Effort Informed Roadmaps (EIRM*): Efficient Asymptotically Optimal Multiquery Planning by Actively Reusing Validation Effort

Valentin N. Hartmann[1], Marlin P. Strub[2], Marc Toussaint[1], and Jonathan D. Gammell[3]

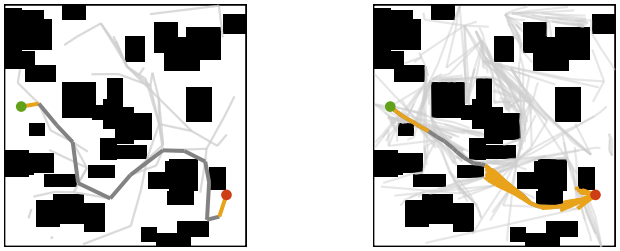
[1] Learning and Intelligent Systems Group, TU Berlin, Germany
[2] Jet Propulsion Laboratory, California Institute of Technology, USA. Work performed while at the University of Oxford.
[3] Estimation, Search, and Planning (ESP) Group of the Oxford Robotics Institute (ORI), University of Oxford, United Kingdom

Abstract. Multiquery planning algorithms find paths between various different starts and goals in a single search space. They are designed to do so efficiently by reusing information across planning queries. This information may be computed before or during the search and often includes knowledge of valid paths.
Using known valid paths to solve an individual planning query takes less computational effort than finding a completely new solution. This allows multiquery algorithms, such as PRM*, to outperform single-query algorithms, such as RRT*, on many problems but their relative performance depends on how much information is reused. Despite this, few multiquery planners explicitly seek to maximize path reuse and, as a result, many do not consistently outperform single-query alternatives.
This paper presents Effort Informed Roadmaps (EIRM*), an almost-surely asymptotically optimal multiquery planning algorithm that explicitly prioritizes reusing computational effort. EIRM* uses an asymmetric bidirectional search to identify existing paths that may help solve an individual planning query and then uses this information to order its search and reduce computational effort. This allows it to find initial solutions up to an order-of-magnitude faster than state-of-the-art planning algorithms on the tested abstract and robotic multiquery planning problems.

Keywords: sampling-based path planning, optimal path planning, multiquery path planning

1 Introduction

A general-purpose path planner aims to find a path that connects a start to a goal, typically in a continuous space. The underlying structure of many environments is static and tends to pose repetitive problems, such as in home [9], construction [10, 14], or kitchen [22] scenarios. Multiquery planners are designed to solve multiple different start-goal queries in static environments by exploiting this repetitiveness to reduce the computational time required to find a solution. A large component of this computational effort for an individual planning query is checking if a path is collision free, i.e., validation effort [16, 29, 34]. This can be reduced in multiquery settings by reusing previously gained knowledge of valid edges to solve subsequent queries more efficiently.



(a) EIRM*, $t_{\text{init}} = 0.00437$s (b) LazyPRM*, $t_{\text{init}} = 2.481$s

Fig. 1: An illustration of EIRM* finding the initial solution orders-of-magnitude faster than LazyPRM* for the 20$^{\text{th}}$ query of a multiquery problem. The green and red disks are the start and the goal, respectively. Edges that are collision checked in this query are yellow, and edges that are reused are dark grey. Light grey edges are valid edges that have been collision checked in previous queries. The time of the initial solution is t_{init}. EIRM* validates two edges to connect to the preexisting graph while LazyPRM* validates 81 edges.

Many planners compute reusable information in advance. Probabilistic Roadmaps (PRM) [20] construct a roadmap and collision check all its edges during preprocessing. This roadmap is then used to simplify individual queries to a graph search over the roadmap, resulting in fast solution times.

If the environment is not available in advance, reusable information needs to be calculated in parallel to solving queries. LazyPRM [2] solves individual queries by optimistically assuming all edges and vertices in the roadmap are valid, and then only checking the edges and vertices of the solution that are not yet validated. It preserves the knowledge of validated and invalidated edges over multiple queries but does not *actively* reuse previously invested effort, so any reuse of effort is by coincidence.

PRM* [19] and LazyPRM* [16] extend these ideas to the optimal planning problem by continually adding more samples to their roadmaps to improve the approximation with additional computational time. This roadmap asymptotically contains the optimal solution with probability one, i.e., is almost-surely asymptotically optimal. While this graph growth improves solution quality, it can also increase initial solution times since new queries start from the previous approximation and searching large graphs can be prohibitively expensive.

This paper presents Effort Informed Roadmaps (EIRM*), an almost-surely asymptotically optimal anytime multiquery planner which seeks to to prioritize finding an initial solution by actively reusing effort from previous queries. EIRM* extends Effort Informed Trees (EIT*) [32] to the challenges of solving multiquery problems quickly by: *(i)* actively seeking to reuse computational effort, and *(ii)* managing graph size over multiple planning queries.

EIRM* quickly finds an initial solution to a planning query by using a search explicitly informed by validation effort. It then uses a cost-informed search to improve this solution by efficiently adding and searching more samples for as long as time allows for the current query. When a new planning query is posed, EIRM* prevents the complexity of this high-resolution graph from negatively affecting search performance by *rewinding* the approximation to the first batch of samples. It then reuses computational effort to solve this new query by both informing the initial search by validation effort and improv-

ing the solution by *replaying* the previous samples and reusing any previously validated edges. This allows EIRM* to find initial solutions to individual queries faster than other planners while almost-surely converging asymptotically to the same global optimum.

We compare EIRM* to other planners available in Open Motion Planning Library (OMPL) [33] on several low- and high-dimensional abstract environments, and simulated robots. On these problems, it solves later queries of a problem up to an order-of-magnitude faster than the tested planners while performing the same on initial queries.

2 Related Work

A general overview of sampling-based motion planning can be found in [8, 11] and a review of work using search effort, including the single-query Bayesian Effort-Aided Search Trees (BEAST) [21], can be found in [32]. This review focuses on multiquery planning, which aims to enable efficient planning over multiple queries by reusing knowledge gained from previous queries in the planning for the current query.

Many multiquery algorithms are based on Probabilistic Roadmaps (PRM) [20]. PRM randomly samples states in the configuration space of the robot to build a roadmap that discretizes the space. It validates all edges in the roadmap in a preprocessing phase before solving the first query. A query is then solved by a search over the roadmap.

LazyPRM [2] avoids the preprocessing and is more suitable for problems when the environment is not known in advance. It initially assumes all edges and vertices are valid, and searches over the graph to find a potential solution. It then collision checks the edges and vertices of this solution candidate that have not been collision checked in previous queries. This approach does not *actively* exploit the knowledge of collision checked edges and any reduction in planning effort is not deliberate.

PRM* [19] and LazyPRM* [16] extend PRM and LazyPRM to obtain almost-sure asymptotic optimality by adding new samples to the roadmap once a solution is found to improve the approximation of the environment. This allows them to probabilistically converge towards the global optimum but, together with the starts and goals from previous queries, these added samples increase the size of the roadmap in every query, and slow down subsequent queries.

Sparse Roadmap Spanners (SPARS) [7] address the unbounded growth of the graph by storing previously found paths in a sparse graph. The construction of the sparse graph is computationally expensive, and can lead to a slower planner overall. SPARS2 [6] improves the construction of the graph and reduces memory requirements, but it is still computationally expensive.

Experience Graphs (E-Graph) [27] contain previously found solutions and enable combining parts of these paths to find solutions to new queries. E-Graphs require running an all-pairs shortest path algorithm, which is computationally expensive. E-Graphs are extended to handle incremental anytime planning by continually updating cost heuristics [28]. This is also computationally expensive and does not always improve total planning time. The increasing graph size is not considered, which may lead to unsustainable graph sizes for high number of queries.

Lightning [1] and Thunder [5] are planning frameworks that store previous solutions in a database and attempt to modify these solutions to solve the current path planning

query. The database in Lightning can grow unbounded, which leads to slow information retrieval. Thunder accelerates information retrieval by storing solutions to previous planning problems in a sparse graph, which eliminates redundant information. As in SPARS, the complexity of inserting paths in the sparse graph can be computationally expensive. Lightning and Thunder only store and reuse solutions and do not retain otherwise validated or invalidated edges between queries.

A different way to avoid the graph growth is to use single-query path planners and modify them to reuse results from previous queries. Bruce and Veloso [3] bias Rapidly Exploring Random Trees (RRTs) [23] with samples from previously found paths that are stored in a waypoint cache. This cache has a fixed size and old samples are eventually forgotten, which requires problem-specific tuning. Other approaches to speed up the planning process learn to sample specific features (e.g., narrow passages) efficiently [4, 17, 18]. This does not actively reuse planning effort, and may require preprocessing to learn the sampling distribution.

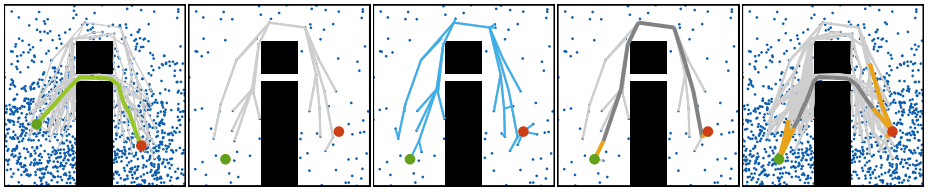
Reconfigurable Random Forests (RRF) [24] extend RRTs to the multiquery setting. RRF grows trees anchored at the start and the goal of a query towards the previously constructed trees, in order to iteratively construct a roadmap. This approach does not result in an anytime planner and does not give any optimality guarantees.

In comparison to the planners reviewed in this section, EIRM* actively tries to find initial solutions quickly. EIRM* estimates remaining validation effort to inform its initial search. This allows EIRM* to actively reuse previous computational effort to find initial solutions quickly, contrary to LazyPRM*. Calculating this effort heuristic is computationally inexpensive compared to the approach taken by E-Graphs. EIRM* prevents the work from earlier queries slowing down later queries by actively managing graph size. Unlike SPARS, EIRM* does this efficiently by rewinding the graph of each query to the initial approximation.

These approaches allow EIRM* to find initial solutions quickly and almost-surely converge asymptotically to the global optimum of each planning query.

3 Effort Informed Roadmaps (EIRM*)

EIRM* extends EIT* to the multiquery setting. EIT* is an almost-surely asymptotically optimal anytime sampling-based path planning algorithm that is based on an asymmetric search which simultaneously calculates and exploits problem-specific heuristics. Both EIT* and EIRM* sample batches of states, and view these states as a series of edge-implicit random geometric graphs (RGGs) [26], as in BIT* [13]. The edges in each RGG are processed in a reverse search informed by an *a priori* heuristic. The reverse search is computationally inexpensive since collisions are checked at a lower resolution than in the forward search, i.e., *sparsely* checked. The reverse search computes approximation-specific heuristic estimates of the cost and effort to reach the goal, and provides a lower bound on the resolution-optimal solution in the current RGG approximation. The forward search is guided by the cost and effort heuristics that are calculated in the reverse search. EIT* and EIRM* both compute a suboptimality bound by inflating the resolution-optimal solution cost and only consider edges that satisfy this suboptimality bound. During the forward search, edges are fully collision checked. For more details on EIT*, see [31].



(a) (b) (c) (d) (e)

Fig. 2: An illustration of EIRM* solving the 20th query of a multiquery problem. The states are blue dots, the start is the green disk, the goal is the red disk, previously validated edges are light grey, newly validated edges in this iteration are yellow, the reverse tree is light blue, and edges that are reused in the solution are dark grey. The solution to the 19th planning query is shown in green along with the available previously validated edges at this resolution, (a). The 20th query is solved by first resetting the approximation to the initial batch and its validated edges, (b). The reverse tree is then grown to the start, (c). Since the reverse search is initially ordered purely on effort, the previously validated edges (which have zero validation effort) are explored first. A solution is found in the current approximation by only validatating two new edges, (d). The approximation is continually refined and the search is ordered by cost to find better solutions, (e). Since later approximations replay previous samples they also include previously validated edges.

In the multiquery setting, the RGG likely contains validated edges from solving previous queries that would require zero validation effort to reuse in a solution. EIRM* leverages these zero-effort edges and avoids unbounded graph growth by modifying EIT*'s batch sampling (Section 3.2) and reverse search (Section 3.3) while using the same forward search (Section 3.4). The batch sampling is modified to rewind the approximation of each query to the initial batch of samples. This approximation is then improved by replaying the same batches of samples as in previous queries in order to reuse effort.

The reverse search of EIRM* differs from EIT* in that the search of each query is ordered by estimated validation effort until an initial solution is found. After finding an initial solution, EIRM* computes an admissible cost heuristic in its reverse search, and the forward search is ordered by cost. EIRM* is illustrated in Fig. 2, and algorithmic details are presented in Algs. 1 to 5, with modifications compared to EIT* in orange.

EIRM* maintains the almost-sure asymptotic optimality and probabilistic completeness of EIT*. Initially only considering the first batch of samples and later adding previous samples to improve the cost results in the same behaviour as EIT* as the number of samples goes to infinity and does not alter formal properties. The full proof for almost-sure asymptotic optimality for EIT*, which implies probabilistic completeness, is presented in [31].

3.1 Notation

We denote the search space as $X \subseteq \mathbb{R}^n$, with the subspace occupied by obstacles, $X_{\text{obs}} \subset X$, and the free space, $X_{\text{free}} = \text{closure}(X \setminus X_{\text{obs}})$. The i-th query consists of the start state, $x_{\text{start},i} \in X_{\text{free}}$, and a set of goal states, $X_{\text{goal},i} \subset X_{\text{free}}$. The states that make up our current RGG are stored in $X_{\text{RGG}} \subset X_{\text{free}}$. A path, π, consists of a series of states through the free space, $\forall s \in [0, 1],\ \pi(s) \in X_{\text{free}}$. The cost, c, of a path is $c(\pi) \in \mathbb{R}_{\geq 0}$. A solution to the i-th query is a path that starts at the start state and ends at a goal state, $\pi(0) = x_{\text{start},i}$ and $\pi(1) \in X_{\text{goal},i}$.

Algorithm 1: EIRM*: `plan_query`$(x_{\text{start},i}, X_{\text{goal},i})$
Changes compared to EIT* in orange.

$w \leftarrow \infty,\ i_{\text{buffer}} \leftarrow 0,\ c_{\text{curr}} \leftarrow \infty$
$X_{\text{RGG}} \leftarrow \{x_{\text{start},i}\} \cup X_{\text{goal},i}$
$X_{\text{RGG}} \xleftarrow{+} \mathcal{B}_{\text{start/goal}}$
$X_{\text{RGG}} \xleftarrow{+}$ `refine_approximation`(∞, m)
$\mathcal{Q}_{\mathcal{F}} \leftarrow$ `expand`$(x_{\text{start},i})$, $\mathcal{Q}_{\mathcal{R}} \leftarrow$ `expand`$(X_{\text{goal},i})$
do
  if `best_rev_edge_improves_sol`$(\mathcal{Q}_{\mathcal{R}}, w)$
    if $w = \infty$
      $(x_s, x_t) \leftarrow \arg\min_{(x_s,x_t)\in\mathcal{Q}_{\mathcal{R}}} k_{\mathcal{R}}^{\text{effort}}(x_s, x_t)$
    else
      $(x_s, x_t) \leftarrow \arg\min_{(x_s,x_t)\in\mathcal{Q}_{\mathcal{R}}} k_{\mathcal{R}}^{\text{cost}}(x_s, x_t)$
    $\mathcal{Q}_{\mathcal{R}} \xleftarrow{-} (x_s, x_t)$
    if `no_sparse_collision`(x_s, x_t)
      $\bar{h}[x_t] \leftarrow \min\{\bar{h}[x_t], \bar{h}[x_s] + \bar{c}(x_s, x_t)\}$
      if $(w = \infty$ **and** $\bar{e}[x_s] + \bar{e}(x_s, x_t) < \bar{e}[x_t])$
      or $(w \neq \infty$ **and** $\hat{h}[x_s] + \hat{c}(x_s, x_t) < \hat{h}[x_t])$
        $\hat{h}[x_t] = \min\{\hat{h}[x_t], \hat{h}[x_s] + \hat{c}(x_s, x_t)\}$
        $\bar{e}[x_t] = \min\{\bar{e}[x_t], \bar{e}[x_s] + \bar{e}(x_s, x_t)\}$
        $\mathcal{Q}_{\mathcal{R}} \xleftarrow{+}$ `expand`(x_t)
      else
        $E_{\text{invalid}} \xleftarrow{+} (x_s, x_t)$
  else if $\min_{(x_s,x_t)\in\mathcal{Q}_{\mathcal{F}}}\{g_{\mathcal{F}}(x_s) + \hat{c}(x_s, x_t) + \hat{h}[x_t]\} < c_{\text{curr}}$
    $(x_s, x_t) \leftarrow$ `get_best_forward_edge`$(\mathcal{Q}_{\mathcal{F}})$
    $\mathcal{Q}_{\mathcal{F}} \xleftarrow{-} (x_s, x_t)$
    if `collision_free`(x_s, x_t)
      $E_{\text{valid}} \xleftarrow{+} (x_s, x_t)$
      if $g_{\mathcal{F}}(x_s) + \hat{c}(x_s, x_t) + \hat{h}[x_t] < c_{\text{curr}})$
        $\mathcal{Q}_{\mathcal{F}} \xleftarrow{+}$ `expand`(x_t)
        if $x_t \in X_{\text{goal},i}$
          $c_{\text{curr}} \leftarrow \min_{x_g \in X_{\text{goal},i}}\{g_{\mathcal{F}}(x_g)\}$
          $w \leftarrow 1$
      else
        $E_{\text{invalid}} \xleftarrow{+} (x_s, x_t)$
  else
    $X_{\text{RGG}} \xleftarrow{+}$ `refine_approximation`(c_{curr}, m)
    $\mathcal{Q}_{\mathcal{F}} \leftarrow$ `expand`$(x_{\text{start},i})$, $\mathcal{Q}_{\mathcal{R}} \leftarrow$ `expand`$(X_{\text{goal},i})$
  $\mathcal{B}_{\text{start/goal}} \xleftarrow{+}$ `add_starts_goals`$(x_{\text{start},i}, X_{\text{goal},i})$
while not `stopped`

Algorithm 2: `expand`(X).

$E_{\text{out}} \leftarrow \emptyset$
foreach $x \in X$ **do**
  $X_{\text{near}} \leftarrow$ `nearest`(x, X_{RGG}, k)
  $X_{\text{near}} \xleftarrow{+} \{x_i | (x, x_i) \in E_{\text{valid}} \wedge x_i \in X_{\text{RGG}}\}$
  foreach $y \in X_{\text{near}}$ **do**
    if $(x, y) \notin E_{\text{invalid}}$ **and** $(y, x) \notin E_{\text{invalid}}$
      $E_{\text{out}} \xleftarrow{+} (x, y)$
return E_{out}

Algorithm 3: `refine_approximation`(c_{best}, m).

$M \leftarrow \emptyset$
while $|M| \leq m$ **do**
  if $i_{\text{buffer}} > |\mathcal{B}|$
    $\mathcal{B} \xleftarrow{+}$ `sample_valid_uniform`()
  if $\hat{f}(x_{i_{\text{buffer}}}) < c_{\text{best}}$
    $M \xleftarrow{+} x_{i_{\text{buffer}}}$
  $i_{\text{buffer}} \leftarrow i_{\text{buffer}} + 1$
return M

Algorithm 4: `best_rev_edge_improves_sol`$(\mathcal{Q}_{\mathcal{R}}, w)$.

if $w = \infty$
  if $\min_{(x_s,x_t)\in\mathcal{Q}_{\mathcal{R}}}\{k_{\mathcal{R}}^{\text{effort}}(x_s, x_t)\} \leq \min_{(x_s,x_t)\in\mathcal{Q}_{\mathcal{F}}}\{k_{\mathcal{F}}^{\text{effort}}(x_s, x_t)\}$
    return `true`
else
  if $\min_{(x_s,x_t)\in\mathcal{Q}_{\mathcal{R}}}\{k_{\mathcal{R}}^{\text{cost}}(x_s, x_t)\} \leq \min_{(x_s,x_t)\in\mathcal{Q}_{\mathcal{F}}}\{k_{\mathcal{F}}^{\text{cost}}(x_s, x_t)\}$
    return `true`
return `false`

Algorithm 5: `get_best_forward_edge`$(\mathcal{Q}_{\mathcal{F}})$.

$(x_s^*, x_t^*) \leftarrow \arg\min_{(x_s,x_t)\in\mathcal{S}}\{\bar{e}(x_s, x_t) + \bar{e}[x_t]\}$
if $g_{\mathcal{F}}(x_s^*) + \bar{c}(x_s^*, x_t^*) + \bar{h}[x_t^*] \leq w\hat{s}$
  return (x_s^*, x_t^*)
else if $\bar{s} \leq w\hat{s}$
  return $\arg\min_{(x_s,x_t)\in\mathcal{Q}_{\mathcal{F}}}\{g_{\mathcal{F}}(x_s) + \bar{c}(x_s, x_t) + \bar{h}[x_t]\}$
else
  return $\arg\min_{(x_s,x_t)\in\mathcal{Q}_{\mathcal{F}}}\{g_{\mathcal{F}}(x_s) + \hat{c}(x_s, x_t) + \hat{h}[x_t]\}$

We denote admissible estimates with a hat, $\hat{\cdot}$, possibly inadmissible estimates with a bar, $\bar{\cdot}$, and previously computed labels, i.e., real numbers attached to specific states, with square brackets, $l[\cdot]$.

Admissible and inadmissible cost estimates between two states, x_i and x_j, are denoted $\hat{c}(x_i, x_j)$, and $\bar{c}(x_i, x_j)$, respectively. We assume that the admissible cost estimate is lower than the inadmissible cost estimate, i.e., $\forall x_i, x_j \in X,\ \hat{c}(x_i, x_j) \leq \bar{c}(x_i, x_j)$.

We use $\hat{g}(x_t) = \hat{c}(x_{\text{start}}, x_t)$ as shorthand for an admissible cost heuristic to come to the target state, x_t, from the start, $\hat{h}(x_t) = \min_{x_g \in X_{\text{goal}}} \hat{c}(x_t, x_g)$ to denote an admissible cost heuristic to go from a target state to a goal state, and $\bar{h}(x_t) = \min_{x_g \in X_{\text{goal}}} \bar{c}(x_t, x_g)$ to denote an inadmissible heuristic to go from a target state to a goal state. An admissible estimate for the total cost for a path going through a state, x_t, is then given by $\hat{f}(x_t) = \hat{g}(x_t) + \hat{h}(x_t)$.

The possibly inadmissible estimate of the planning effort between two states, x_i and x_j, is $\bar{e}(x_i, x_j)$. The possibly inadmissible estimate for the effort to come to a target state, x_{t}, from the start state, x_{start}, is $\bar{d}(x_{\text{t}})$, which may be defined independently of $\bar{e}$.

For two sets, A and B, $A \xleftarrow{+} B$ denotes $A \leftarrow A \cup B$ and $A \xleftarrow{-} B$ denotes $A \leftarrow A \setminus B$. The cardinality of a set is denoted by $|\cdot|$.

3.2 Approximation

EIRM* incrementally adds batches of m states to the RGG to build a discrete approximation of the search space (Alg. 1, Line 35). Informed sampling [12] can be used to focus the approximation on the part of the space that can improve the solution once an initial solution to the current query has been found, if appropriate.

EIRM* considers connections between each sample and its k-nearest neighbours or states within a distance r as well as previously validated edges independent of their distance to the state currently under consideration (Alg. 2, Line 4).

If not handled explicitly, the size of the RGG will grow unbounded over the course of multiple queries. EIRM* manages the growth of the graph by rewinding the sampling-based approximation to the first batch to find the initial solution to each query and by pruning starts and goals from the graph to limit graph growth.

3.2.1 Batch Rewinding High-resolution approximations often contain high-quality solutions, but are computationally expensive to search due to the computational cost of the nearest-neighbour lookup and the required depth of the search. When fast solution times are desired, low-resolution approximations are often better since the computational cost of these operations reduce with the number of samples.

EIRM* stores all sampled states x_i for the duration of the multiquery problem in a buffer, $\mathcal{B} = (x_1, x_2, \cdots, x_n)$, and a new batch of samples is added to the approximation from the buffer when `refine_approximation` is called (Alg. 3). If the buffer does not contain enough samples, new states are first sampled and added to the buffer (Alg. 3, Line 4). Samples from the buffer are only added to the batch if they can improve the current solution (Alg. 3, Line 5). The current position in the buffer, i_{buffer}, is incremented as samples from the buffer are used (Alg. 3, Line 7), and is reset once a new planning query is considered (Alg. 1, Line 1).

3.2.2 Start/Goal Pruning The size of the graph will grow unbounded with the number of queries if all starts and goals of every query are kept in the graph. Keeping all starts and goals in the graph may also result in a nonuniform distribution of states if the starts and goals are not uniformly distributed.

Forgetting all starts and goals prevents unbounded graph growth and maintains the state sampling distribution, but discards the effort spent validating associated edges. These conflicting behaviours are balanced by keeping the starts or goals in a buffer, $\mathcal{B}_{\text{start/goal}}$, if they satisfy a user-specified criterion (Alg. 1, Line 37). The stored starts and goals are added to the RGG at the same time as the new start and goals of the current query (Alg. 1, Line 3) and the first batch of samples.

3.3 Reverse Search

EIRM* first tries to find a solution as quickly as possible, and then tries to decrease the cost as quickly as possible. The reverse search is therefore initially ordered on validation effort, and afterwards on cost.

The reverse search considers the best edge, starting at the source state, x_s, to the target state, x_t, from the edge-queue, $\mathcal{Q}_\mathcal{R}$. This queue is lexicographically ordered by effort if no solution is available,

$$k_\mathcal{R}^\text{effort}(x_\mathrm{s}, x_\mathrm{t}) = \left(\bar{e}[x_\mathrm{s}] + \bar{e}(x_\mathrm{s}, x_\mathrm{t}) + \bar{d}(x_\mathrm{t}), \hat{h}[x_\mathrm{s}] + \hat{c}(x_\mathrm{s}, x_\mathrm{t}) + \hat{g}(x_\mathrm{t})\right), \tag{1}$$

and by cost once a solution was found,

$$k_\mathcal{R}^\text{cost}(x_\mathrm{s}, x_\mathrm{t}) = \left(\hat{h}[x_\mathrm{s}] + \hat{c}(x_\mathrm{s}, x_\mathrm{t}) + \hat{g}(x_\mathrm{t}), \bar{e}[x_\mathrm{s}] + \bar{e}(x_\mathrm{s}, x_\mathrm{t}) + \bar{d}(x_\mathrm{t})\right), \tag{2}$$

where $\bar{d}$ and $\hat{g}$ are *a priori* estimates of the effort and cost to go, respectively. The elements of the key are estimates of the total computational effort and the total solution cost of a path through an edge. Their ordering depends on whether the current query already has a solution.

The edge with the lowest key is extracted (Alg. 1, Lines 8 to 11) and collisions are checked sparsely at evenly distributed states along the edge (Alg. 1, Line 13). If no collision is found, the computed cost heuristics, $\bar{h}[\cdot]$ and $\hat{h}[\cdot]$, and the effort heuristic, $\bar{e}[\cdot]$, of the target state, x_t, are updated (Alg. 1, Lines 14, 17 and 18). The target state is then expanded, and the edges to its neighbours are inserted into the reverse queue, (Alg. 1, Line 19) and the iteration restarts. If a collision was found, the edge is added to the set of invalid edges, E_invalid (Alg. 1, Line 21). The reverse search terminates when it is guaranteed to have found the resolution-optimal solution on the current RGG approximation or no solution is found, as in A* (Alg. 1, Line 7), where $k_\mathcal{F}^\text{effort}$ and $k_\mathcal{F}^\text{cost}$ are the lexicographical sortings of the forward search and are defined analogously to the respective reverse keys.

3.4 Forward Search

The forward search is based on anytime explicit estimation search (AEES) [35] and is guided by the calculated heuristics to effectively find solutions to each query. This search completely checks edges for collision and is more computationally expensive than the reverse search.

The previously computed admissible cost heuristic provides a lower bound, $\hat{s}$, on the resolution-optimal solution in the current RGG,

$$\hat{s} = \min_{(x_\mathrm{s}, x_\mathrm{t}) \in \mathcal{Q}_\mathcal{F}} \left\{ g_\mathcal{F}(x_\mathrm{s}) + \hat{c}(x_\mathrm{s}, x_\mathrm{t}) + \hat{h}[x_\mathrm{t}] \right\}, \tag{3}$$

where $g_\mathcal{F}(x_\mathrm{s})$ is the cost to come through the forward tree to the source state, x_s, and $Q_\mathcal{F}$ is the edge-queue of the forward search. A potentially more accurate estimate of the resolution-optimal cost can be calculated with the inadmissible cost heuristic,

$$\bar{s} = \min_{(x_\mathrm{s}, x_\mathrm{t}) \in \mathcal{Q}_\mathcal{F}} \left\{ g_\mathcal{F}(x_\mathrm{s}) + \bar{c}(x_\mathrm{s}, x_\mathrm{t}) + \bar{h}[x_\mathrm{t}] \right\}. \tag{4}$$

This possibly inadmissible estimate can be more accurate than its admissible counterpart since the inadmissible cost heuristic can use information that may overestimate the true cost. The focal set, $\mathcal{S}$, is the set of edges that can possibly lead to a solution within the current suboptimality bound, $w\bar{s}$,

$$\mathcal{S} = \left\{ (x_\mathrm{s}, x_\mathrm{t}) \mid g_\mathcal{F}(x_\mathrm{t}) + \bar{c}(x_\mathrm{t}, x_\mathrm{s}) + \bar{h}[x_\mathrm{s}] \leq w\bar{s} \right\}. \tag{5}$$

EIRM* expands the next edge (Alg. 1, Line 23, Alg. 5) considering the focal set and fully collision checks the edge (Alg. 1, Line 25). If the edge is found to be invalid, it is labeled as such (Alg. 1, Line 33) and the reverse search is restarted.

Edges are selected for expansion by first considering the minimum remaining validation effort in the focal set,

$$\underset{(x_\mathrm{s}, x_\mathrm{t}) \in \mathcal{S}}{\arg\min} \left\{ \bar{e}(x_\mathrm{s}, x_\mathrm{t}) + \bar{e}[x_\mathrm{t}] \right\}. \tag{6}$$

If this edge can improve the solution, it is selected (Alg. 5, Line 3). If not, the edge with the lowest inadmissible cost estimate is selected if it is estimated to lead to a solution within the current suboptimality bound (Alg. 5, Line 5). Otherwise, the edge with the lowest admissible cost estimate is selected (Alg. 5, Line 7).

The forward search continues until it is known that the best edge in the forward queue can not improve the solution (Alg. 1, Line 22), or a solution is found. If a solution is found, the best achieved cost is updated (Alg. 1, Line 30), the search is then ordered by cost by setting the suboptimality factor to one (Alg. 1, Line 31), the approximation is refined (Alg. 1, Line 35), and the loop restarts with the reverse search.

This search continues as long as time allows and almost-surely converges asymptotically to the optimal solution.

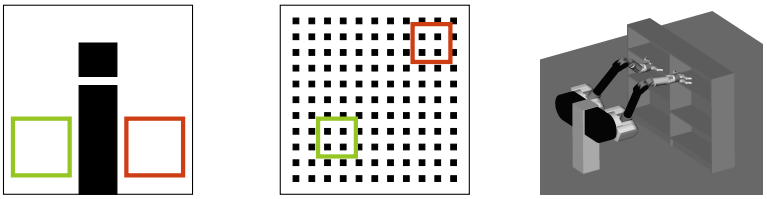
4 Experiments & Results

We evaluated EIRM* on a set of simulated scenarios[1], and compared it to a selection of both single- and multiquery planners available in OMPL: PRM*, LazyPRM*, RRT-Connect, RRT*, and EIT*. The OMPL version of SPARS/SPARS2 was not included due to performance.

RRT* used a goal bias of 0.05. Both RRT-based planners used maximum edge lengths of 0.3, 0.5, 1.25, and 2.4 in $\mathbb{R}^2$, $\mathbb{R}^4$, $\mathbb{R}^8$, and $\mathbb{R}^{14}$, respectively. EIRM* and EIT* used the k-nearest neighbour method and sampled 100 states per batch. The *a priori* heuristic for both admissible and inadmissible cost in EIRM* and EIT* was the Euclidean distance. The *a priori* heuristic for inadmissible effort between two states, i.e., $\bar{e}$, in EIT* and EIRM* was the Euclidean distance divided by the needed remaining collision checking resolution. In order to fully exploit the possibility of preexisting zero-effort edges, EIRM* used the zero heuristic for the inadmissible effort to come, i.e., $\bar{d}$.

In order to limit the growth of the graph, EIRM* kept starts and goals after a query if the number of required collision checks, i.e., the validation effort, to reach the state from the closest existing neighbour was larger than $50{,}000$ and otherwise forgot them.

[1] All experiments were run using OMPL 1.5, on a laptop with an Intel i7-4720HQ CPU @ 2.60GHz processor with 16GB RAM.



(a) Wall Gap (b) Repeating Rectangles (c) Bookshelf

Fig. 3: An illustration of the abstract scenarios in $\mathbb{R}^2$ with the subregions from which starts (green) and goals (red) are drawn uniformly, (a), (b). An illustration of the bookshelf scenario where the two-armed robot simulates picking/placing objects from the bookshelf, (c). Two versions of the repeating rectangles problem were considered, one in which start-goal queries are drawn from the subregions and the other in which they are drawn uniformly from the whole space.

Table 1: Cumulative median initial solution time, t_{init}, in seconds and cumulative median solution costs, c_{init} and c_{final}, for all evaluated planners on a selection of scenarios. The cumulative median of a value is the median value of 100 runs summed over 100 queries. The initial cost is the cost of the first solution, while the final cost is the cost at the end of the planning time. Planners not run on a specific scenario are marked with – and the **bold** value in each column is the best attained value. The ∞ indicates that it was not possible to compute the cumulative median for a planner due to the planner failing more than 50% of the time on any query. Confidence intervals are not reported here due to space constraints. Figures 4 and 5 are representative of the relative confidence intervals for the respective experiments.

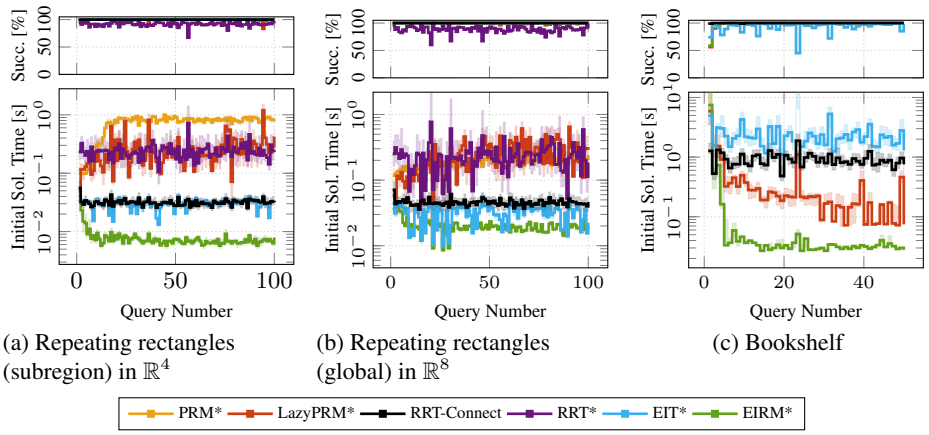
	Wall Gap (Fig. 3a)			Repeating Rectangles (Fig. 3b)						Bookshelf (Fig. 3c)		
				subregion			global					
	$\mathbb{R}^2$			$\mathbb{R}^4$			$\mathbb{R}^8$			$\mathbb{R}^{14}$		
	t_{init}	c_{init}	c_{final}	t_{init}	c_{init}	c_{final}	t_{init}	c_{init}	c_{final}	t_{init}	c_{init}	c_{final}
PRM*	4.21	98.9	98.1	73.9	137.2	136.4	18.6	206.4	204.8	–	–	–
LazyPRM*	2.34	**95.4**	**94.5**	25.7	**112.6**	**111.7**	24.0	**166.3**	165.0	20.5	**520.0**	**504.8**
RRT-Connect	1.53	185.1	185.1	3.18	163.5	163.5	4.55	257.7	257.7	45.9	884.2	884.2
RRT*	5.67	162.1	121.4	24.1	155.5	134.2	23.8	269.3	233.0	–	–	–
EIT*	1.51	145.9	95.9	3.00	157.9	112.1	3.21	200.8	**147.3**	∞	∞	∞
EIRM*	**0.28**	153.4	96.5	**0.77**	195.6	113.7	**2.0**	414.4	155.7	**10.4**	987.1	509.4

4.1 Abstract Scenarios

We considered two abstract scenarios with different obstacle configurations in $\mathbb{R}^2$, $\mathbb{R}^4$, and $\mathbb{R}^8$ (Fig. 3). The scenario in Fig. 3b was tested with the starts and goals were sampled uniformly at random from both subregions and sampled uniformly at random over the whole search space. The subregion scenario often occurs in construction or warehouse settings where robots move between two regions.

Each planner was run 100 times with different pseudorandom seeds on a multiquery problem consisting of a sequence of 100 different queries. The query sequence was defined for each problem by randomly sampling 100 starts and goals, and the same random sequence was used for all 100 runs of all planners. The maximum runtime per query was 0.5s, 2s, and 2s in $\mathbb{R}^2$, $\mathbb{R}^4$, and $\mathbb{R}^8$, respectively. The collision detection resolution was set to $5 \cdot 10^{-6}$ in the abstract problems to imitate the computational cost of collision checking of the robotic experiment, as in [31].

The median initial solution time per query along with confidence intervals for the repeating rectangles with subregion starts and goals in $\mathbb{R}^4$, and globally sampled starts



(a) Repeating rectangles (subregion) in $\mathbb{R}^4$

(b) Repeating rectangles (global) in $\mathbb{R}^8$

(c) Bookshelf

Fig. 4: The median initial solution times per query over 100 runs for the repeating rectangles with subregion starts and goals in $\mathbb{R}^4$, (a), globally sampled starts and goals in $\mathbb{R}^8$, (b), and the bookshelf scenario, (c). The solid line is the median initial solution time per query, and the shaded area is the nonparametric 99% confidence interval. Unsuccessful runs are treated as having infinite cost. For the bookshelf scenario, PRM* and RRT* were not run due to performance.

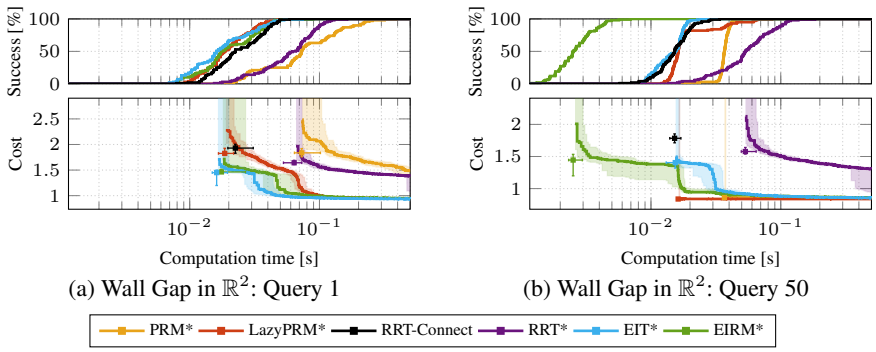
and goals in $\mathbb{R}^8$ are shown in Fig. 4. Table 1 summarizes both the cumulative median initial solution time across all queries (i.e., the integral of the plots shown in Fig. 4), and the corresponding cumulative median initial cost along with the cumulative median final cost of all planners. The evolution of the cost for the 1$^{\text{st}}$ and the 50$^{\text{th}}$ query on the example of the wall gap in $\mathbb{R}^2$ is presented in Fig. 5.

The initial solution time achieved by EIRM* is faster than the time achieved by all the other planners. The initial cost for the subregion scenarios is comparable to RRT-Connect, while in the globally sampled setting the cost is higher than the cost of the other planners since they use more computational time to find a initial solution. In both the subregion and the globally sampled start-goal scenarios, EIRM* converges to a solution that is similar to the other optimizing planners when given the same amount of computational time.

4.2 Robotic Scenario

We considered a two-armed robot ($\mathbb{R}^{14}$) with the queries chosen such that they simulate rearranging objects on a bookshelf (Fig. 3c). Each planner was run 100 times with different pseudorandom seeds on a sequence of 50 different queries and was run for 10s for each query. As for the abstract experiments, the sequence of random starts and goals was constant for all attempts. The Flexible Collision Library (FCL) [25] was used for collision checking, and the collision detection resolution was set to 0.036 for the bookshelf scenario, as in [31].

Figure 4c shows the initial solution time taken per query. Table 1 again summarizes the cumulative initial solution time across all queries and the corresponding cumulative median initial and final costs. EIRM* achieves up to an order-of-magnitude faster initial solutions for some queries and is approximately twice as fast cumulatively compared to the other planners.



(a) Wall Gap in $\mathbb{R}^2$: Query 1

(b) Wall Gap in $\mathbb{R}^2$: Query 50

Fig. 5: The planner performance in two queries for the wall gap in $\mathbb{R}^2$. The success plots (top) show the percentage of successful runs over time. The cost evolution plots (bottom) show the median cost at a given time as a thick line, with the nonparametric 99% confidence interval as shaded area. The squares show the median initial solution time for the query and the corresponding median initial cost. The absence of a solution is treated as having an infinite cost.

4.3 Initial Solutions

The relative benefits of explicitly reusing previous search effort and managing graph size are evaluated by limiting planners to only finding an initial solution. Not letting the planners run until convergence reduces the problems of unbounded graph growth for PRM* and LazyPRM*.

These experiments were run for the repeating rectangles scenario with subregion start-goal queries in $\mathbb{R}^8$ and for the bookshelf experiment. The experimental setup is the same as previously for both scenarios, but with early stopping after finding a solution.

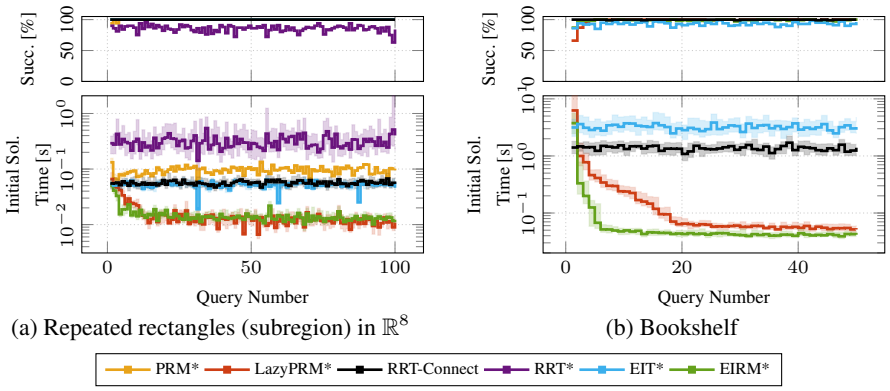
Figure 6 shows the initial-solution time plots for the experiments. EIRM* still achieves better median initial solution times in the bookshelf scenario and comparable times for the repeating rectangle scenario demonstrating the value of explicitly reusing information on more difficult problems.

4.4 Construction Scenario

Task and Motion Planning (TAMP) problems often pose multiquery scenarios where the environments changes and existing edges in the roadmap may be invalidated. We demonstrate a basic modification of EIRM* in a simplified construction setting [15] where a mobile manipulator ($\mathbb{R}^8$) stacks 36 bricks to build a wall (Fig. 7). The bricks that make up the wall are all the same, and are all picked up in the same location, simulating a conveyor belt that brings the bricks to the robot.

There are typically two planning problems in such a scenario. The first is picking up the bricks and stacking it on the wall, the second is returning to the pickup location. These two problems are often treated independently since the collision-checking envelope of the robot is different with and without a brick. We demonstrate the second scenario where the robot returns from placing a brick to pick up a new brick.

EIRM* was modified to remove the edges and vertices in its roadmap invalidated by newly placed bricks. It was not possible to efficiently make such modifications to



(a) Repeated rectangles (subregion) in $\mathbb{R}^8$ (b) Bookshelf

Fig. 6: The success plot (top) and the median initial solution times per query (bottom) for two experiments when the planning process was stopped as soon as an initial solution was found. In the success plot, the line indicates the number of runs that have solved the query at the end of the given planning time. In the median initial solution plot, the solid line is the median initial solution time per query, and the shaded area is the nonparametric 99% confidence interval. Unsuccessful runs are treated as having infinite cost. For the bookshelf scenario, PRM* and RRT* were not run due to performance.

PRM* and LazyPRM*, so EIRM* was only compared to the single-query planners which require no modifications, RRT-Connect and EIT*. The planners were run 25 times with different pseudorandom seeds on a 36 query sequence with 10s for each query. FCL was used for collision checking.

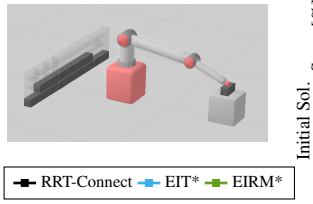
The median initial time plot (Fig. 7) is promising. While the time needed by RRT-Connect increases as the wall is built and the environment becomes more complex, the time taken by EIRM* decreases with the number of queries. Future work will focus on fully adapting EIRM* to changing environments by developing more efficient ways to remove invalidated edges and vertices.

5 Discussion

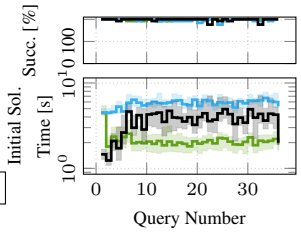
EIRM* consistently outperforms all tested planners in the time necessary to find an initial solution. The difference to other planners is most pronounced when queries are between subregions, since the previous paths are more likely to be part of future solutions. An improvement of the initial query time can still be observed in problems with uniformly distributed starts and goals. The cost of the quick initial solution from EIRM* is usually higher than the cost of the paths from other planners but the final cost is similar to the cost of other almost-surely asymptotically optimal planners.

5.1 Initial Solution Time

Figure 4 and Table 1 show that EIRM* finds initial solutions up to an order-of-magnitude faster than the other tested planners. It does this by explicitly seeking to reuse previous search effort and rewinding the approximation.



(a)



(b)

	t_{init}	c_{init}	c_{final}
RRT-Connect	131.2	**221.5**	221.5
EIT*	207.1	262.2	243.3
EIRM*	**76.3**	288.7	**214.2**

(c)

Fig. 7: An illustration of the construction scenario showing the mobile manipulate picking up a brick with translucent bricks illustrating the target position of future bricks, (a). The associated planner performance, (b), shows the success (top) and median initial solution times per query (bottom) over 25 runs and the culmulative median initial times and culmulative initial and final costs are shown in Table (c). In the success plot, the line indicates the percentage of runs that have solved the query in the given planning time. In the median initial solution plot, the solid line is the median initial solution time per query and the shaded area is the nonparametric 99% confidence interval. Unsuccessful runs are treated as having infinite cost.

LazyPRM* fails to solve some of the tested problems reliably due to the growing graph size when improving solution cost. In the experiments where the planners were stopped when an initial solution is found to limit this growth, EIRM* still achieves similar or better results than LazyPRM* on initial solution time. EIRM* also needs fewer queries to benefit from previously invested effort compared to LazyPRM* since EIRM* explicitly tries to reuse validated edges.

Rewinding the approximation of the environment to the first batch of samples makes the performance of the planner independent of the previous query's final resolution since every query starts from a coarse resolution. This may remove important paths that were found in later approximations, e.g., narrow passages. Future work could investigate promoting *promising* samples to earlier batches by ordering the samples in the buffer with an *importance* metric. This could lead to both quicker and higher quality initial solutions. Similarly, pruning starts and goals too aggressively might lead to a loss of invested effort. In future work, we intend to investigate the start and goal pruning method.

It might be beneficial to explore other heuristics for the possibly inadmissible effort, and the stopping conditions for the effort ordered reverse search. We noticed actively reducing validation effort means that in some experiments validation effort is no longer the main computational cost of EIRM*. This suggests that future work could include other time intensive steps of the algorithm in the effort heuristic, e.g., nearest neighbour lookups, which took up to 30% of the planning time in our setting. We currently run the reverse search until no solution candidate with a lower remaining validation effort exists. It might be faster overall to stop the reverse search earlier, and use an earlier solution candidate even if it may not be the path with the minimum remaining validation effort.

5.2 Objective Value

Table 1 reports the initial and final costs of the solution. The initial path cost found by EIRM* is usually higher than LazyPRM*, but its final cost is within a few percentage points of the best found solution. In some cases, EIRM* appeared not to converge

efficiently to the best solution when the optimal solution was close to the straight-line path. This may be due to rejection sampling from the sample buffer to obtain informed samples when refining the RGG. Future work may consider how to efficiently sample the informed set while maintaining the uniform distribution of samples in the buffer.

If a suboptimal solution is acceptable, it might be desirable to smoothly interpolate between an effort-ordered and a cost-ordered search to allow for more path reuse. This could be achieved with multi-objective A* [30]. The labels for the cost and effort would then not only depend on the state itself, but also on which path was taken to get to the state. It is future work to investigate how to best incorporate this approach in EIRM*.

6 Conclusion

Multiquery planners aim to efficiently solve multiple diverse motion planning problems in the same environment. This is generally achieved by keeping the approximation built during the previous queries. This can speed up the planning process, but few planners fully exploit the invested effort.

This paper presents EIRM*, a planner that explicitly aims to find paths with a low remaining validation effort. This is achieved by using an asymmetric search that calculates cost and effort heuristics in a computationally cheap reverse search. The heuristics are then used to guide the forward search, in which the edges are fully collision checked.

EIRM* demonstrates that explicitly reusing computational effort and managing graph size between queries finds an initial solution quickly and then rapidly improves it. This is shown to outperform existing state-of-the-art planners on initial solution time while achieving similar solution quality on multiple different planning scenarios consisting of low- and high-dimensional abstract problems and robotic simulations. Information on the OMPL implementation of EIRM* is available at https://robotic-esp.com/code.

Acknowledgement This research has been supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2120/1 – 390831618 and UK Research and Innovation and EPSRC through ACE-OPS: From Autonomy to Cognitive assistance in Emergency OPerationS [EP/S030832/1].

References

1. Berenson D, Abbeel P, Goldberg K (2012) A robot path planning framework that learns from experience. In: ICRA, pp 3671–3678
2. Bohlin R, Kavraki LE (2000) Path planning using lazy PRM. In: ICRA, pp 521–528
3. Bruce J, Veloso MM (2002) Real-time randomized path planning for robot navigation. In: Robot Soccer World Cup, Springer, pp 288–295
4. Chen B, Dai B, Lin Q, Ye G, Liu H, Song L (2020) Learning to Plan in High Dimensions via Neural Exploration-Exploitation Trees. In: ICLR
5. Coleman D, Şucan IA, Moll M, Okada K, Correll N (2015) Experience-based planning with sparse roadmap spanners. In: ICRA, pp 900–905
6. Dobson A, Bekris KE (2014) Sparse roadmap spanners for asymptotically near-optimal motion planning. IJRR pp 18–47
7. Dobson A, Krontiris A, Bekris KE (2012) Sparse Roadmap Spanners. In: WAFR, pp 279–296
8. Elbanhawi M, Simic M (2014) Sampling-based robot motion planning: A review. Ieee access

9. Faust A, Ramirez O, Fiser M, Oslund K, Francis A, Davidson J, Tapia L (2018) PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-based Planning. In: ICRA, pp 5113–5120
10. Funk N, Chalvatzaki G, Belousov B, Peters J (2021) Learn2Assemble with Structured Representations and Search for Robotic Architectural Construction. In: CoRL
11. Gammell JD, Strub MP (2021) Asymptotically optimal sampling-based motion planning methods. Annual Review of Control, Robotics, and Autonomous Systems pp 295–318
12. Gammell JD, Barfoot TD, Srinivasa SS (2018) Informed sampling for asymptotically optimal path planning. T-RO pp 966–984
13. Gammell JD, Barfoot TD, Srinivasa SS (2020) Batch Informed Trees (BIT*): Informed asymptotically optimal anytime search. IJRR pp 543–567
14. Hartmann VN, Oguz OS, Driess D, Toussaint M, Menges A (2020) Robust Task and Motion Planning for Long-Horizon Architectural Construction Planning. In: IROS, pp 6886–6893
15. Hartmann VN, Orthey A, Driess D, Oguz OS, Toussaint M (2021) Long-horizon multi-robot rearrangement planning for construction assembly. To Appear in T-RO
16. Hauser K (2015) Lazy collision checking in asymptotically-optimal motion planning. In: ICRA, pp 2951–2957
17. Ichter B, Harrison J, Pavone M (2018) Learning sampling distributions for robot motion planning. In: ICRA, pp 7087–7094
18. Ichter B, Schmerling E, Lee TWE, Faust A (2020) Learned critical probabilistic roadmaps for robotic motion planning. In: ICRA, pp 9535–9541
19. Karaman S, Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. IJRR pp 846–894
20. Kavraki LE, Svestka P, Latombe JC, Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. T-RO pp 566–580
21. Kiesel S, Gu T, Ruml W (2017) An effort bias for sampling-based motion planning. In: IROS, pp 2864–2871
22. Lagriffoul F, Dantam NT, Garrett C, Akbari A, Srivastava S, Kavraki LE (2018) Platform-independent benchmarks for task and motion planning. RA-L pp 3765–3772
23. Lavalle SM (1998) Rapidly-exploring random trees: A new tool for path planning. Tech. rep.
24. Li TY, Shie YC (2002) An incremental learning approach to motion planning with roadmap management. In: ICRA, pp 3411–3416
25. Pan J, Chitta S, Manocha D (2012) FCL: A general purpose library for collision and proximity queries. In: ICRA, pp 3859–3866
26. Penrose M (2003) Random geometric graphs. OUP Oxford
27. Phillips M, Cohen BJ, Chitta S, Likhachev M (2012) E-Graphs: Bootstrapping Planning with Experience Graphs. In: R:SS, p 110
28. Phillips M, Dornbush A, Chitta S, Likhachev M (2013) Anytime incremental planning with e-graphs. In: ICRA, pp 2444–2451
29. Solovey K, Kleinbort M (2020) The critical radius in sampling-based motion planning. IJRR pp 266–285
30. Stewart BS, White CC (1991) Multiobjective A*. Journal of the ACM p 775–814
31. Strub MP (2021) Leveraging multiple sources of information to search continuous spaces. PhD thesis, University of Oxford
32. Strub MP, Gammell JD (2022) AIT* and EIT*: Asymmetric bidirectional sampling-based path planning. To appear in IJRR
33. Şucan IA, Moll M, Kavraki LE (2012) The Open Motion Planning Library. RAM pp 72–82
34. Sánchez G, Latombe JC (2001) A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking. In: ISRR
35. Thayer J, Benton J, Helmert M (2012) Better parameter-free anytime search by minimizing time between solutions. In: SOcS