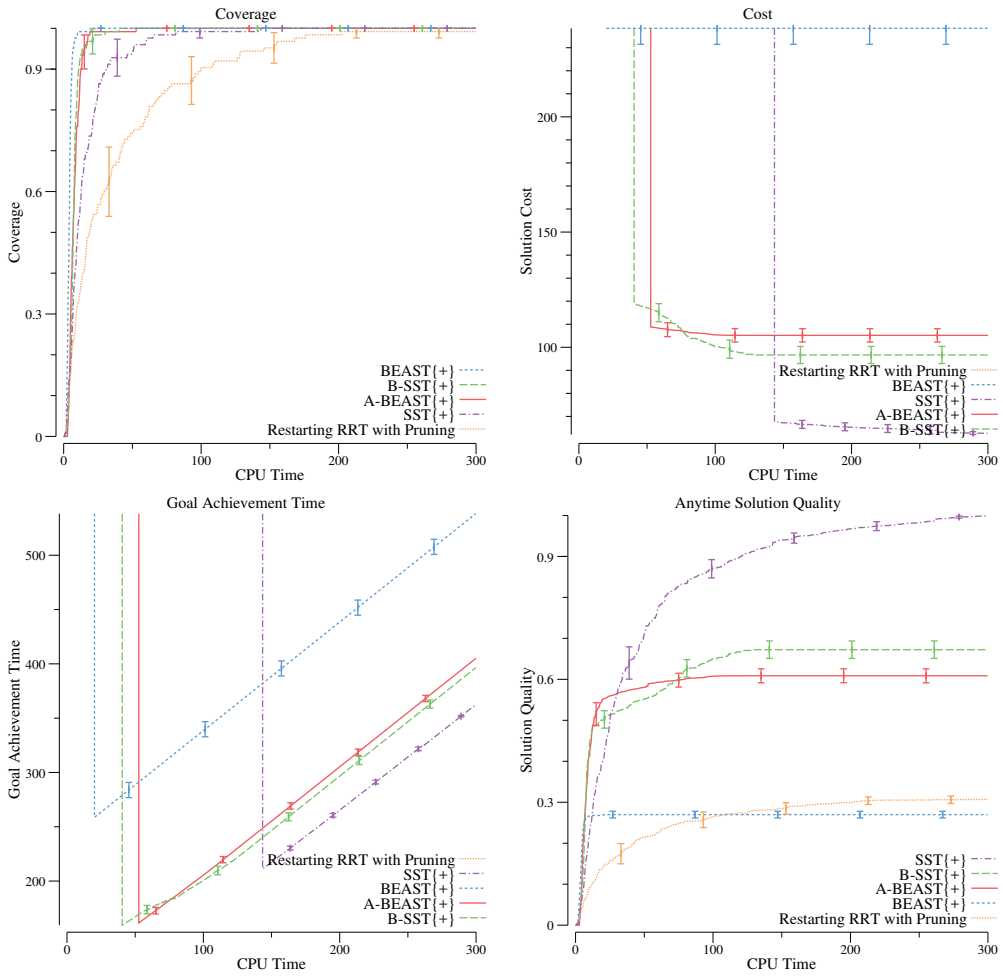


Figure 11: Dynamic car results in the single-wall workspace.

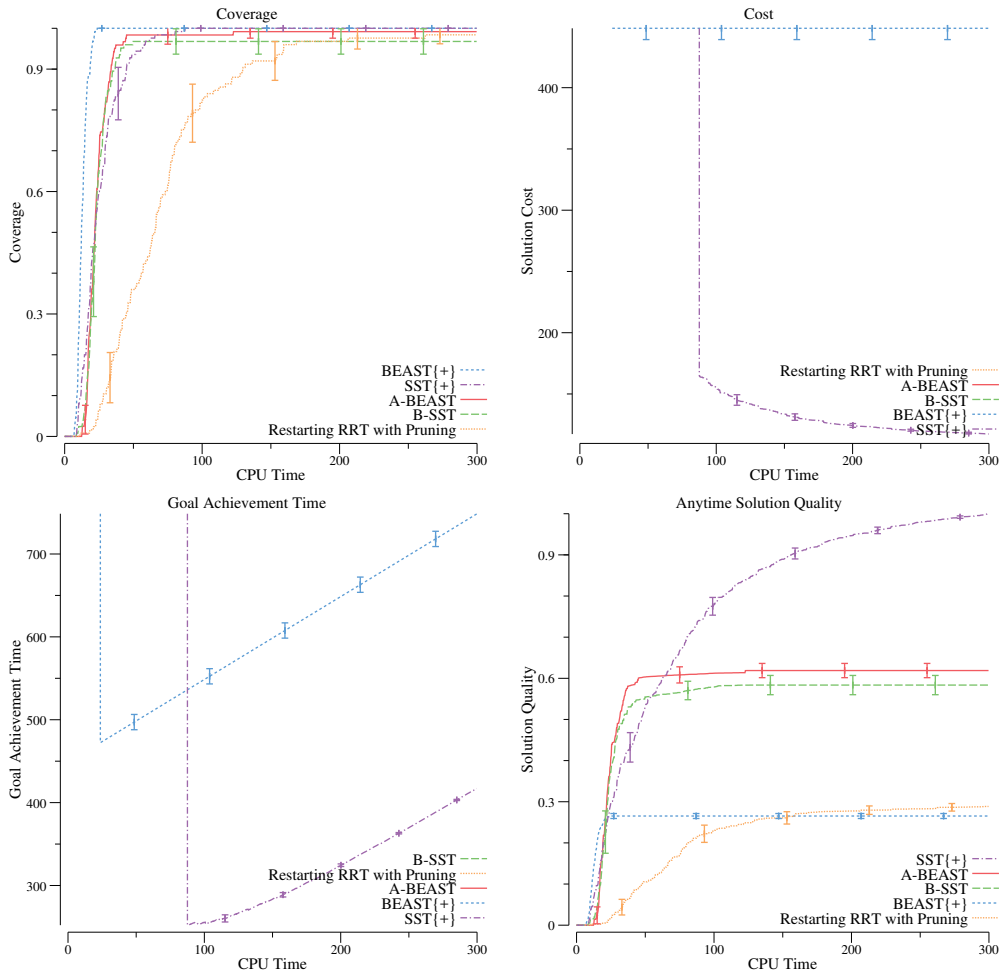


Figure 12: Dynamic car results in the 3-ladder workspace.

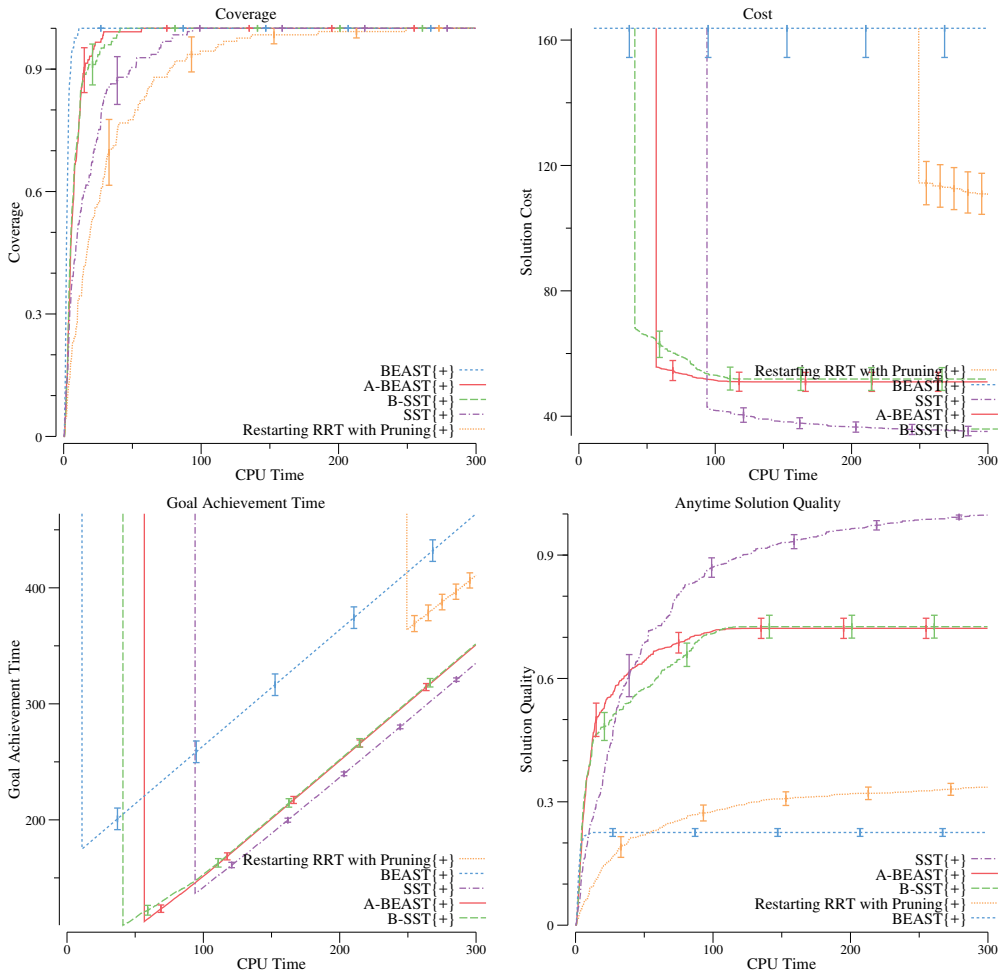


Figure 13: Dynamic car results in the parking-lot workspace.

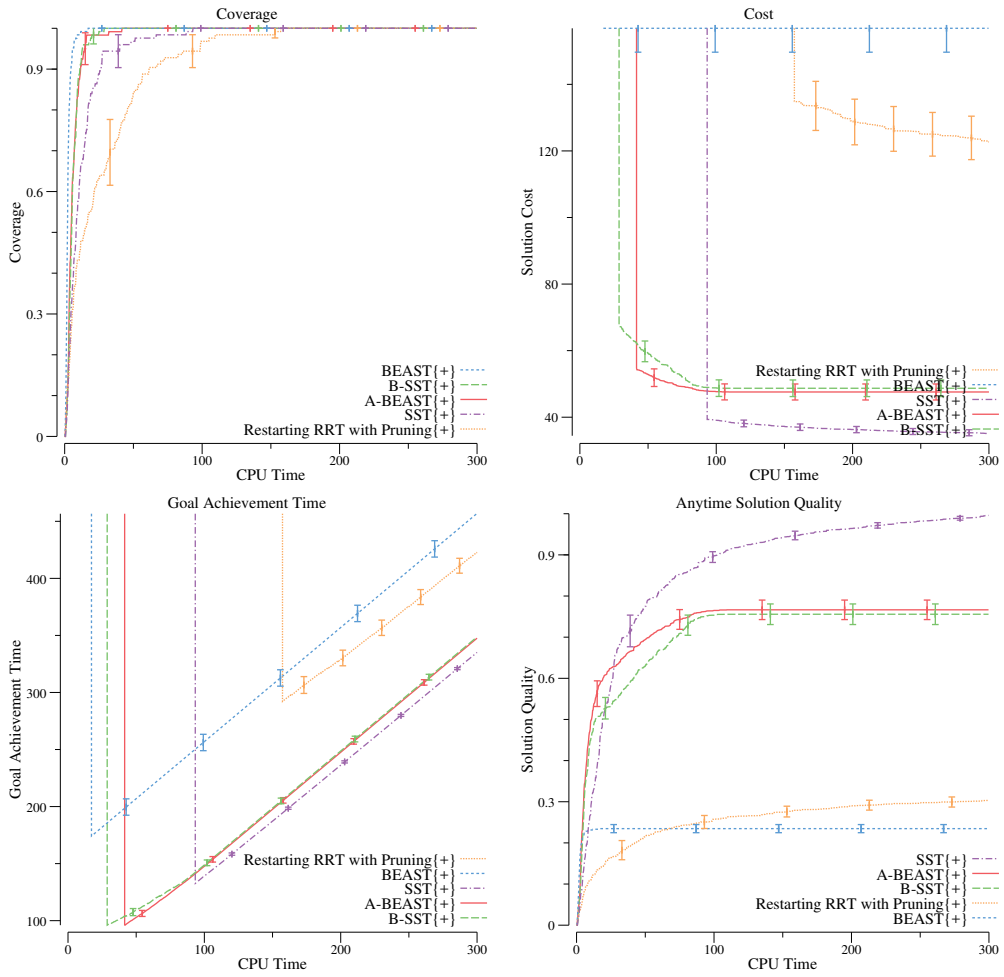


Figure 14: Dynamic car results in the intersection workspace.

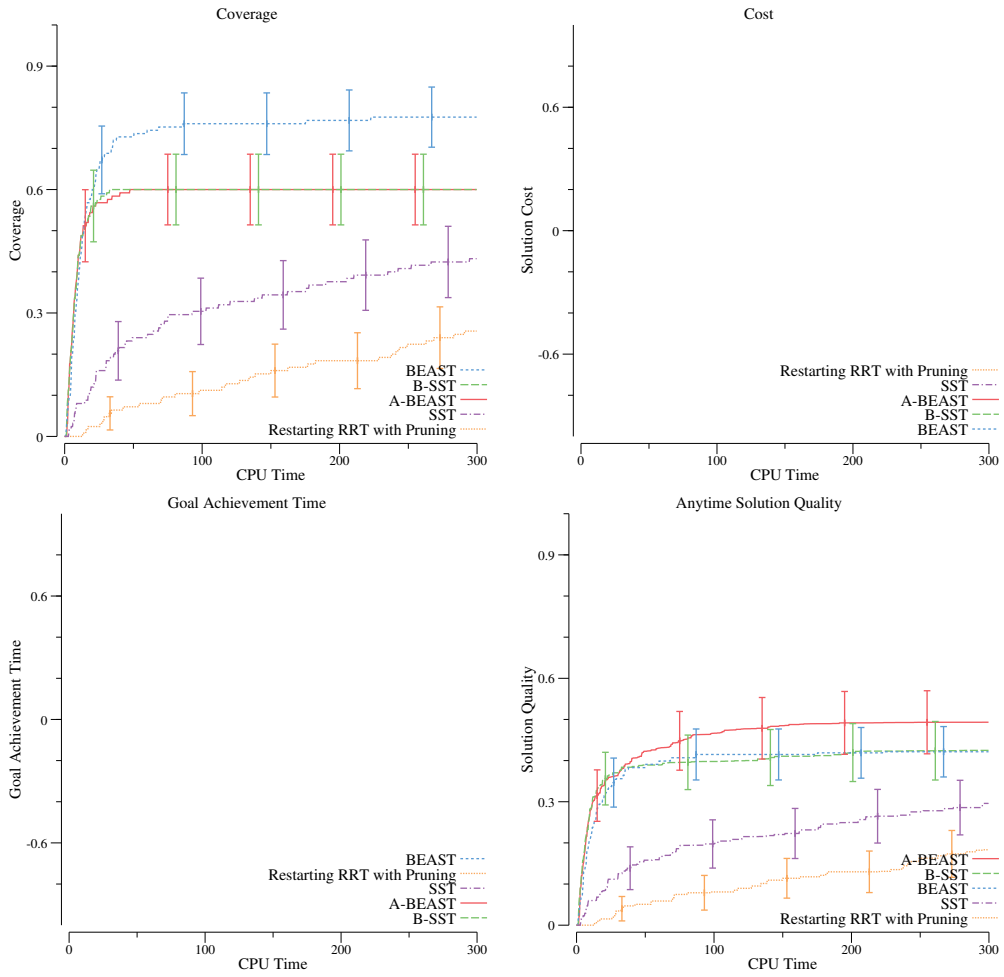


Figure 15: Hovercraft results in the forest workspace.

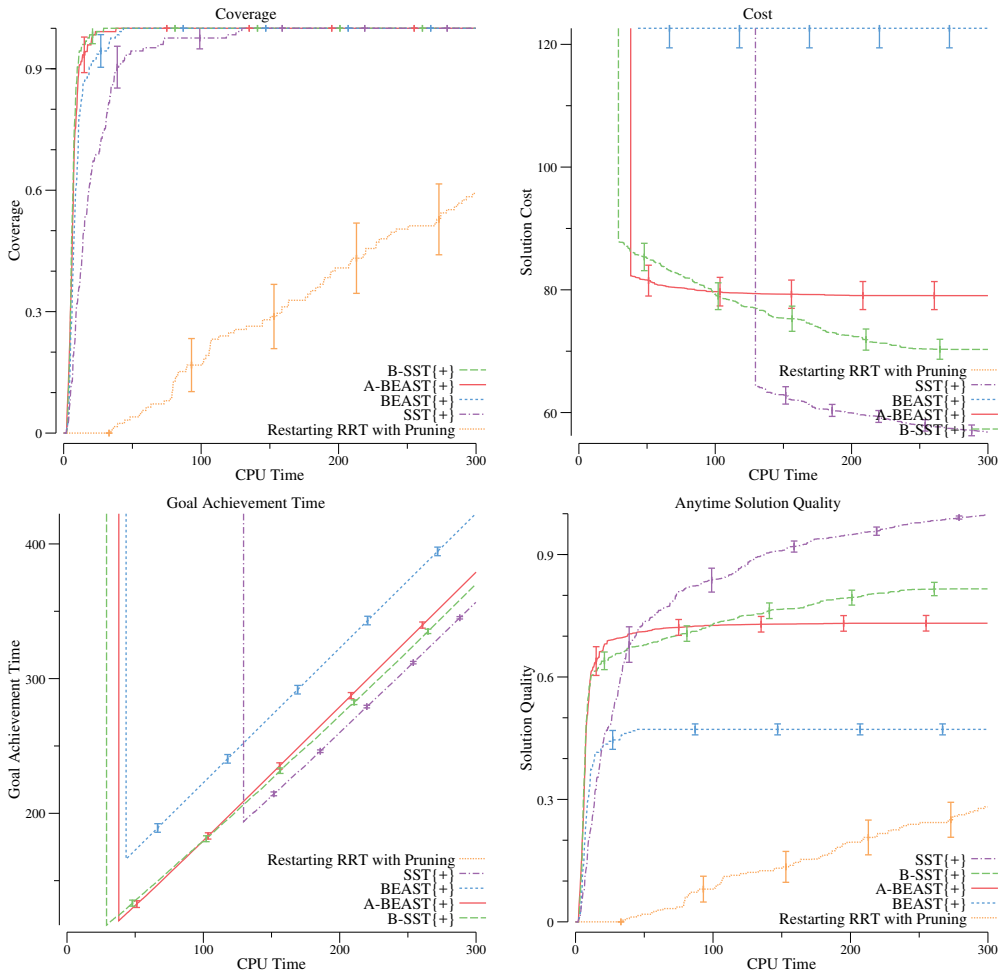


Figure 16: Hovercraft results in the single-wall workspace.

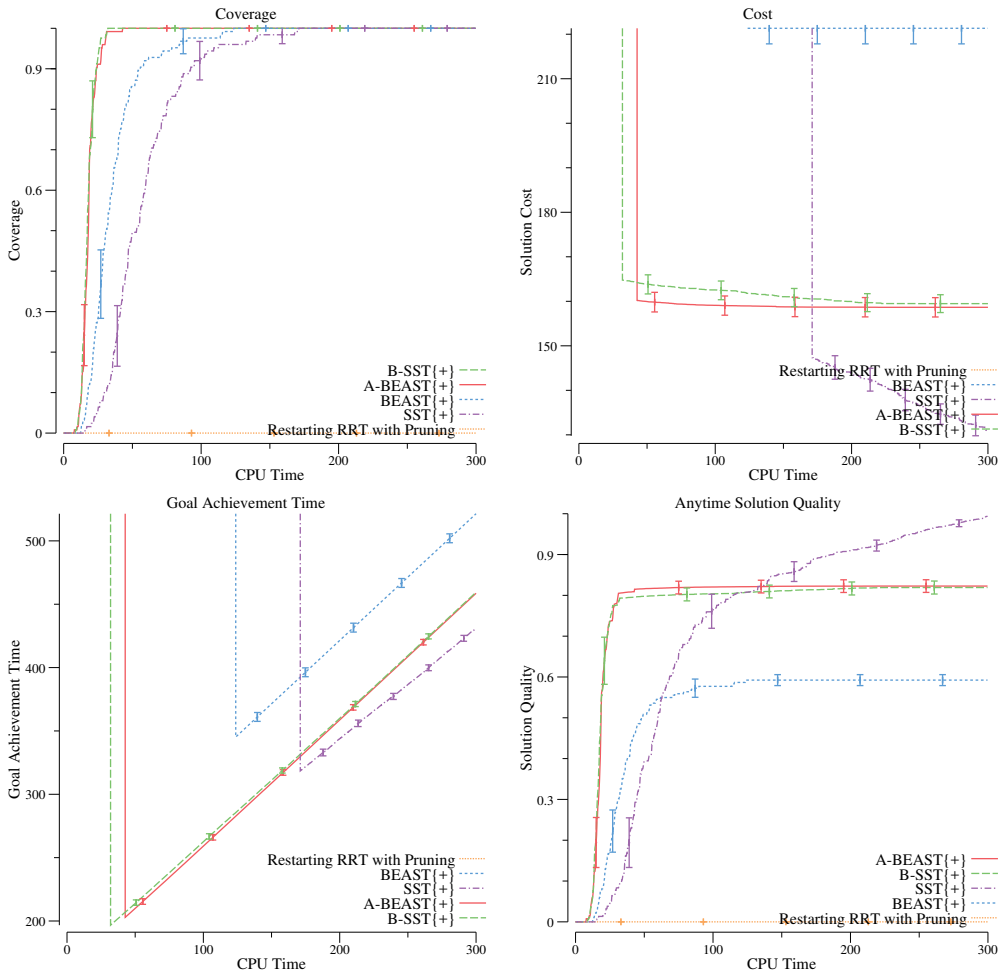


Figure 17: Hovercraft results in the 3-ladder workspace.

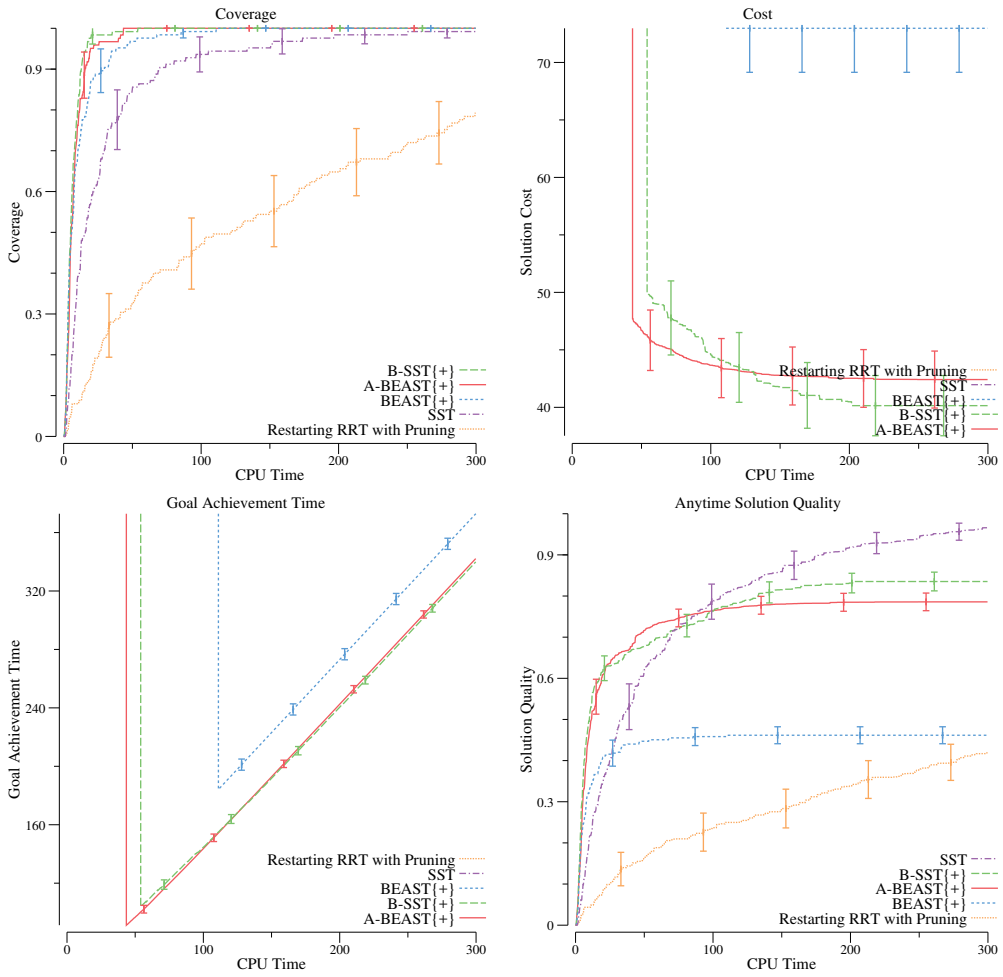


Figure 18: Hovercraft results in the parking-lot workspace.

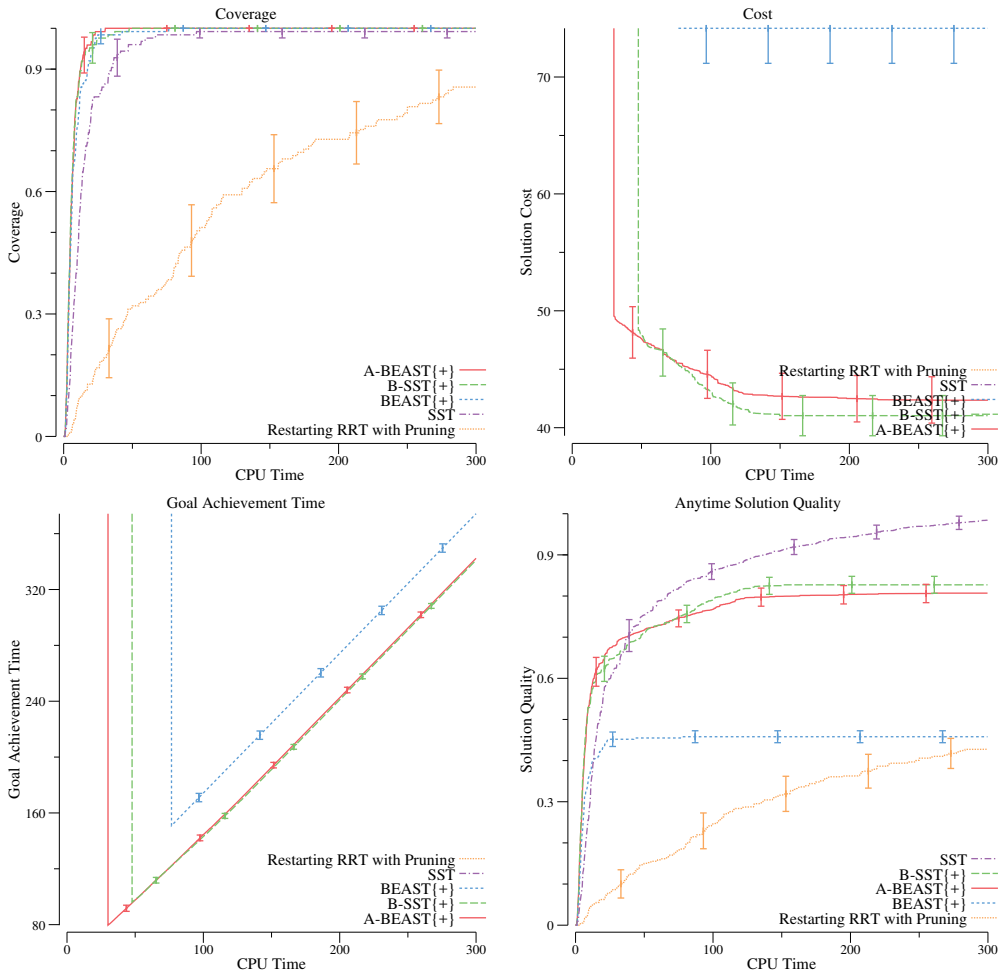


Figure 19: Hovercraft results in the intersection workspace.

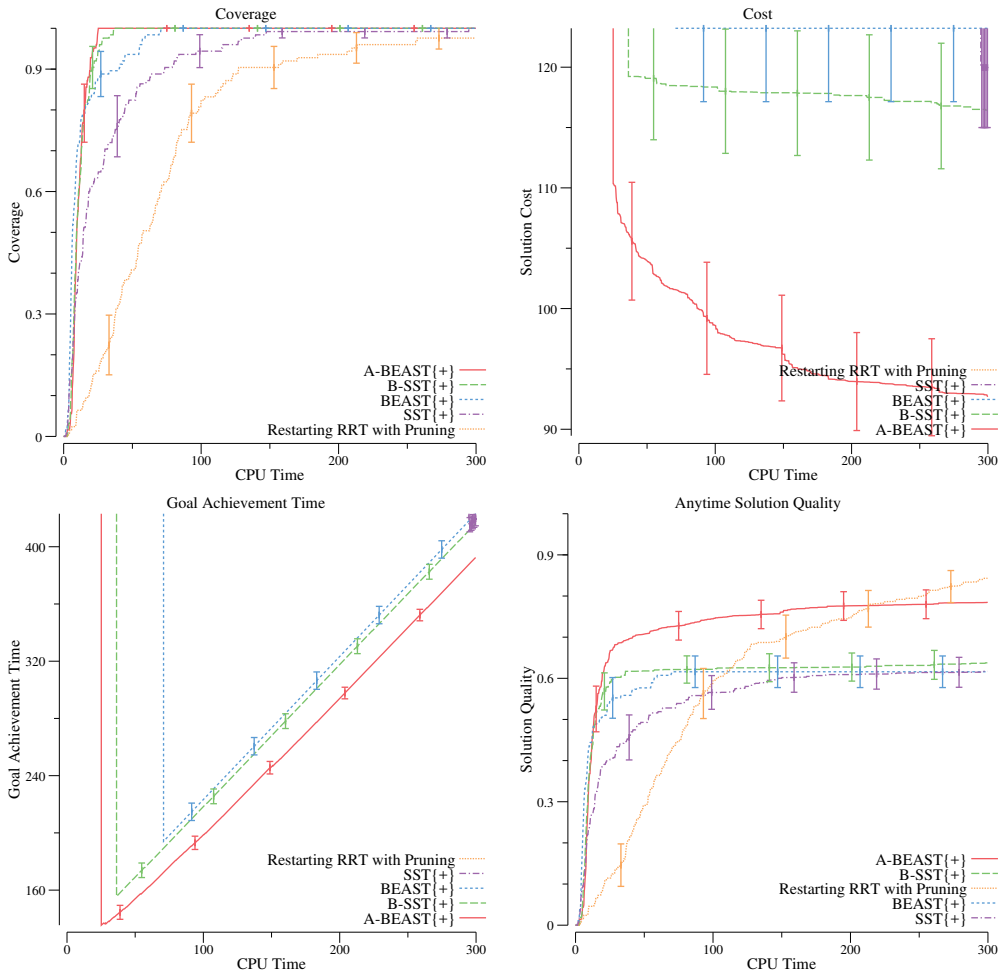


Figure 20: Quadrotor results in the forest workspace.

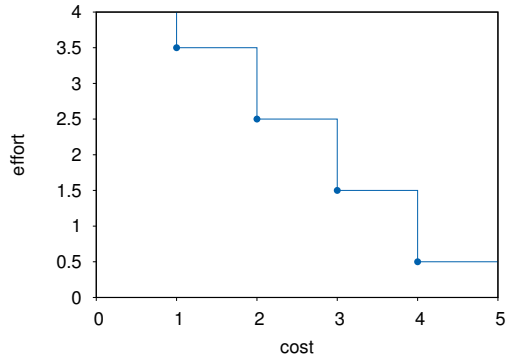


Figure 21: An example effort-cost trade off curve.

References

[Hauser and Zhou 2015] Hauser, K., and Zhou, Y. 2015. Asymptotically optimal planning by feasible kinodynamic planning in state-cost space. In *arXiv preprint arXiv:1505.04098*.

[Kiesel and Ruml 2016] Kiesel, S., and Ruml, W. 2016. A bayesian effort bias for sampling-based motion planning. In *ICAPS workshops on Planning and Robotics*, 158–165.

[Kiesel, Burns, and Ruml 2015] Kiesel, S.; Burns, E.; and Ruml, W. 2015. Achieving goals quickly using real-time search: Experimental results in video games. *Journal of Artificial Intelligence Research* 54:123–158.

[Kiesel 2016] Kiesel, S. 2016. *Robotics Needs Non-classical Planning*. Ph.D. Dissertation, University of New Hampshire.

[Koenig and Likhachev 2002] Koenig, S., and Likhachev, M. 2002. D* lite. In *AAAI/IAAI*, 476–483.

[LaValle and Kuffner 2001] LaValle, S. M., and Kuffner, J. J. 2001. Randomized kinodynamic planning. *The International Journal of Robotics Research* 20(5):378–400.

[Le and Plaku 2014] Le, D., and Plaku, E. 2014. Guiding sampling-based tree search for motion planning with dynamics via probabilistic roadmap abstractions. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 212–217. IEEE.

[Li, Littlefield, and Bekris 2015] Li, Y.; Littlefield, Z.; and Bekris, K. E. 2015. Sparse methods for efficient asymptotically optimal kinodynamic planning. In *Algorithmic Foundations of Robotics XI*. Springer. 263–282.

[Şucan and Kavraki 2009] Şucan, I. A., and Kavraki, L. E. 2009. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*. Springer. 449–464.

[Thayer, Benton, and Helmert 2012] Thayer, J. T.; Benton, J.; and Helmert, M. 2012. Better parameter-free anytime search by minimizing time between solutions. In *SOCS*, 120–128.