2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

Branch and Bound for Informative Path Planning

Jonathan Binney and Gaurav S. Sukhatme

Abstract— **We present an optimal algorithm for informative path planning (IPP), using a branch and bound method inspired by feature selection algorithms. The algorithm uses the monotonicity of the objective function to give an objective function-dependent speedup versus brute force search. We present results which suggest that when maximizing variance reduction in a Gaussian process model, the speedup is significant.**

I. INTRODUCTION

Path planning is one of the core problems in robotics, and approaches to path planning can be roughly decomposed based on the objective functions that they optimize. Perhaps the most common objective function is the minimization of path length, which leads to the shortest path between two points. Because the overall length of the path is equal to the sum of the lengths of smaller portions of the path, dynamic programming approaches can efficiently solve these kinds of problems.

In some cases, however, we may wish to optimize a more complicated function of the robot's path. In this paper, we consider the problem of planning a path for a robot equipped with sensors that take measurements of its environment. Autonomous underwater vehicles (Fig. 1) are an example of such a system. They provide scientists with the ability to collect data such as temperature, conductivity, salinity, as well as a variety of other quantities.

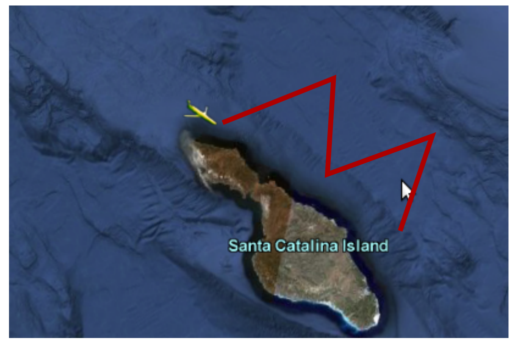


Fig. 1: Position and a planned path for one of our underwater gliders near Catalina island. We deploy the gliders in the Southern California Bight (SCB), to collect data including temperature, conductivity and salinity.

Depending on the application area, ground or air vehicles can also be used as platforms to move sensors through areas which a scientist wants to study. If the scientist manually specifies the exact trajectory for the robot to take while collecting sensor measurements, the problem is relatively straightforward. The problem that we examine in this work is that of having a robot autonomously decide what path to take while collecting measurements, based on a probabilistic model of the quantity being studied. We refer to this as informative path planning (IPP).

A. Related Work

The informative path planning problem combines concepts from sensor placement and path planning, and much of the prior work is from these two areas. We also borrow ideas from the feature subset selection literature, which has made extensive use of branch and bound methods.

1) Path Planning and Probabilistic Search: There has been extensive research into the problem of path planning for mobile robots. Many of these methods work on discrete graphs, and use dynamic programming, combined with heuristics to improve running time [1]. For path planning in a continuous space, quite a bit of work has been done on generating random graphs or trees and then solving a discrete problem [2][3]. The goal in these cases is usually either to find a feasible path or to optimize path length. Because the objective functions for IPP are more complicated, dynamic programming methods cannot provide an optimal algorithm.

More complicated objective functions are used in probabilistic robot search, where the goal is typically to find a target in the shortest amount of time, given a probabilistic model of the target location and motion. These problems have been solved in a number of ways, including finite horizon based planning [4]. There has also been research into using branch and bound for probabilistic robot search [5], which motivated us to look at the applicability of branch and bound methods to IPP.

2) Adaptive Sampling and Feature Selection: The problem of where to take measurements to best monitor a quantity of interest has been studied fairly extensively in the context of sensor placement. In the sensor placement problem, the goal is to place a number of immobile sensors to best monitor a sensed quantity of interest. Although there is a limit on the

Jonathan Binney (binney@usc.edu) and Gaurav S. Sukhatme (gaurav@usc.edu) are with the Robotic Embedded Systems Laboratory and Department of Computer Science at the University of Southern California (USC). This work was supported in part by the NOAA MERHAB program under grant NA05NOS4781228 and by NSF as part of the Center for Embedded Network Sensing (CENS) under grant CCR-0120778, by NSF grants CNS-0520305 and CNS-0540420, by the ONR MURI program (grants N00014-09-1-1031 and N00014-08-1-0693) by the ONR SoA program and a gift from the Okawa Foundation. Jonathan Binney was supported by a doctoral fellowship from USC.

978-1-4673-1405-3/12/$31.00 ©2012 IEEE

number of sensors, they can be placed in arbitrary locations relative to each other, and the greedy algorithm has been shown to perform well in cases where the objective function is submodular [6][7].

Feature selection is a similar problem to sensor placement, and involves choosing a limited number of features from some larger set in order to maximize an objective function of the chosen set. For cases where the objective function is monotonic, branch and bound methods have been used extensively [8][9]. The upper bound that we use for our algorithm is essentially the path planning version of the upper bounds used in these feature selection papers.

3) Applied Adaptive Sampling for Oceanography: The problem of where to take samples that best allow modeling of environmental quantities has also been studied for specific models, especially in oceanography. Genetic algorithms were used to determine sampling locations for an AUV in [10]. Adaptive modeling approaches for ocean observing are presented in [11] and [12]. Quite a bit of other research has been done on adaptive sampling for specific environmental applications and models as well. The difference between this work and those is that they typically look at specific models and focus on getting accurate prediction results for real applications. Our work makes a simplifying assumption by coarsely discretizing the problem, and does not go into the question of whether the model used is accurate for any one particular phenomenon. Instead we focus on the structure behind path planning for a general class of objective functions which are monotonic and in which visiting locations greatly decreases the usefulness of visiting nearby locations. We attempt to come up with an algorithm which can optimally solve small to medium sized problems of this type with a reasonable amount of computation. In the future, examining the performance of our algorithm on more realistic and complex models of specific environmental phenomena would be interesting.

4) Informative Path Planning: Recently, a number of papers have addressed the problem of planning informative paths for robots. The mixed integer approach in [13] allows for a large number of constraints to be handled, but requires linear objective functions. A more realistic objective function is used in [14], and the recursive-greedy algorithm for submodular orienteering from [15] is used to solve it. Our prior work in [16] is also based on this submodular orienteering approach, and considers extensions specific to AUVs.

A branch and bound extension to the recursive-greedy algorithm is also presented in [14], using the greedy sensor placement algorithm as an upper bound on informative path planning. Although they show that this can speed up the recursive-greedy algorithm, the solutions found by recursive-greedy are not necessarily optimal to begin with. Also, because greedy sensor placement is itself an approximation, the resulting value must be multiplied by $(1 - \frac{1}{e})^{-1}$ to get a true upper bound, weakening its ability to prune parts of the search tree.

The algorithm that we present in this paper also uses a branch and bound method, but starts with exhaustive search rather than recursive-greedy. We do this because even for submodular objective functions, the recursive-greedy algorithm provides an approximation guarantee which is worse than constant (the approximation factor grows logarithmically with the number of nodes in the path). Exhaustive search, on the other hand, gives an optimal solution.

The contribution of this paper is the extension of the branch and bound techniques from feature selection methods to the informative path planning problem. The resulting algorithm is typically much more faster than exhaustive search, but still guaranteed to find the optimal path. We give results that attempt to analyze how the structure of the problem affects the running time of our algorithm.

II. Problem

In adaptive sampling problems, the goal is to choose samples which best allow estimation of the target variables. Informative Path Planning (IPP) extends the adaptive sampling problem to the case of a robot which takes samples as it moves along a path. Instead of directly choosing sample locations, we must choose a path which has "good" sample locations along it.

A. Path Planning Notation

We formulate IPP as a discrete problem. First we form a graph G which represents the mobility of the robot. Each node $v \in V$ represents a waypoint which the robot can visit, and each edge $e \in E$ is a movement that the robot can make to travel between two nodes. A cost $C(e)$ associated with each edge represents how difficult it is for the robot to traverse the edge. Depending on how the graph is constructed, edge lengths may represent distance, traversal times, or the amount of energy needed for a robot to travel between two nodes.

A robot path consists of a sequence of edges to traverse $P = [e_0, e_1, ...e_k]$, and the length of a path is the sum of the costs of the edges,

$$\|P\| = \sum_{i \in P} C(e).$$

For the sake of clarity, we represent a path as a sequence of edges to traverse rather than a sequence of nodes to visit. There may be multiple edges that a robot can traverse between two waypoints, so representing the path as just a sequence of nodes would require an extra graph transformation.

To solve the general path planning problem, we need to find a path P^* on the graph such that

$$P^* = \arg\max_P f(P),$$

such that $\|P\| < B$, where B is the maximum path length allowed. Here $f(P)$ is an objective function which determines how "good" a path is. We define the objective function we use in section II-C.

B. Gaussian Processes

Informative Path Planning (IPP) is the set of problems where the objective function $f(P)$ captures the "informativeness" of the samples that will be taken by the vehicle as it traverses the path P. This can be done, *e.g.* by maximizing mutual information, or by maximizing the reduction in variance of the scalar field being estimated. In either case, a probabilistic model of the scalar field being sampled is needed.

For this paper, we approximate a model of the scalar field over the entire region by a grid of evenly spaced samples. The locations of the samples are denoted by $X = \{x_0, x_1, ..., x_n\}$, and the corresponding value of the scalar field at each of these locations is denoted by $Y = \{y_0, y_1, ..., y_n\}$. For convenience, we choose the sample locations X to be the node locations in the vehicle's waypoint graph.

Gaussian Processes (GPs) provide a convenient and widely used way of building this probabilistic model. In a GP, the output variables are modeled as a multivariate Gaussian distribution, with one dimension for each variable. The covariance matrix K is built using a kernel function $k(x_i, x_j)$. A commonly used kernel is the squared exponential (SE) kernel

$$k(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|}{2l^2}).$$

Roughly speaking, the hyper-parameter l describes how quickly the field de-correlates with distance. A large value for l means that even spatially distant locations are quite correlated, whereas a small value for l means that variables are almost completely independent. This simple kernel defines a stationary GP; the non-stationary case has some interesting implications for informative path planning but we leave that to future work.

The hyper-parameters are typically learned from previously collected data. For the purpose of this paper, we assume that the hyper-parameters have already been learned, and their values are fixed. Once the hyper-parameters have been fixed the kernel function depends only on the sample locations X, and not on the output variables Y. Because of this, it is possible to calculate the predicted variance of each of the variables given a set of sampled locations, before the samples are actually taken.

Adopting notation similar to [17], for two subsets $A \subset X$ and $B \subset X$ we will use $K(A, B)$ to denote the matrix where element i, j is equal to $k(a_i, b_j)$. The ordering of elements within each set is chosen arbitrarily, as it will not affect the value of the objective function. Using this notation, we can calculate the posterior covariance matrix of all variables X after taking a set of samples A as

$$\Sigma_{post} = K(X, X) - K(X, A)[K(A, A) + \sigma_n I]^{-1} K(A, X). \tag{1}$$

Where σ_n is the measurement noise. The measurement noise depends on the type of sensor used, but is often much smaller than the variance of the field itself, and so makes a very small contribution. The objective function we will discuss in the following section is a function of this posterior covariance matrix.

C. Objective Function

In order to define an objective function that describes how "informative" the measurements taken by a vehicle are, we look to the extensive literature on sensor placement. Algorithms for determining the best sensor placements for measuring a scalar field modeled by a GP have used entropy [18], mutual information [6], and variance reduction [19]. Mutual information is commonly used because it is submodular [6] [19], but we will instead use the average reduction in variance

$$f_{ARV}(A) = \frac{1}{N}[sum(diag(K(X, X))) - sum(diag(\Sigma_{post}))],$$

because it is monotonic [19]. Here A is the set of sensor locations, X is the set of all possible sensor locations, $K(X, X)$ is the prior covariance of all of the variables, and Σ_{post} is the covariance after conditioning on the samples using Eq. 1. Although in some cases $f_{ARV}(A)$ may also be submodular [20], the algorithm we present in this paper does not require this property.

D. Informative Path Planning

Now that we have introduced notation for path planning and Gaussian processes, we are ready to define the informative path planning (IPP) problem. In words, we want to find to find the path from a node v_{start} to a node v_{end} whose samples result in the greatest average reduction in variance in the GP model. More formally, we want to find the optimal path P^*, such that

$$P^* = \arg\max_P f_{ARV}(P),$$

subject to a maximum budget $\|P^*\| \leq B$. We abuse notation slightly and use P both as the path and as the set of measurement locations along the path. For the purpose of this paper, we assume that the robot will collect samples at the nodes of the graph, and so we use "node" to refer both to the vertex in the graph, and the location of the sample. Modifying the definition of IPP to allow sample locations along edges instead of at nodes is relatively simple, but since this paper focuses on a branch and bound algorithm for IPP, we leave those details out.

III. Algorithm

In this section we first discuss the application of standard finite horizon planning to IPP. Then we describe how branch and bound methods have previously been used for feature selection. In feature selection, a monotonic objective function such as variance reduction can be used, but instead of planning a path along which measurements are taken, any subset of a given number of features may be chosen. Finally, we present a novel way to combine the branch and bound concept from feature selection with finite horizon planning to optimally solve IPP.

A. Finite Horizon Planning for IPP

Brute force search can be used to optimally solve the informative path planning problem. This method evaluates the objective function on all possible paths, and chooses the path with the highest value. Although this approach is optimal, it is not efficient - for all but the simplest graphs, the complexity of brute force search is exponential in the length of the path. Because of this, it is common to use brute force search to only look ahead a few nodes while planning the path, choose the best path up to that horizon, and then replan. If a horizon of one is used, this reduces to a purely greedy algorithm. If the horizon is long enough to include all paths within the planning budget, then this method is equivalent to brute force search, and will find the optimal solution.

To illustrate the importance of planning with a long horizon when solving IPP, we compare three examples in Fig. 2. For all three cases, a finite-horizon planner has been used to plan a path starting at the bottom left node, ending at the top right node, subject to a maximum length of 14. For the objective, we have used $f_{AVR}(P)$, the average variance reduction of the GP prediction at all nodes.

In Fig. 2a, a horizon of one has been used. This results in a completely greedy strategy, maximizing the gain in objective at each step. The planner chooses a path that moves to the center of the graph initially, which appears to be a good strategy. Once there, however, it does not have an effective way to spend the rest of its budget; it ends up moving to the top of the graph and is unable to collect any samples in the bottom right.

For comparison, the path in Fig. 2c has been planned with a horizon of ten steps. Because it looks far ahead, it is able to plan a path with nicely spaced traversals of the graph. The resulting path gets much better coverage than when planning with a horizon of one.

We have specifically chosen this example to be as simple as possible, using a square graph with unit edge costs. Even in this simple instance of IPP, a purely greedy algorithm is far from optimal. It is possible to construct graphs in which greedy algorithms will do far worse - for instance a directed graph in which greedy makes a poor choice early on and can never recover because the cost to retrace its steps is very high. This would be the case, for instance, when using an Autonomous Surface Vehicle (ASV) to take measurements along the surface of a river. A greedy planner might choose to head downstream to take useful samples early on, and not have enough budget left to acquire samples upstream later.

The results in Fig. 2 qualitatively show the benefit of using longer horizons when solving the IPP problem; quantitative results are presented in section IV-B. In practice, the length of horizon used will likely be chosen based on the amount of time available for computation. Our branch and bound method reduces the amount of computation needed, making it possible to use a longer horizon without requiring more planning time.

B. Branch and Bound

In section III-A, we explained the importance of using a long horizon for IPP. Unfortunately, because the running time of a finite horizon planner is exponential in the size of the horizon, we are limited by the amount of computation available. Parallel processing provides only limited help - because of the exponential complexity of the planning problem, multiplying the number of processors by m will only increase the horizon by $\log m$.

Instead, we look to branch and bound methods as a way of saving computation. When doing brute force search, branch and bound methods keep track of the best solution found so far, and its associated objective value. When exploring new parts of the search tree, it computes an upper bound on the highest objective value that can possibly be achieved. If this upper bound is worse than the current best solution, then this portion of the search tree cannot contain the optimal solution, and is pruned [21].

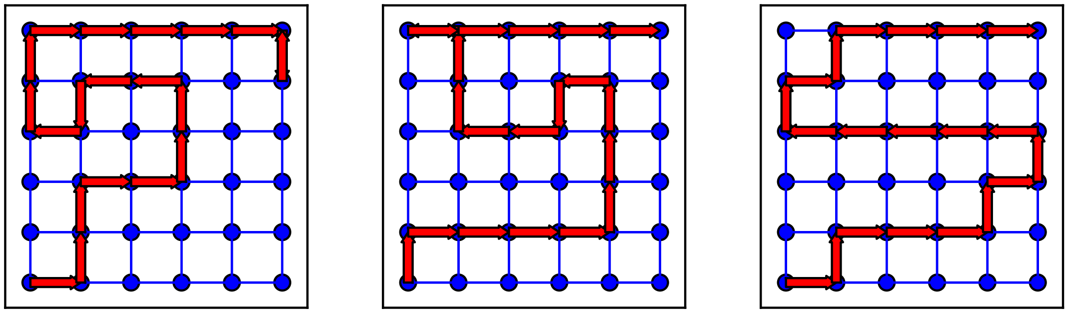
The key to branch and bound methods is finding an easily computable upper bound for the objective function. If the upper bound is loose (is often much higher than the actual objective), then few branches of the search tree will be pruned. The goal is to find a function which is as tight as possible, while still being a valid upper bound.

The upper bound that we use for IPP is inspired by the one used for feature selection in [8]. Feature selection is the problem of choosing a subset A of k features out of a set X of features. Branch and bound algorithms for feature selection rely on the fact that the objective function is monotonic in the set of features chosen. Starting with the set of all features X, features are removed one by one until only k features remain. Features are removed in all possible combinations, resulting in a search tree with the set X at the top, and sets of size k at the leaves.

Constructing the search tree in this way, combined with the monotonicity of the objective function, means that while exploring the search tree the objective function always decreases. This means that if at any point the objective value is already lower than a known value for a valid set of k features, the rest of the current subtree can be pruned.

C. The IPP-BNB Algorithm

The pseudocode for our planner is given in algorithm 1. The algorithm works recursively to find a path from node v_s to node v_e, subject to a maximum budget B. The argument P is the path which has been planned so far. X_{pilot} is the set of measurements which have been taken before running the planner (for example by any static sensors already in place). We use $L(v_i, v_j)$ to denote the length of the shortest path between two nodes. P^* and m^* are the best path seen so far and its objective value, and are global variables. After the algorithm runs, P^* will be the optimal path.

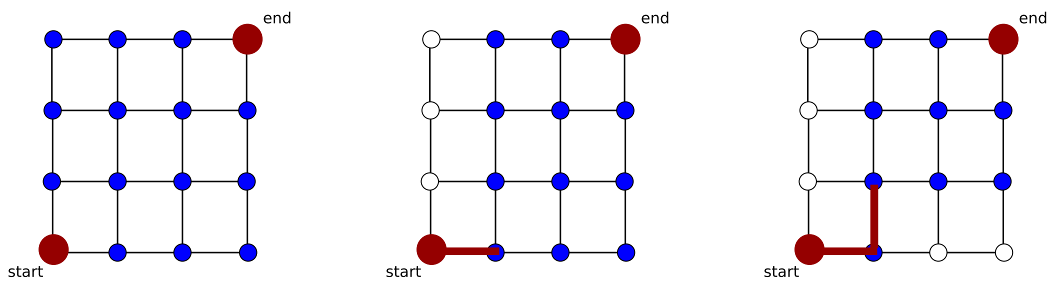


(a) Horizon of one step (greedy).

(b) Horizon of five steps.

(c) Horizon of ten steps.

Fig. 2: Paths planned by a finite horizon planner for IPP, using three different horizons. Because our objective function is expected variance reduction in a stationary, isotropic GP, the planner is essentially trying to “spread out” the path as much as possible.



(a) Initialization: start and end specified, planning commences.

(b) After recursing once.

(c) After recursing twice.

Fig. 3: An illustration of our branch and bound algorithm with a budget of 6. (a) Only the start and end nodes have been fixed. With a budget of 6, there is still some feasible path which would visit any given node in the graph. (b) The algorithm has fixed one edge in the path. There is a remaining budget of 5, and no feasible path reaches the nodes marked in white, so they have been removed from the reachable set. (c) Another edge has been fixed, and more nodes have been removed from the reachable set.

In algorithm 1, on line 9, a recursive call is made for each outgoing edge from the current node, to plan the remaining path. This is just a recursive brute force search, looking for the best possible path. On line 8, the algorithm checks the upper bound. If the upper bound for the sub-path that has been planned so far is lower than a previously found solution (found on some other branch of recursion), then further recursion is not necessary - this branch of the search tree can be pruned.

The pseudocode for the upper bound is given in algorithm 2. The set R is the reachable set - all sample locations which could be reached by a feasible path, given the current start and end nodes and the remaining budget. We treat the sample locations as being on an edge, and check the reachability of each edge on line 3. The reason for having samples along edges instead of just at nodes is to allow for samples taken as the robot moves between waypoints. For the results in this paper, however, we use 2 samples for each edge - one at each end, so all sample locations are actually nodes in the graph.

The execution of the algorithm is shown visually in Fig. 3. For the purpose of this explanation, we use a graph with unit length edges and a budget of $B = 6$. At the top of our search tree, we fix only the start and end nodes of the path, as shown in Fig. 3a. Initially, there is some feasible path which would reach any given node in the graph without exceeding the budget, and so all nodes are in the reachable set.

For each possible edge leaving the start node, we recursively call our algorithm. The state during the first such recursive call is shown in Fig. 3b. There is a budget of 5

Algorithm 1 IPP-BNB$(v_s, v_e, B, P, X_{pilot})$

is_leaf = true
for each edge leaving v_s **do**
  $P_{new} = P \cdot$ [edge]
  $v_{new} =$ end node of edge
  $B_{new} = B -$ cost of edge
  if $L(v_{new}, v_e) \leq B_{new}$ **then**
    is_leaf = false
    if UBOUND$(v_{new}, v_e, B_{new}, P_{new}, X_{pilot}) > m^*$ **then**
      IPP-BNB$(v_{new}, v_e, B_{new}, P_{new}, X_{pilot})$
    end if
  end if
end for
if is_leaf **then**
  $m \Leftarrow f_{ARV}$(samples on $P \cup X_{pilot}$)
  if $m > m^*$ **then**
    $P^* \Leftarrow P$
    $m^* \Leftarrow m$
  end if
end if
return

Algorithm 2 UBOUND(v_s, v_e, B, X_{pilot})

$R = \{\}$
for each edge in the graph **do**
  if $L(v_s,$ start node of edge$)$ + cost of edge + $L($end node of edge$, v_e) \leq B$ **then**
    $R \Leftarrow R \cup \{$samples on this edge$\}$
  end if
end for
return $f_{ARV}(R \cup X_{pilot})$

remaining, and so there is no feasible path that will reach the nodes marked in white and still be able to reach the end node within the budget. We remove those nodes from the reachable set and recalculate the upper bound. If the upper bound is lower than the objective achieved by another path we have already explored, then we can stop exploring this part of the search tree.

If we cannot yet prune this portion of the tree, a recursive call is made for each possible next edge. The state during this next recursive call is shown in Fig. 3c. Two more nodes have been removed from the reachable set, and the upper bound is re-evaluated.. We continue recursively in this way until either the upper bound is lower than the objective achieved by some path that has already been examined, or the budget is exceeded.

We have also implemented the finite horizon version of IPP-BNB, but do not give pseudocode because it is only a slight change. Instead of recursing until there is no more budget, the algorithm takes the best path up to the horizon, fixes the first edge in the path, and then plans again out to

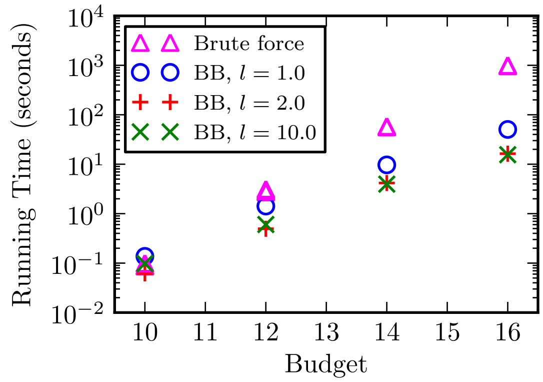


Fig. 4: Running times for various values of the length scale in the GP kernel. The y-axis is shown in log scale. The brute force running times are identical for each of the three length scales used.

the horizon. In the full horizon case, since we only prune out solutions for which the upper bound is worse than a solution already considered, IPP-BNB will always find the optimal solution.

IV. Results

Because the branch and bound algorithm finds the optimal solution, there is little use in showing the objective values achieved for various IPP scenarios. Instead, we examine the running time needed to find the optimal solution, and compare this to the brute force approach.

For these results we use a 5x5 square graph with all edge lengths set to one. We use a squared exponential kernel. In a field trial, the hyper-parameters of the GP (just the characteristic length in this case) would be learned from an initial, manually planned run of the vehicle. To account for this, we look at the effect of varying the characteristic length on the runtime of our algorithm.

A. Varying the Characteristic Length

Here we vary the characteristic length l of the kernel, to see how this affects the running time of our algorithm. The characterstic length determines how quickly measurements become de-correlated over distance. With $l = 1$, only nodes that are very close to each other have much correlation in their values. With $l = 10$, even nodes on opposite sides of the graph are almost perfectly correlated.

For each of three different characteristic lengths, Fig. 4 shows the running times of brute force and our algorithm. As expected, the running time of brute force search does not change; regardless of the values of the objective function, all possible paths are tried. Interestingly, our branch and bound algorithm provides a greater speedup for $l = 2$ and $l = 10$ than it does for $l = 1$. The intuition for this that when the nodes are almost uncorrelated, very few paths are poor enough to prune. Even if a path takes the robot into an area

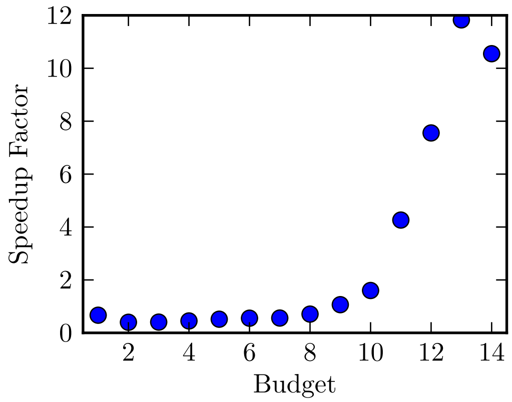


Fig. 5: Speedup factor (brute force running time divided by branch and bound running time) for a finite horizon planner with various horizon lengths. Although we vary the horizon, we fix the overall budget at 14.

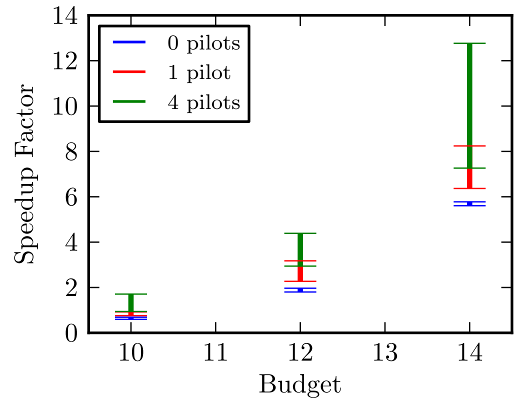


Fig. 7: Running time speedup (brute force running time divided by branch and bound running time) for varying number of pilot measurements. For each number of pilot measurements, 5 trials were done, and the standard deviations of the speedup is shown by the error bars.

where many other samples have already been taken, a high objective value can be achieved as long as the robot does not sample the exact same location multiple times. Still, even in the case of $l = 1$, our algorithm achieves a significant speedup versus brute force search.

B. Finite Horizon Planning

In section IV-A, we planned with no horizon, and so the algorithm found the optimal solution. Here, we examine how our planner affects running time for the finite-horizon case.

In Fig. 5, we have fixed the budget to 14, and the characteristic length to $l = 2$. For very small horizons, we can see that adding branch and bound to the planner actually makes it slower (speedup < 1). This is because when planning for a small horizon, the branches of the search tree that get pruned are relatively small, and pruning them does not make up for the overhead of calculating the upper bound. As the horizon gets larger, branch and bound provides a significant speedup. In the future, it would be interesting to try to predict the speedup gained by using branch and bound for various kernel functions, hyper-parameter values, horizon lengths, and graph sizes, and use this to decide when to calculate the bound.

C. Adding Pilot Measurements

Next we test the effect of adding pilot measurements. These measurements could be locations that have been manually sampled before the experiment or static sensors that are already in place. In either case, they are measurements which are available to the planner before planning a path. The goal of the planner is then to choose a path that provides measurements which best augment the pilot measurements. In this section we have again fixed the characteristic length of the kernel to $l = 2$.

The result of planning with various numbers of pilot measurements is shown in Fig. 6a-6c. In each case we randomly place the pilot measurements at nodes of the graph, and then plan an augmenting path. As expected, the robot chooses paths which avoid the locations already sampled, in order to collect new data.

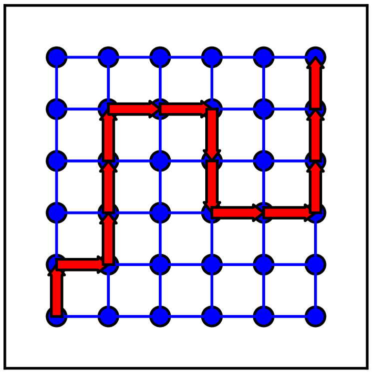
To quantify the effect that adding pilot measurements has on the speedup gained by our branch and bound method, we ran 5 trials for each number of pilot measurements. The average and standard deviation of the speedups gained are plotted in Fig. 7.

As we increase the number of pilot measurements, we see a greater speedup from using the branch and bound method versus brute force search. The intuition behind this is that when there are pilot measurements, the upper bound in our planner can prune parts of the search tree that would take measurements near those pilot samples. Put another way, conditioning on the pilot samples adds interesting structure to the GP covariance matrix, which our planner can use to discard paths that have no hope of taking useful measurements.

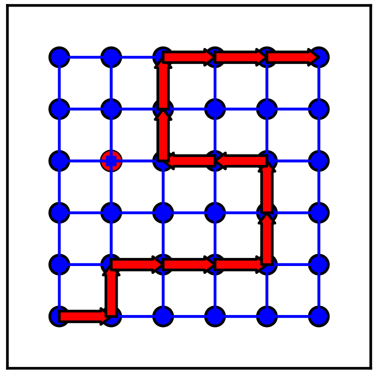
V. Conclusions

We have presented a new branch and bound algorithm for the informative path planning problem, and analyzed its performance in terms of running time in various scenarios. Our algorithm provides a significant speedup over brute force search while still finding the optimal path. Although the running time still appears to be exponential in the length of the path, because of the speedup that branch and bound provides finite horizon planners with longer horizons could be used.

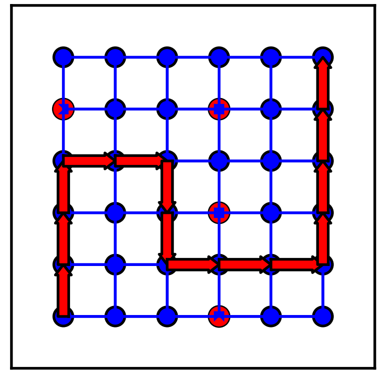
In future work, we plan to investigate the performance of this algorithm on non-stationary and anisotropic GP kernels.



(a) No pilot measurements.



(b) One pilot measurement.



(c) Four pilot measurements.

Fig. 6: One of the trials for each number of pilot measurements. Pilot measurement locations are marked by red circles. The planner is given a start location at the bottom left node, an end location at the top right node, and a maximum budget of 14 (each edge has unit length). Pilot measurements are places where samples have already been taken, for instance by static sensors. The planner must choose a path that best augments the set of pilot measurements.

We also plan to examine ways to extend our algorithm to work in an anytime fashion. In its current form, although it maintains a "best path found so far" while planning, it could take an arbitrarily long time to find paths that are close to optimal. This is because of the deterministic order in which the planner explores the space of possible paths.

There are also many more advanced branch and bound methods in the feature selection literature which may provide additional speedups. In particular, using a heuristic to explore the most promising areas of the search tree first may increase the lower bound more quickly, reducing the amount of the search tree that ends up getting searched.

REFERENCES

[1] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.

[2] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, p. 378, 2001.

[3] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in High-Dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[4] G. Hollinger and S. Singh, "Proofs and experiments in scalable, near-optimal search by multiple robots," *Proceedings of Robotics: Science and Systems IV, Zurich, Switzerland*, 2008.

[5] H. Lau, S. Huang, and G. Dissanayake, "Probabilistic search for a moving target in an indoor environment," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 3393–3398, 2006.

[6] C. Guestrin, A. Krause, and A. P. Singh, "Near-optimal sensor placements in gaussian processes," in *Proceedings of the 22nd international conference on Machine learning*, (Bonn, Germany), pp. 265–272, ACM, 2005.

[7] A. Krause, A. Singh, and C. Guestrin, "Near-Optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies," *J. Mach. Learn. Res.*, vol. 9, pp. 235–284, 2008.

[8] P. M. Narendra and K. Fukunaga, "A branch and bound algorithm for feature subset selection," *IEEE Transactions on Computers*, vol. C-26, pp. 917–922, Sept. 1977.

[9] P. Somol, P. Pudil, and J. Kittler, "Fast branch & bound algorithms for optimal feature selection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 7, pp. 900–912, 2004.

[10] K. D. Heaney, G. Gawarkiewicz, T. F. Duda, and P. F. J. Lermusiaux, "Nonlinear optimization of autonomous undersea vehicle sampling strategies for oceanographic data-assimilation," *Journal of Field Robotics*, vol. 24, pp. 437–448, June 2007.

[11] P. F. Lermusiaux, "Adaptive modeling, adaptive data assimilation and adaptive sampling," *Physica D: Nonlinear Phenomena*, vol. 230, pp. 172–196, June 2007.

[12] D. Wang, P. F. Lermusiaux, P. J. Haley, D. Eickstedt, W. G. Leslie, and H. Schmidt, "Acoustically focused adaptive sampling and on-board routing for marine rapid environmental assessment," *Journal of Marine Systems*, vol. 78, Supplement, pp. S393–S407, Nov. 2009.

[13] N. Yilmaz, C. Evangelinos, P. Lermusiaux, and N. Patrikalakis, "Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming," *IEEE Journal of Oceanic Engineering*, vol. 33, no. 4, pp. 522–537, 2008.

[14] A. Singh, A. Krause, C. Guestrin, and W. Kaiser, "Efficient informative sensing using multiple robots," *Journal of Artificial Intelligence Research*, vol. 34, pp. 707–755, 2009.

[15] C. Chekuri and M. Pal, "A recursive greedy algorithm for walks in directed graphs," in *46th Annual IEEE Symposium on Foundations of Computer Science, 2005.*, pp. 245–253, 2005.

[16] J. Binney, A. Krause, and G. S. Sukhatme, "Informative path planning for an autonomous underwater vehicle," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2010.

[17] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. the MIT Press, 2006.

[18] C. Ko, J. Lee, and M. Queyranne, "An exact algorithm for maximum entropy sampling," *Operations Research*, vol. 43, pp. 684–691, Aug. 1995.

[19] A. Krause, H. B. McMahan, C. Guestrin, and A. Gupta, "Robust submodular observation selection," *Journal of Machine Learning Research*, vol. 9, pp. 2761—2801, Dec. 2008.

[20] A. Das and D. Kempe, "Algorithms for subset selection in linear regression," in *Proceedings of the 40th annual ACM symposium on Theory of computing*, (Victoria, British Columbia, Canada), pp. 45–54, ACM, 2008.

[21] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.