Effort Level Search in Infinite Completion Trees with Application to Task-and-Motion Planning

Marc Toussaint[1], Joaquim Ortiz-Haro[1,2], Valentin N. Hartmann[1], Erez Karpas[3], Wolfgang Hönig[1]

***Abstract*— Solving a Task-and-Motion Planning (TAMP) problem can be represented as a sequential (meta-) decision process, where early decisions concern the skeleton (sequence of logic actions) and later decisions concern what to compute for such skeletons (e.g., action parameters, bounds, RRT paths, or full optimal manipulation trajectories). We consider the general problem of how to schedule compute effort in such hierarchical solution processes. More specifically, we introduce infinite completion trees as a problem formalization, where before we can expand or evaluate a node, we have to solve a preemptible computational sub-problem of a priori unknown compute effort. Infinite branchings represent an infinite choice of random initializations of computational sub-problems. Decision making in such trees means to decide on where to invest compute or where to widen a branch. We propose a heuristic to balance branching width and compute depth using polynomial level sets. We show completeness of the resulting solver and that a round robin baseline strategy used previously for TAMP becomes a special case. Experiments confirm the robustness and efficiency of the method on problems including stochastic bandits and a suite of TAMP problems, and compare our approach to a round robin baseline. An appendix comparing the framework to bandit methods and proposing a corresponding tree policy version is found on the supplementary webpage[1].**

I. Introduction

When designing robotics planning algorithms we are often faced with decisions as follows: Should the algorithm try to directly use path optimization to find a feasible path, or a sampling-based path finder? When a path planner did not find a path after a second, should it give up and instead try to solve for a completely different action sequence? When the algorithm already sampled five different feasible grasp poses (using a constraint solver) but for none of these a consistent approach path was found, should it try a sixth pose (with a new random seed) or instead try an alternative action sequence?

Our view is that decisions as these should be made by the algorithm itself, rather than hard-coded in the algorithm's design. In this paper, we provide a general problem formulation to address such meta decision making, essentially enabling a search over alternative compute paths to a solution.

More specifically, we consider problems where finding an overall solution requires to solve a sequence of sub-problems, akin to "assembling a solution" in several steps. Sub-problem solvers are assumed to be preemptible, i.e., their computation can be paused to reallocate compute effort at alternative places. The possible sequences of sub-problems form a decision tree. Nodes in this tree represent sub-problems, and we can expand a node only when it is *complete*, i.e., after its sub-problem is solved. Further, a node may have infinite potential children, namely to enumerate random seeds of sub-solvers and thereby ensuring probabilistic completeness of the overall solver. We call the resulting decision tree an *infinite completion tree*. A solver needs to incrementally forward search the tree in order to find a solution while minimizing the expected compute effort. To this end, it needs to decide where to invest compute effort by preempting sub-solvers and reassigning compute effort to other sub-problems, or by considering alternative compute paths.

In this setting we propose *Effort Level Search* (ELS), which balances allocation of compute effort in completion trees with infinite branching. The proposed heuristic defines level sets polynomial in (compute) depth and (branching) width. Our contributions include:

1) The introduction of completion trees as a novel problem formulation for sequential computational solvers that preemptively assign compute efforts to sub-problems, which also generalizes previous effortful bandit formulations.
2) Effort Level Search (ELS) to balance search depth and width in such trees, guaranteeing completeness, with Round Robin as special case.
3) A tree policy variant of Effort Level Search that generalizes UCT (Upper Confidence applied to Trees) and can address effort bandit settings.

We evaluate ELS on analytic problems that test for robustness, as well as a suite of 8 robotic TAMP problems.

II. Related Work

A. Classical Search & Meta-Reasoning

In our problem formulation we need to reason over infinite branching and decisions on investing compute effort. Partial Expansion A* [3], [6] can handle problems with an infinite branching factor. However, these do not consider the effort of evaluating nodes. Lazy Receding Horizon A* manages to search in graphs with expensive edge evaluations [9]. Each edge evaluation can be viewed as a sub-problem to be solved, but this algorithm assumes the structure of the graph (without edge costs) is known and interrupting completion of an edge (or conversely, deciding on which sub-problem to resume investing compute) is not considered.

[1]LIS Lab, TU Berlin, Germany
[2]Machines in Motion Laboratory, New York University, USA
[3]Cognitive Robotics Lab, Technion, Israel

This research has been supported by the German Research Foundation (DFG) under Germany's Excellence Strategy – EXC 2002/1–390523135 *Science of Intelligence*, as well as the German-Israeli Foundation for Scientific Research (GIF), grant I-1491-407.6/2019.

[1]https://www.user.tu-berlin.de/mtoussai/24-CompletionTrees/

Generally, such decision problems are related to rational meta-reasoning [12], which has been used before in heuristic search [8], [10], [13]. However, these algorithms assume the search tree is finite.

B. Bandits with Compute Costs and Interrupts

There are several problem formulations that combine bandits with compute efforts or resource limits. Bandits with knapsacks [1], [16] assumes that each query of a bandit consumes resources on which arbitrary constraints can be formulated. The decision strategies roughly follow the idea of "bang for buck", i.e., based on expected return per expected effort. Unlike in our formulation, whenever a decision is made, the full resources are immediately consumed and a return received – there is no notion of interrupting compute or completion.

In contrast, [2] considers a problem formulation more closely related to ours, where whenever a bandit is queried anew (in other words, a new task of type k spawned), a latent compute cost and return is sampled. The decision maker can at any time interrupt a bandit query (task computation) and decide to continue with another. This formulation introduces the notion of an *incomplete bandit*, which we adopt in our work. However, in our framework returns are only received at completion of leaf nodes in a hierarchical tree, and branchings in this tree can be infinite.

C. Planners in Robotics

In [5], a general overview on task-and-motion planning (TAMP) solvers is provided, in which decisions on investing compute efforts are inherent. In [7] a TAMP solver was proposed that allocates compute time in a round robin strategy to various sampling based path planners, where the time is iteratively increased. Multi-bound tree search (MBTS) [15] introduces various levels of admissible bounds which guide search and thereby compute investment. The sub-problems are similar to the ones we consider here, but the search strategy through the tree is fixed, and the computations are not preemptible.

In [11], the problem of finding feasible keyframes in a TAMP problem, i.e., a handover-pose or a picking-pose of a robot, was modeled as a factor graph, and UCT was applied to find an efficient sampling strategy. Similarly, in [4], the TAMP problem was modelled as a factor graph, and two algorithms to traverse the factor graph are proposed. As in other work, the computations are not preemptible, and the search strategy is manually designed.

III. Problem Formulation: Infinite Completion Trees

We first define the abstract notion of a completion tree and later discuss how it represents our problem setting. A completion tree is defined by a root node n_0 and a successor function succ(n, i) that returns the i^{th} child node of n. Every node n has two fixed properties $(\hat{c}_n, B_n)$:

- the latent (a priori unknown) compute effort $\hat{c}_n \in \mathbb{R}_+ \cup \{\infty\}$, and
- its branching $B_n \in \mathbb{N} \cup \{\infty\}$ (i.e., number of children).

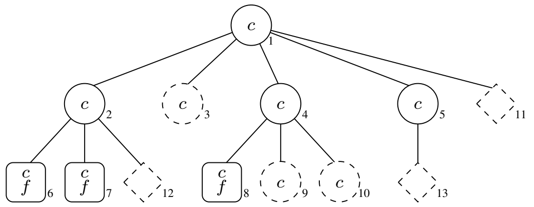

Fig. 1. Partially expanded Completion Tree. While the underlying decision tree has infinite branching, the state of search can be represented as a partially expanded compute tree with incomplete nodes (dashed), complete terminal nodes (boxes), and widening candidates at infinite branchings (diamonds). c_n indicates the total compute yet invested in a node n; y_n is the return at terminal nodes. The solver needs to decide on a next dashed node to invest a unit compute budget σ.

The successor function succ(n, i) is defined only for integers $0 < i \le B_n$. A node with $B_n = 0$ is called *terminal*. A terminal node n has an additional fixed property y_n:

- the latent terminal return value $y_n \in \mathbb{R}$.

Every node n in addition has a *state* variable $c_n \in \mathbb{R}_+$, which is the total compute invested in the node, and initialized to $c_n = 0$. When $c_n \ge \hat{c}_n$ exceeds the latent compute effort, the node is called *complete*.

The problem is to forward search this potentially infinite tree under the key constraint that the compute efforts $\hat{c}_n$ and terminal returns y_n are a priori unknown and the only way to identify their value is to complete the nodes by investing compute effort. Compute effort can only be invested if the parent node is complete, and initially, only the root node n_0 is complete.

In each iteration, the search algorithm decides on an incomplete node n that has a complete parent in which a small unit compute budget σ is invested. If $c_n + \sigma \ge \hat{c}_n$, the node n becomes complete and $c_n \leftarrow \hat{c}_n$; otherwise it remains incomplete and $c_n \leftarrow c_n + \sigma$. To this end, a search algorithm can build a partially expanded tree that includes the completed nodes and their children, which represents the decision space, see Fig. 1.

The above is an anytime, non-terminating search process. To define an objective for search, let us index decision iterations with $t = 1, ...$ After t iterations, let $C(t)$ be the total compute effort invested in the tree so far; and conversely, let $t(\hat{C}) = \min\{t : C(t) \ge \hat{C}\}$ be the iteration where $\hat{C}$ is exceeded.[2] Further, let $S(t)$ be the set of terminal nodes (with $y_n > -\infty$) found after iteration t, and $Y(t) = \{y_n : i \in S(t)\}$ their returns. We define the metrics:

- *total compute to first solution:*

$$C_{\text{first}} = C(\min\{t : |Y(t)| > 0\}) \ , \tag{1}$$

- *best return at given compute $\hat{C}$:*

$$y_{\text{best}}(\hat{C}) = \max Y(t(\hat{C})) \ , \tag{2}$$

- *mean return at given compute $\hat{C}$:*

$$y_{\text{mean}}(\hat{C}) = \tfrac{1}{|Y(t)|} \textstyle\sum_{y \in Y(t(\hat{C}))} y \ . \tag{3}$$

[2]Typically $C(t) \approx t\sigma$, but in practice less as some completions happen before a unit compute budget σ was consumed.

As a remark, the assumption that nodes of underlying completion trees have fixed (deterministic) properties (latent compute effort $\hat{c}_n$ and terminal return y_n) might seem limiting. However, in the appendix[1] A we discuss how effortfull bandits can be represented in our formulation. We further discuss that the notion of regret is not trivial to transfer, which is why we use the metrics above.

Further, our assumption that each iteration invests a small unit compute budget σ (e.g., $\sigma = \frac{1}{100}$sec) is an approximation of real time decision making on premptively pausing computations and reassigning compute.

A. Relation to Robotic Planning

The abstract problem formulation provided above is fully deterministic and it might be unclear how this relates to sample- and optimization-based solvers in robotics.

The computational problem associated to a node could be a non-linear mathematical program (NLP) or a path finding problem. While solvers are typically randomized (e.g. when initializing an NLP solver), we generally consider the pseudo random seed at a node to be fixed. Thus, given the fixed random seed, a node has a deterministic (but unknown) compute effort $\hat{c}_n$ to be solved. We allow $\hat{c}_n = \infty$ to represent that a solver does not terminate. As in preemptive scheduling we assume the problem solver can be paused after compute effort σ and resumed at the next call. At termination of the solver, the node becomes complete and we observe the final return y_n (negative cost) at terminal nodes, which can also be $-\infty$ if the NLP is infeasible.

Random initialization of sub-solvers is our core motivation for considering infinite branching: Solvers often require sampling streams or restarts with various random initialization to ensure probabilistic completeness. In a completion tree this is represented by considering the choice of random seed as an explicit branching: branchings in our tree not only concern which computational problem to solve next, but also which random seed to choose to initialize the solver. Generally, anything usually represented probabilistically is, in our framework, represented by infinite branching (including random returns or compute efforts for a bandit).

Section V-B will explicitly define the computational modules represented by nodes when we evaluate the application of Completion Trees to TAMP solving.

IV. Effort Level Search

When deciding on a node to invest compute, the choices can be rather heterogeneous, e.g. between a node we have already invested in (e.g. node 3 in Fig. 1) or a novel node (node 11), between nodes at the same level or across levels, or between nodes with more or less data of siblings available (e.g. node 12 vs. 13). The challenge is to define a single score to prioritize between such decisions. Our approach defines effort level sets that prioritize decisions in the tree and systematically expands along level sets. We show completeness and that a round robin approach [7] becomes a special case. In appendix[1] D we also introduce a tree policy approach which prioritizes decisions within a layer and uses a tree policy as in Monte-Carlo Tree Search (MCTS) to eventually decide on an incomplete node, which helps to relate our proposed methods to MCTS approaches and bandit settings.

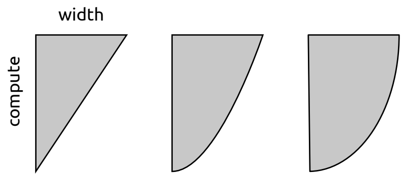

Fig. 2. Illustration of effort level sets: Left with linear compute and widening; middle with square widening penalty; right with both square.

A. Widening, Deepening, and Effort Level-sets

We introduce an *effort level*, which combines penalization of compute depth and widening. Specifically, we define the single-decision effort e_n for node n as the sum

$$e_n = D(c_n) + W_{\text{par}(n)}(j_n) \tag{4}$$

where $D(\cdot)$ is a function that penalizes compute depth, and $W_\cdot(\cdot)$ is a function that penalizes widening and depends on the enumeration j_n of a child[3] and potentially also on the parent par(n). Based on this, we introduce the effort-level E_n at node n as the sum

$$E_n = \textstyle\sum_{k \in \pi(n)} e_k \tag{5}$$

along the path $\pi(n)$ from the root to node n, including n.

In the simplest case we might choose linear deepening and widening penalties, i.e., $e_n = c_n + \gamma j_n$, which leads to a level set as illustrated in Fig. 2(left), where width means the sum of enumerations $\sum_{k \in \pi(n)} j_k$, and compute the total compute cost $\sum_{k \in \pi(n)} c_k$. In the illustration, search goes deep in nodes with small enumeration j_n (left edge), and wide for shallow nodes, where deep and shallow refer to total compute costs.

However, other choices of the effort and widening penalties allow us to express other priors about the search. In particular, at an infinite branching we may intepret children to be i.i.d. samples of an underlying distribution, akin to individual samples of a bandit represented by the parent node. From Upper Confidence Bound (UCB) methods we learn that the explorative value of taking more and more samples from the same bandit decreases, typically with $1/\sqrt{n}$. Translating this to our case we should increasingly penalize widening at a branching as the chance of finding better solutions becomes smaller. In essence this motivates the option to choose alternative widening penalties that increase polynomially with the enumeration, $W(j_n) = (j_n)^p$. Fig. 2(middle) aims to illustrate a quadratic choice $p = 2$ (but oversimplifies as it neglects that our effort is a sum of square widening penalties rather than a square penalty of total width).

[3]We assume children to be enumerated with $j = 1, ..$ below a parent. In the case of $B_n = \infty$ we assume that $W_{\text{par}(n)}(j_n)$ (and any other heuristic) is *non-decreasing* with increasing j. For instance, when the branching relates to a choice of random seed, we expect all children to have an equal lower bound, as the underlying problems are i.i.d. samples of the same random problem.

Algorithm 1 Generalized Search with incomplete nodes

Generic search scheme (akin to A*) adapted to the case of incomplete nodes, where nodes maintain a prioritization $f_n \in \mathbb{R}^+ \cup \infty$, and have indicator methods *isTerminal* and *isComplete*. Further, nodes have methods WIDEN and DEEPEN to return sibling and child nodes for expansion (see below), and a method COMPUTE to update its state.

Input: root node, added into priority queue
Output: sorted list of solutions, initialized empty

```
1: repeat
2:     $n \leftarrow$pop a node with lowest $f_n$ from queue
3:     insert set WIDEN($n$) in queue
4:     COMPUTE($n$), which updates $f_n$
5:     if not isComplete($n$): reinsert $n$ in queue
6:     elif isTerminal($n$): append $n$ to solutions
7:     else: insert the set DEEPEN($n$) of children in queue
8: until queue empty or time-out
```

Algorithm 2 Effort Level Search (ELS)

Equals Generalized Search with the following instances of WIDEN, COMPUTE, and DEEPEN that are based on a node's effort level E_n and parent's branching factor $B_{\text{par}(n)}$:

```
1: function WIDEN($n$)
2:     if $c_n = 0$ and $B_{\text{par}(n)} = \infty$:
3:         return new sibling $k$ with
               $E_k = E_{\text{par}(n)} + e_k$
4:     else: return $\emptyset$
5: end function
6: function COMPUTE($n$)
7:     invest a unit compute budget on problem $n$
8:     increment $c_n$ by actual compute time
9:     update $e_n, E_n$
10: end function
11: function DEEPEN($n$)
12:     if $B_{\text{par}(n)} < \infty$: return all child nodes $k$ with
               $E_k = E_n + e_k$
13:     else: return set with only first child node $k$ with
               $E_k = E_n + e_k$
14: end function
```

Concerning the compute cost of a single problem we might have a prior that costs up to a level c_0 are normal and not to be penalized harshly, while compute costs significantly beyond c_0 might indicate a non-convergent solver, with the chance of finding better solutions becoming smaller and smaller, and therefore should be penalized super-linearly, see Fig. 2 (right). We therefore generally propose widening and deepening penalties of the form

$$e_n = (c_n/c_0)^{p_c} + (j_n/w_0)^{p_w} + \epsilon \tag{6}$$

with parameters $c_0, p_c, w_0, p_w, \epsilon$, where $\epsilon > 0$ describes a small penalty for each level, e.g., to represent the cost of node creation itself.

Algorithms 1 and 2 describes Effort Level Search (ELS) in terms of a generalized search scheme, which includes widening, investing compute, and deepening. The new single node generated via WIDEN (line 3) and DEEPEN (line 13) with $c_n = 0$ corresponds to the diamond nodes in figures.

B. Analysis

1) Solution optimality vs. compute path optimality vs. search optimality: We should distinguish various notions of optimality: The optimality of found solutions y_n (and related to this, completeness of search); the "effort optimality" of the compute path from root to a found solution node n; and the total effort (spent in the full completion tree) that search required to find solutions.

A basic implication of ELS being a level set method is:

Proposition 1 (Completeness and solution optimality): Assuming an optimal solution with finite effort level and finite return exists, and assuming unbounded increasing widening penalty $W(\cdot)$ for infinite branchings (i.e., $W(j)$ increases with j and $\lim_{j\to\infty} W(j) = \infty$.), then ELS will find an optimal solution in finite time.

The proof (given in appendix[1] B) is straight-forward (as with A*), showing that any terminal with finite effort is reached with finite total invested compute. But as we distinguish between effort and return, this does not mean that the best (in our case, lowest return) solution is returned first. The above only proves that it finds an optimal solution eventually. So it would be ambiguous to call ELS optimal in a traditional search algorithm sense. Instead, we might characterize it as returning effort optimal solutions first.

2) Special Case: Round Robin: Hauser et al. [7] previously proposed to fairly distribute available compute to all open jobs in a round robin manner, and interleave this with creating new compute jobs (in our terms, widening) at the end of each round. We show here that this can be (approximately) cast a special case of ELS with linear widening and deepening penalties. Approximately here merely means neglecting effects that arise from the discretization unit σ of compute budgets; we have exact equivalence if $\hat{c}_n$ are multiple of σ so that all compute calls make full use of the provided unit compute budget σ.

In our completion tree representation, the round robin scheme is easily realized by a FIFO queue of all nodes, including the diamond node. While the previous work [7] presented the scheme only in the flat case, our completion tree representation generalizes directly also to hierarchical computation. Alg. 3 describes the FIFO method; a detail is that when a diamond node is called and remains incomplete, it inserts itself again into the FIFO *before* its sibling is inserted (lines 7 and 8). We can count *rounds* explicitly by imagining a new marker being inserted into the FIFO whenever a marker is popped – each round calls all open nodes once. The proof (appendix[1] C) shows by induction that at the end of round N, for each node n in the FIFO, the sum of the number of compute steps $\sum_{k\in\pi(n)} c_k/\sigma$ along its path $\pi(n)$ plus the sum of widening enumerations $\sum_{k\in\pi(n)} j_k$ along its path equals N,

$$N = \textstyle\sum_{k\in\pi(n)} c_k/\sigma + \sum_{k\in\pi(n)} (j_k - 1) \ . \tag{7}$$

Algorithm 3 Round Robin

Shown to be (approximately) equivalent to ELS with linear widening and deepening penalties and no effort heuristics

Input: root node, added into FIFO queue

Output: set of solutions, initialized empty

```
1: repeat
2:     $n \leftarrow$pop queue
3:     if $c_n = 0$ ($n$ is diamond): flag = true, else: flag = false
4:     COMPUTE($n$), which updates $f_n$
5:     if not isComplete($n$): reinsert $n$ in FIFO
6:     elif isTerminal($n$): insert $n$ in solutions
7:     else: insert (diamond) child in FIFO
8:     if flag: insert sibling $k$ in FIFO
9: until queue empty or time-out
```

The round number N therefore is equivalent to an effort level with linear compute and widening penalty, namely $c_0 = \sigma, p_c = w_0 = p_w = 1, \epsilon = 0$.

V. Experiments

We perform experiments on three problem types:

- On synthetic test problems (described below) we evaluate the robustness of the method to find a good solution in random settings, and to find a needle-in-a-haystack solutions hidden in a costly compute line of the tree.
- On Logic Geometric Programs and puzzle problems we evaluate the robustness for sequential robotic manipulation planning.
- On a Bandit problem (in appendix E) we evaluate the tree policy version of our method.

A. Synthetic Problems

We consider an infinite decision tree of fixed depth d and infinite branching at each level. Every n has the latent compute effort $\hat{c}_n \in \mathbb{R}_+$, but in addition a latent scalar $\hat{y}_n \in \mathbb{R}_+$. The return y received at terminal nodes (at depth d) is the *sum* of all $\hat{y}_n$ from root to leaf ($\hat{y}_{\text{root}} = 0$) plus noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$. In that way, although the value $\hat{y}_n$ at non-terminal nodes is never observed, they do contribute to the return at terminals and thereby imply correlations within branches. We generate the following synthetic problems:

a) Random(d): Given a fixed depth d, for every (queried) node we uniformly sample $\hat{c}_n \sim \mathcal{U}([1, 10])$ and $\hat{y}_n \sim \mathcal{U}([0, 1])$.

b) SingularLine(d,t,c): Problem parameter d fixes the depth of the infinite decision tree. Parameter t identifies the tree path which at each branching takes the t-th decision. E.g., $t = 1$ would be the first found with depth first search. All terminal nodes have return $y = 0$, except for the single terminal node of the t-line (needle in a haystack) which returns $y = 1$. Further, all nodes have equal latent compute cost $\hat{c}_n = 1$ except for *all* nodes on the t-line, which have (higher) latent compute cost $\hat{c}_n = c$.

With SingularLine and $c = 10$ we can investigate whether the method is robust to finding a solution also when hidden in a costly compute path.

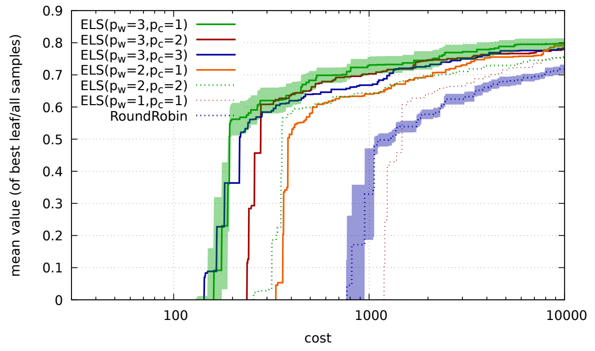

Fig. 3. Random Problem (higher is better): Best solution found in a random compute tree of depth 3 and infinite branching. The curves show averages over 10 random problems. To avoid clutter, only for best and worst also the standard deviation of the mean estimator is illustrated (others have similar deviations).

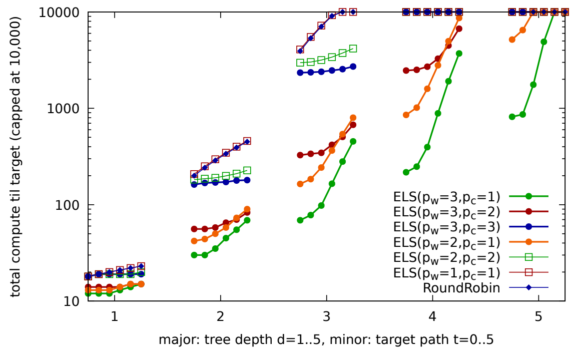

Fig. 4. Singular Line Problem (lower is better): Total compute cost until a singular target in branching path t of a tree of depth d is found. As the problem and methods are deterministic, no errorbars are given; the variation of problem parameters t and d is to indicate robustness of result.

c) Results on Random: Fig. 3 displays the best solution found, $y_{\text{best}}(C)$, in a random problem of depth 3 and infinite branching at each level over the total invested compute effort. The curves display averages over 10 runs on different random problems. We find that ELS with higher coefficients to penalize compute and branching systematically outperform RoundRobin.

d) Results on Singular Line: Fig. 4 displays total compute, C_{first}, required to find the single best terminal in the SingularLine problem, for varying depth d and target branch t. The latent compute cost for nodes along the target path is fixed to $c = 10$ (whereas all other nodes have $\hat{c}_n = 1$). ELS with higher coefficients p_w, p_c finds the singular solution significantly faster than RoundRobin.

Appendix E evaluates a tree policy version of ELS also on synthetic effort-full bandits problems.

B. Robot Manipulation Benchmarks

We tested the method on 8 different Task-and-Motion Planning problems, more specifically, on finding solutions to the Logic-Geometric Programming (LGP) [14] formulation of these TAMP problems. In this formulation, nodes on different levels of the completion tree represent different computational problems that "assemble a solution", following the previous work [15]:

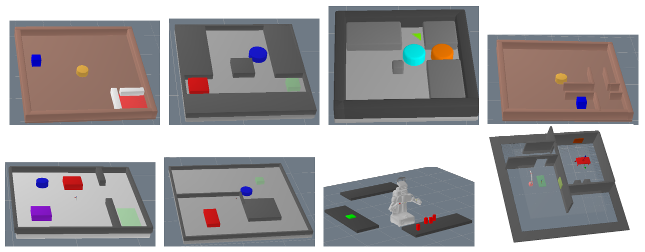

Fig. 5. 8 LGP sample problems.

- Nodes on the first level represent a particular choice of skeleton. To complete a node, an A*-solver needs to find a novel skeleton that solves the logical constraints. There are typically infinitely many skeletons that fulfil the constraints, thus the first branching is infinite.
- Nodes on the second level represent a particular random seed for waypoint solving, i.e., finding waypoints that fulfill the mode switch constraints implied by the skeleton of the parent node [15]. The random seed determines the randomized initialization of this NLP, which includes random initial placements or relative transforms that relate to random object grasps or placements. As these NLPs have many local optima, many random seeds will lead to infeasibility, and their run time varies without guarantee of termination. Only some NLP solutions will lead to feasible down-stream path finding problems. Our solver therefore needs to systematically explore alternative random seeds, leading to an infinite branching at this level.
- Nodes on the third and following levels represent path finding problems: to complete a node a RRT path finder needs to find paths to connect the waypoints. Again, the run time varies and there is no guarantee of termination.
- Nodes on the last level represent a final motion optimization NLP which takes the RRT path pieces as initialization. The completion of these nodes requires most compute effort and again, there is no guarantee of termination or robust convergence.

In this setting compute effort is identical to real time. All solvers (RRTs and NLP solvers) are implemented in a "stepping" manner, meaning that when a node is queried, only one more step (e.g., Newton step) is performed and the compute effort (real time) measured. In this rather large tree of possible computations our solver needs to decide where to invest compute effort to most efficiently find solutions.

Fig. 6 displays the time to finding a first solution for the 8 problems. The box plots show distributions over 10 random runs. Run time is capped at 120 seconds. RoundRobin consistently performs worse; for many problems a choice of $p_w = 3, p_c = 2$ or $p_w = 3, p_c = 1$ more robustly finds solutions quickly, showing that stronger penalization of width is beneficial for find TAMP solutions.

Fig. 7 displays data from the same runs differently, illustrating the number of solutions found within the limit of 120 s. RoundRobin consistently performs worse; for some of the harder problems ELS does not always find a solution within 120 s.

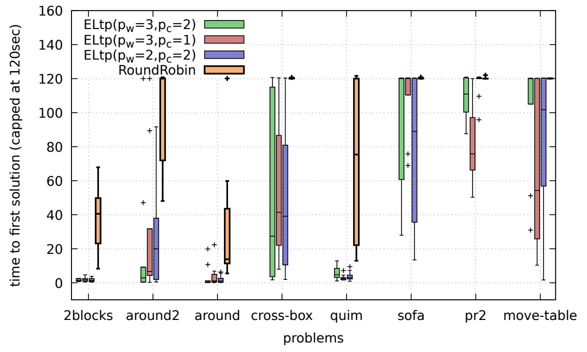

Fig. 6. LGP First Time (lower is better): Total compute needed to find the first feasible solution of various LGP problems. Boxplots show distributions for 10 runs with different random seeds (which determine the initialization of NLP solvers in the LGP). Runs are capped at 120 s.

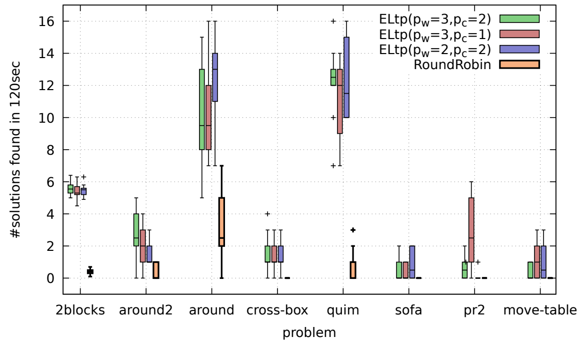

Fig. 7. LGP Solutions per Time (higher is better): The number of solutions found during 120 s, from the same runs as for Fig. 6. Note that for '2blocks', #solutions/10 is displayed.

VI. Conclusion

We introduced a novel problem formulation, completion trees, to represent the challenge of deciding where to invest compute in alternative paths to "assemble" a solution of robotic manipulation planning problems. Our algorithm, ELS, is a first step to systematically and robustly allocate compute, balancing (computation) depth with (branching) width. Even if sub-problems are NLPs prone to local optima, by introducing branching for potentially infinite choices of random seeds we gain completeness of the overall solver.

However, our proposed method mostly focuses on systematic compute allocation in the case of infinite branchings, but hardly exploits the collected data to further inform search. As the problem formulation generalizes previous bandits, budgeted bandits, and MCTS problem formulations we see great potential to combine strategies from these areas with the controlled widening of our method to better exploit the available data on both returns and invested compute.

Further, the current method is limited in that it cannot exploit anytime solvers within nodes, as there is no strict time of completion and they could return better solutions with more compute.

References

[1] Ashwinkumar Badanidiyuru, Robert Kleinberg, and Aleksandrs Slivkins. Bandits with knapsacks. *Journal of the ACM (JACM)*,

65(3):1–55, 2018.

[2] Semih Cayci, Atilla Eryilmaz, and Rayadurgam Srikant. Learning to control renewal processes with bandit feedback. *Proc. of the ACM on Measurement and Analysis of Computing Systems*, 3(2):1–32, 2019.

[3] Ariel Felner, Meir Goldenberg, Guni Sharon, Roni Stern, Tal Beja, Nathan R. Sturtevant, Jonathan Schaeffer, and Robert Holte. Partial-expansion A* with selective node generation. In *Proc. of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. AAAI Press, 2012.

[4] Caelan Garrett, Tomás Lozano-Pérez, and Leslie Kaelbling. Sample-based methods for factored task and motion planning. In *Proc. of Robotics: Science and Systems (R:SS)*, 2017.

[5] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[6] Meir Goldenberg, Ariel Felner, Roni Stern, Guni Sharon, Nathan R. Sturtevant, Robert C. Holte, and Jonathan Schaeffer. Enhanced partial expansion A*. *J. Artif. Intell. Res.*, 50:141–187, 2014.

[7] Kris Hauser. Task planning with continuous actions and nondeterministic motion planning queries. In *Proc. of AAAI Workshop on Bridging the Gap between Task and Motion Planning*, 2010.

[8] Erez Karpas, Oded Betzalel, Solomon Eyal Shimony, David Tolpin, and Ariel Felner. Rational deployment of multiple heuristics in optimal state-space search. *Artif. Intell.*, 256:181–210, 2018.

[9] Aditya Mandalika, Oren Salzman, and Siddhartha S. Srinivasa. Lazy receding horizon a* for efficient path planning in graphs with expensive-to-evaluate edges. In *International Conference on Automated Planning and Scheduling*, pages 476–484. AAAI Press, 2018.

[10] Dylan O'Ceallaigh and Wheeler Ruml. Metareasoning in real-time heuristic search. In *Proc. of the Eighth Annual Symposium on Combinatorial Search, SOCS*, pages 87–95. AAAI Press, 2015.

[11] Joaquim Ortiz-Haro, Valentin N. Hartmann, Ozgur S. Oguz, and Marc Toussaint. Learning efficient constraint graph sampling for robotic sequential manipulation. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2021.

[12] Stuart J. Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49(1-3):361–395, 1991.

[13] Shahaf S. Shperberg, Andrew Coles, Erez Karpas, Wheeler Ruml, and Solomon Eyal Shimony. Situated temporal planning using deadline-aware metareasoning. In *International Conference on Automated Planning and Scheduling*, pages 340–348. AAAI Press, 2021.

[14] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence*, 2015.

[15] Marc Toussaint and Manuel Lopes. Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017.

[16] Long Tran-Thanh, Archie Chapman, Alex Rogers, and Nicholas Jennings. Knapsack based optimal policies for budget–limited multi–armed bandits. In *Proc. of the AAAI Conference on Artificial Intelligence*, volume 26, pages 1134–1140, 2012.

[17] Yingce Xia, Tao Qin, Weidong Ma, Nenghai Yu, and Tie-Yan Liu. Budgeted Multi-Armed Bandits with Multiple Plays. In *International Joint Conference on Artificial Intelligence*, volume 6, pages 2210–2216, 2016.

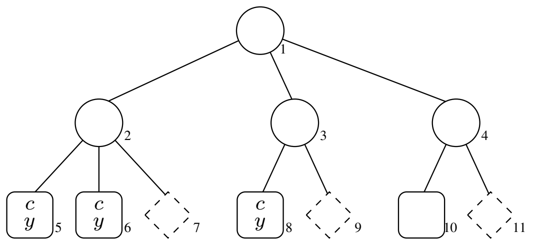

Fig. 8. Budgeted Bandits case. Special case with three bandits represented by nodes $n = 2, 3, 4$, and drawing samples y from bandits requires compute effort c with unknown distribution $p_n(c)$.

Appendix

A. Budgeted Bandits as a Special Case

Our problem setting can be used to represent basic budgeted bandits [17] and bandits-with-knapsacks [16], [1], where (in their simplest version, with only one resource) each bandit n has a latent cost distribution $p_n(c)$, and each time n is queried, one has to spend effort $c \sim p_n(c)$ to draw a sample $y \sim p_n(y)$. Fig. 8 shows a Completion Tree representation, where the finite number of bandits are direct children of the root, and drawing samples is represented as infinite branching below each bandit, each with latent compute effort $c \sim p_n(c)$. Existing budgeted bandit formulations do not consider pre-emptive scheduling, which corresponds to assuming no compute limit, $\sigma = \infty$, when investing in a new sample.

By casting budgeted bandits as a special case of our problem formulation we first see how we generalize over the previous formulation: 1) We allow for pre-emptive scheduling, 2) we consider hierarchical problems, similar to how UCT generalizes UCB1, 3) we may have infinite bandits on the base level, and 4) bandits themselves may require compute for their completion, rather than only sampling from them.

Conversely, this special case formulation makes explicit in what sense we are faced with decision making under uncertainty, and we can draw on existing budgeted bandit and UCT literature to derive decision policies also for our problem.

Concerning the objective, in bandit settings the standard definition of regret is $\sum_{t=1}^{T}[\max_a R_{a,t} - R_t]$, where a is the optimal decision in iteration t. In standard settings (including budgeted bandits) any decision in each interation t generates a sample, and thereby a return R_t, and the expectation of $\max_a R_{a,t}$ is stationary (i.e., independent of t). However, in our setting we have a finite compute budget σ and many iterations will yield no return at all as terminal nodes are not complete yet. Only when a terminal sample node becomes complete, and among them from an optimal bandit, we may yield a sample with maximal expectation $\max_a R_{a,t}$. Therefore, optimal decisions often first have to invest compute to yield no return, before maximal expected return is possible.

The regret can also be interpreted as "area above curve R_t" (clipped at R_{max} from above). We defined $y_{\text{best}}(C)$ and $y_{\text{mean}}(C)$ as functions of compute instead of interation, and can therefore transfer the concept of regret as area under curve (clipped at some y_{min} from below). Note that this equals to C_{first} if non-solutions have return 0 and a solution return 1.

B. Proof of Proposition 1

Proof: Let k be a leaf node representing an optimal solution with finite effort level $E_k < \infty$. As ELS is a level set search it will complete k before any other node with effort larger than E_k. Let N be the number of *all* nodes (in the infinite decision tree) that have effort $\leq E_k$. We need to prove that N is finite, as this implies finite total effort $\leq NE_k$ and therefore finite time.

N is largest if there are no compute costs, i.e., all nodes are completed at first query with $c_n = 0$. In this case, all nodes will have effort $E_n \geq d_n\epsilon$, with d_n the node's tree depth, and the maximal depth for a node with $E_n \leq E_k$ is $d_n = E_k/\epsilon$. On the other hand, as the widening penalty is unbounded increasing, there exists a maximal width w such that $W(w) \geq E_k$. A tree with maximal depth and width is finite, proving N is finite.

■

C. Proof of Round Robin Equivalence

Proof: We neglect compute budget discretization in the sense that we assume $\hat{c}_n$ are multiple of σ. At round $N = 0$, only the root node with $c_n = 0$ and $j_n = 1$ exists, confirming $N = c_n/\sigma + (j_n - 1)$. By induction: In any future round, the nodes inserted back into the FIFO are either yet incomplete nodes that already were in the FIFO but have incremented c_n by σ, or diamond siblings that have a widening enumeration incremented by one, or diamond children that inherit the additional compute step of the just completed parent but no additional widening step (which explains $j_k - 1$). All cases increase the quantity $\sum_{k \in \pi(n)} c_k/\sigma + \sum_{k \in \pi(n)}(j_k - 1)$ by 1, making it equal to the round number N. ■

D. Tree Policy-Based Solvers

In UCB/UCT, we get a return in every round. In our settings, most compute decisions will not complete a terminal node and therefore receive no return, which is why UCB/UCT is not directly applicable. However, the general concept of using a tree policy to decide on leaf nodes can be combined with ELS to address bandit settings. In particular, in queue-based search schemes – as our Generalized Search (Alg. 1) – all nodes in the queue have their fixed level f_n which is not influenced by data observed at other nodes, e.g., siblings or nodes in the same sub-branch. In contrast, in tree policy approaches, data collected in sub-branches is backed up to influence decisions (and thereby prioritization) for all nodes in the sub-branch, which allows for the type of learning and sharing of prioritization of whole sub-branches that underlies UCT.

To demonstrate the application of effort levels to bandit settings we therefore propose the following tree policy that applies to general completion trees with infinite branchings, and thereby extends UCT to also decide on widening rather than only descending (see Fig. 1).

Algorithm 4 Generalized Tree Policy Search

Generalized MCTS variant...

Input: root node

Output: list of terminal nodes

```
1: repeat
2:     init $n$ as root
3:     repeat $n \leftarrow \textsc{TreePolicy}(n)$ until $n$ is incomplete
4:     insert $\textsc{widen}(n)$ in queue
5:     $\textsc{compute}(n)$, $\textsc{BackupCompute}$ compute effort $c$
6:     if not $isComplete(n)$: – nothing to do –
7:     elif $isTerminal(n)$: $\textsc{BackupSample}$ sample $y$
8:     else: expand tree by the set $\textsc{deepen}(n)$
9: until time-out
```

Algorithm 5 ELS-TreePolicy

Equals Generic TreePolicy Search for specific instances of TreePolicy, BackupCompute, and BackupSample. $\pi(n)$ is the path of nodes from n to root.

```
1: function $\textsc{BackupCompute}(n, c)$
2:     $\forall k \in \pi(n) : \bar{C}_k \leftarrow \bar{C}_k + c$
3: end function

4: function $\textsc{BackupSample}(n, y)$
5:     $\forall k \in \pi(n) : \bar{Y}_k \leftarrow \bar{Y}_k + y,\ n_k \leftarrow n_k + 1$
6: end function

7: function $\textsc{TreePolicy}(n)$
8:     $k$ ←incomplete child of $n$ with lowest effort level $E(k)$
9:     if $k = \emptyset$ (no incomplete child) or $E(k) > \bar{E}(n)$:
           return complete child with highest UCB1 score
10:    return $k$
11: end function
```

First, using standard backups for both, compute efforts and sample values (see Alg. 5), we maintain at each node the sum $\bar{C}_n$ of compute invested in this node *and* all descendants, and the sum $\bar{Y}_n$ and count N_n of all samples observed at descendants. Note that, for incomplete nodes $c_n = \bar{C}_n$. We now use the total compute $\bar{C}_n$ to define the effort

$$e_n = D(\bar{C}_n) + W(j_n)\ , \tag{8}$$

For incomplete nodes this is as before; but for internal nodes this includes penalization of compute at descendants.

At a given parent node n, the tree policy has to decide on a child k, which may be an incomplete node (including diamond), or complete node without data $N_k = 0$, or complete node with data N_k. Again, the challenge is to define a single score to coherently decide among these heterogeneous choices. We propose a two stage approach: First, let

$$\hat{j} = \operatorname*{argmin}_k e_k \tag{9}$$

be the child with minimal effort (which is defined for all children, no matter if complete or with data). Further, let

$$\hat{k} = \operatorname*{argmin}_{j:N_k>0} \bar{Y}_k/N_k + \beta\sqrt{2\ln N_n/N_k} \tag{10}$$

be the UCB1 choice for complete children with data. If no child has data, or if $e_{\hat{j}} < D(\bar{C}_n)$ (the compute effort for the parent n), the tree policy decides for $\hat{j}$, otherwise the tree policy decides for $\hat{k}$.

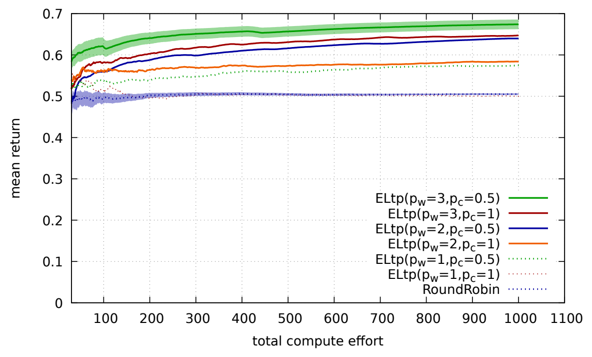

Fig. 9. Bandit Problem (higher is better): The avarage return (i.e., mean of all returns collected until compute effort c_{tot}) on an infinite branching random Bandit problem. The hypothetical best bandit (from infinite alternatives) gives returns uniform in [0.5,1]. Curves show averages from 10 runs on different random bandit problems, with standard deviations for best and worst.

E. Evaluation on Bandit Problems

We evaluate the ELS-TreePolicy on random infinite completion bandit problems, which in our framework corresponds to a two-level tree with infinite branching at both levels (first level is the choice of bandit, second level the sample). Similar to the Random problem above, bandits have a random latent compute cost uniform in $[1, 10]$ and a latent mean return $y_n \sim \mathcal{U}[\frac{1}{4}, \frac{3}{4}]$; sampling adds uniform noise $\mathcal{U}[-\frac{1}{4}, \frac{1}{4}]$ to give a return in $[0, 1]$. When the best bandit is found and completed, the maximal expected return is 0.75.

Fig. 9 displays the mean return, $y_{\text{mean}}(C)$ over the total compute investment. As expected, RoundRobin only achieves the same average return as picking a random bandit (0.5). Interestingly, ELS-TreePolicy performs best when choosing a sub-linear coefficient to penalize compute, $p_c = 0.5$.