2023 European Control Conference (ECC)
June 13-16, 2023. Bucharest, Romania

Hierarchical Finite State Machines for Efficient Optimal Planning in Large-scale Systems

Elis Stefansson[1] and Karl H. Johansson[1]

Abstract— **In this paper, we consider a planning problem for a hierarchical finite state machine (HFSM) and develop an algorithm for efficiently computing optimal plans between any two states. The algorithm consists of an offline and an online step. In the offline step, one computes exit costs for each machine in the HFSM. It needs to be done only once for a given HFSM, and it is shown to have time complexity scaling linearly with the number of machines in the HFSM. In the online step, one computes an optimal plan from an initial state to a goal state, by first reducing the HFSM (using the exit costs), computing an optimal trajectory for the reduced HFSM, and then expand this trajectory to an optimal plan for the original HFSM. The time complexity is near-linearly with the depth of the HFSM. It is argued that HFSMs arise naturally for large-scale control systems, exemplified by an application where a robot moves between houses to complete tasks. We compare our algorithm with Dijkstra's algorithm on HFSMs consisting of up to 2 million states, where our algorithm outperforms the latter, being several orders of magnitude faster.**

I. Introduction

A. Motivation

Large-scale control systems are becoming ubiquitous as we move towards smarter and more connected societies. Therefore, analysing and optimising the performance of such systems is of outmost importance. One common approach to facilitate the analysis of a large-scale system is to break up the system into subsystems, analyse these subsystems separately, and then infer the performance of the whole system. This ideally ease the analysis and enables reconfiguration (e.g., if one subsystem is changed, the whole system does not need to be reanalysed).

One framework suited for modelling systems made of subsystems is the notion of a hierarchical finite state machine (HFSM). Originally introduced by Hashel [8], an HFSM is a machine composed of several finite state machines (FSMs) nested into a hierarchy. The motivation is to conveniently model complex systems in a modular fashion being able to represent and depict subsystems and their interaction neatly.

In this paper, we are interested in how to optimally plan in HFSMs. As an example, consider a robot moving between warehouses as in Fig. 1. In each warehouse, there are several locations the robot can go to, and at each location the robot can do certain tasks (e.g., scan a test tube), with decision costs given by some cost functional. This system can naturally be modelled as an HFSM with a hierarchy consisting of three layers (warehouse, location and task layer). A key question is then how to design efficient planning algorithms for such HFSMs that take into account the hierarchical structure of the system, seen as a first step towards more efficient planning algorithms for large-scale systems in general.

[1]School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Sweden. Email: {elisst,kallej}@kth.se. The authors are also affiliated with Digital Futures.

This work was partially funded by the Swedish Foundation for Strategic Research, the Swedish Research Council, and the Knut och Alice Wallenberg foundation.

B. Contribution

We consider optimal planning in systems modelled as HFSMs and present an efficient algorithm for computing an optimal plan between any two states in the HFSM. More precisely, our contributions are three-fold:

Firstly, we extend the HFSM formalism in [3] to the case when machines in the hierarchy have costs, formalised by Mealy machines (MMs) [12], and call the resulting hierarchical machine a hierarchical Mealy machine (HiMM).

Secondly, we present an algorithm for efficiently computing optimal plans between any two states in an HiMM. The algorithm consists of an offline step and an online step. In the offline step, one computes exit costs for each MM in the HiMM. It needs to be done only once for a given HiMM, and it is shown to have time complexity scaling linearly with the number of machines in the HiMM, able to handle large systems. In the online step, one computes an optimal plan from an initial state to a goal state, by first reducing the HiMM (using the exit costs), computing an optimal trajectory to the reduced HiMM, and then expand this trajectory to an optimal plan for the original HiMM. The partition into an offline and online step enables rapid computations of optimal plans by the online step. Indeed, it is shown that the online step obtains an optimal trajectory to the reduced HiMM in time $O(\mathrm{depth}(Z)\log(\mathrm{depth}(Z)))$, where $\mathrm{depth}(Z)$ is the depth of the hierarchy of the considered HiMM Z, and can then use this trajectory to retrieve the next optimal input of the original HiMM in time $O(\mathrm{depth}(Z))$, or obtain the full optimal plan u at once in time $O(\mathrm{depth}(Z)|u|)$, where $|u|$ is the length of u. This should be compared with Dijkstra's algorithm which could be more than exponential in $\mathrm{depth}(Z)$ [6], [7].

Thirdly, we show-case our algorithm on the robot application introduced in the motivation and validate it on large hierarchical systems consisting of up to 2 million states, comparing our algorithm with Dijkstra's algorithm. Our algorithm outperforms the latter, where the partition into an offline and online step reduces the overall computing time, and the online step computes optimal plans in just milliseconds compared to tens of seconds using Dijkstra's algorithm.

978-3-907144-08-4 ©2023 EUCA

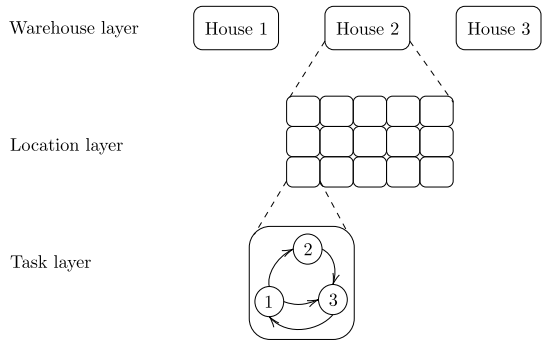

Fig. 1: A mobile robot moving between warehouses modelled as a three-layer HFSM.

C. Related Work

Traditionally, HFSMs has been used to model reactive agents, such as wrist-watches [8], rescue robots [15] and non-player characters in games [13]. Here, the response of the agent (the control law) is represented as an HFSM reacting to inputs from the environment (e.g., "hungry"), where subsystems typically correspond to subtasks (e.g, "get food"). This paper differs from this line of work by instead treating the environment as an HFSM where the agent can *choose* the inputs fed into the HFSM (i.e., inputs are now decision variables), with aim to steer the system to a desirable state. In discrete event systems [4], a variant of HFSMs known as state tree structures has been used to compute safe executions [10], [18]. We differ in the HFSM formalism and focus instead on optimal planning with respect to a cost functional.

There is an extensive literature when it comes to path planning in discrete systems [2], [9]. Hierarchical path planning algorithms, e.g., [5], [11], [14], are the ones most reminiscent to our approach due to their hierarchical structure. Related algorithms consider path planning on weighted graphs, pre-arranging the graph into clusters to speed up the search, and could be used to plan FSMs. However, to apply such methods for an HFSM, we would first need to flatten the HFSM to an equivalent flat FSM, making the algorithm agnostic to the modular structure of the HFSM (and thus less suitable for instance reconfigurations in the HFSM), and could also in the worst case (when reusing identical components of the HFSM, see [1], [19] for details) cause an exponential time complexity. It is therefore beneficial to instead consider path planning in the HFSM directly. This is done in this work. The work [17] seeks an execution of minimal length between two configurations in a variant of an HFSM. This paper differ in the HFSM formalism and consider non-negative transition costs instead of just minimal length.

Finally, the work [3] formalises HFSMs without outputs, and uses a modular decomposition to decompose an FSM into an equivalent HFSM. Planning is not considered. In this work, we extend the HFSM formalism from [3] to HFSMs with outputs (modelling costs) and consider optimal planning.

D. Outline

The outline of the paper is as follows. Section II formally defines HiMMs and formulates the problem statement. Section III presents our planning algorithm. Section IV validate our algorithm in numerical evaluations. Finally, Section V concludes the paper. An extended version of this paper can be found at [16] that contains all the proofs in Appendix.

II. Problem Formulation

A. Hierarchical Finite State Machines

We follow the formalism of [3] closely when defining our hierarchical machines, extending their setup to the case when machines also have outputs. Formally, we consider Mealy machines [12] and then define hierarchical Mealy machines.

Definition 1 (Mealy Machine): An MM is a tuple $M = (Q, \Sigma, \Lambda, \delta, \gamma, s)$, where Q is a finite set of states; Σ is a finite set of inputs, the input alphabet; Λ is a finite set of outputs, the output alphabet; $\delta : Q \times \Sigma \rightharpoonup Q$ is the transition function, which can be a partial function[1]; $\gamma : Q \times \Sigma \to \Lambda$ is the output function; and $s \in Q$ is the start state.

An MM M works as follows. When initialised, M starts in the start state s. Next, given a current state $q \in Q$ and input $x \in \Sigma$, M outputs $\gamma(q, x) \in \Lambda$, and transits to the state $\delta(q, x)$ if $\delta(q, x) \neq \emptyset$ (i.e., if $\delta(q, x)$ is defined), otherwise M stops. Repeating this process results in a trajectory of M:

Definition 2 (Trajectory): A sequence $z = \{(q_i, x_i)\}_{i=1}^{N}$ with $N \in \mathbb{Z}^+$ is a *trajectory* of an MM $M = (Q, \Sigma, \Lambda, \delta, \gamma, s)$ (starting at $q_1 \in Q$) if $q_{i+1} = \delta(q_i, x_i) \neq \emptyset$ for $i \in \{1, \dots, N-1\}$.

In this work, we assume that we can *choose* the inputs. In such settings, we also talk about plans and their corresponding induced trajectories. More precisely, a *plan* is a sequence $u = (x_i)_{i=1}^{N} \in \Sigma^N$ with $N \in \mathbb{Z}^+$, and we call $z = \{(q_i, x_i)\}_{i=1}^{N}$ the *induced trajectory* to u starting at $q_1 \in Q$ if z is a trajectory. Finally, we sometimes use the notation $Q(M)$, $\Sigma(M)$, $\Lambda(M)$, δ_M, γ_M and $s(M)$ to stress that e.g., $Q(M)$ is the set of states of M.

Remark 1: Here, δ is a partial function to model stops in the machine. In the hierarchical setup, this allows higher-layer machines in the hierarchy to be called when lower-layer machines are completed. See [3] for a detailed account.

Definition 3 (Hierarchical Mealy Machine): An HiMM is a pair $Z = (X, T)$, where X is a set of MMs with input set Σ and output set Λ (the MMs in Z), and T is a tree with the MMs in X as nodes (specifying how the MMs in X are composed in Z). More precisely, each node $M \in X$ in T has $|Q(M)|$ labelled outgoing arcs $\{M \xrightarrow{q} M_q\}_{q \in Q(M)}$, where either $M_q \in X$ (meaning that state q of M corresponds to the MM M_q one layer below in the hierarchy of Z) or $M_q = \emptyset$ (meaning that q is just a state without refinement). For brevity, call $Q_Z := \cup_{M \in X} Q(M)$ the nodes of Z and $S_Z := Q_Z \cap \{q : M_q = \emptyset\}$ the states of Z. The depth of Z, $\mathrm{depth}(Z)$, is the depth of the tree T, i.e., the maximum

[1]We use the notation $f : A \rightharpoonup B$ to denote a partial function from a set A to a set B (i.e., a function that is only defined on a subset of A). If $f(a)$ with $a \in A$ is not defined, then we write $f(a) = \emptyset$.

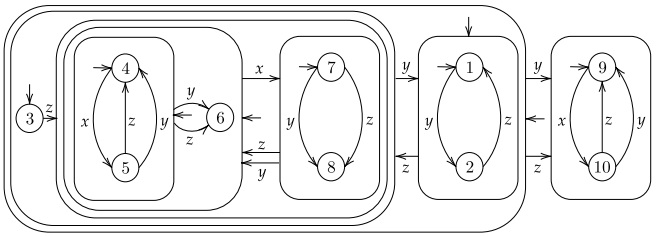

Fig. 2: Example of an HiMM taken from [3] but with added outputs, assumed to be unit costs.

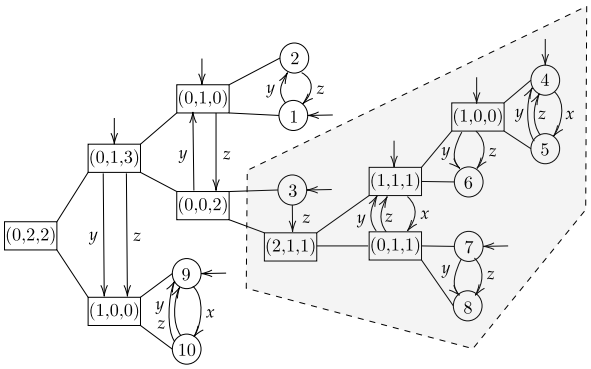

Fig. 3: Tree representation of the HiMM in Fig. 2. The tuple on an MM M is the optimal exit costs (c_x^M, c_y^M, c_z^M) obtained by the offline step. The light-grey region depicts the part that is removed in the online step with $s_{\text{init}} = 2$ and $s_{\text{goal}} = 10$.

over all directed path lengths in T. Furthermore, we also have notions of the start state, the transition function and the output function (where $(X_i \xrightarrow{v} X_j) \in T$ means that there is an arc labelled v from X_i to X_j in T):

(i) Start function: The function $\text{start} : X \to S_Z$ is

$$\text{start}(X_i) = \begin{cases} \text{start}(X_j), & (X_i \xrightarrow{s(X_i)} X_j) \in T, X_j \in X \\ s(X_i), & \text{otherwise.} \end{cases}$$

(ii) Hierarchical transition function: Let $q \in Q(X_j)$, where $X_j \in X$, and $v = \delta(q, x)$. Then the hierarchical transition function $\psi : Q_Z \times \Sigma \rightharpoonup S_Z$ is defined as

$$\psi(q, x) = \begin{cases} \text{start}(Y), & v \neq \emptyset, (X_j \xrightarrow{v} Y) \in T, Y \in X \\ v, & v \neq \emptyset, \text{otherwise} \\ \psi(w, x), & v = \emptyset, (W \xrightarrow{w} X_j) \in T, W \in X \\ \emptyset, & v = \emptyset, \text{otherwise.} \end{cases}$$

(iii) Hierarchical output function: Let $q \in Q(X_j)$ where $X_j \in X$ and $v = \delta(q, x)$. Then the hierarchical output function $\chi : Q_Z \times \Sigma \rightharpoonup \Lambda$ of Z is defined as

$$\chi(q, x) = \begin{cases} \gamma_{X_j}(q, x), & v \neq \emptyset, (X_j \xrightarrow{v} Y) \in T, Y \in X \\ \gamma_{X_j}(q, x), & v \neq \emptyset, \text{otherwise} \\ \chi(w, x), & v = \emptyset, (W \xrightarrow{w} X_j) \in T, W \in X \\ \emptyset, & v = \emptyset, \text{otherwise.} \end{cases}$$

An HiMM $Z = (X, T)$ works analogously to an MM. When initialised, the HiMM starts at state $\text{start}(M_0) \in S_Z$ (where M_0 is the root of T). Next, given a current state $q \in S_Z$ and input $x \in \Sigma$, Z outputs $\chi(q, x) \in \Lambda$, and transits to the state $\psi(q, x) \in S_Z$ if $\psi(q, x) \neq \emptyset$, otherwise Z stops. Furthermore, a trajectory, plan, and induced trajectory are (with obvious modifications) defined totally analogously as for an MM.

Remark 2 (Intuition): For intuition regarding Definition 3, consider the HiMM $Z = (X, T)$ in Fig. 2. The HiMM is from [3] but with added outputs, for simplicity assumed to be all unit costs, hence, we omit writing them. Here, the states of Z are the circles labelled 1 to 10, while the nodes of Z are the circles plus the (unlabelled) rounded rectangles, which we denote by the states they contain, e.g., $\{9, 10\}$. Furthermore, the inputs are $\Sigma = \{x, y, z\}$, where labelled arrows denote corresponding transitions, and small arrows specify start states. For example, in the MM having nodes $\{4, 5\}$ and 6 as states, one starts in $\{4, 5\}$ and can e.g., transition from $\{4, 5\}$ to 6 with input y. To get intuition concerning the hierarchical transition function ψ, consider the case when Z is in state 2 and apply input y. If there would have been a y-transition from 2, then we would have just moved according to that transition. However, this is not the case and hence, we instead move up iteratively in the hierarchy until we find a node that has a y-transition (or stop Z if we don't find one). In this case, the node $\{1, 2\}$ (just above 2 in the hierarchy) does not support a y-transition either, but the node $\{1, \ldots, 8\}$ (above $\{1, 2\}$) does and we therefore move according to that transition, i.e., to $\{9, 10\}$. Once moved, we iteratively follow the start states down in hierarchy until we arrive at a state of Z, in this case from $\{9, 10\}$ to 9. With this step, the procedure is complete, that is, $\psi(2, y) = 9$. Moreover, the output $\chi(2, y)$ captures the transition corresponding to the y-transition going from $\{1, \ldots, 8\}$ with y, i.e., $\chi(2, y) = \gamma_M(\{1, \ldots, 8\}, y)$, where M is the MM in Z having the node $\{1, \ldots, 8\}$ as state. The motivation is to support modularity neatly, where we in our case care about that we moved in M from $\{1, \ldots, 8\}$ with y, but not how we arrived at $\{1, \ldots, 8\}$ (with y) from layers below. Future work will consider variations of this setup. Finally, we also depict the HiMM in a tree-like structure given by Fig. 3, useful when illustrating the planning algorithm given in the next section.

In this paper, we exclusively consider HiMMs such that $\Lambda \subset \mathbb{R}^+$ (henceforth implicitly assumed), interpreting $\chi(q, x)$ as the *cost* for executing x at node q. For such an HiMM Z, we also define a cumulative cost. More precisely, for a trajectory $z = \{(q_i, x_i)\}_{i=1}^N$ of Z, the cumulative cost is $C(z) = \sum_{i=1}^N \chi(q_i, x_i)$ if all $\chi(q_i, x_i) \neq \emptyset$, and $C(z) = +\infty$ otherwise.[2] We put $C(z) = +\infty$ in the latter case since we do not want Z to stop.

B. Problem Statement

We now formalise the problem statement. Let $Z = (X, T)$ be an HiMM such that $\Lambda \subset \mathbb{R}^+$. Consider state $s_{\text{init}} \in S_Z$, called the *initial state*, and $s_{\text{goal}} \in S_Z$ called the *goal state*

[2] In fact, since z is a trajectory, only $\chi(q_N, x_N)$ may be empty.

Algorithm 1 Hierarchical planning

Input: HiMM $Z = (X, T)$ and states $s_{\text{init}}, s_{\text{goal}}$.
Output: Optimal plan u to $(Z, s_{\text{init}}, s_{\text{goal}})$.
1: **Offline step:**
2: $(c_x^M, z_x^M)_{x\in\Sigma, M\in X} \leftarrow \text{Offline\_step}(Z)$
3: **Online step:**
4: $z \leftarrow \text{Reduce\_and\_solve}(Z, s_{\text{init}}, s_{\text{goal}}, (c_x^M, z_x^M)_{x\in\Sigma, M\in X})$
5: $u \leftarrow \text{Expand}(z, (z_x^M)_{x\in\Sigma, M\in X}, Z)$

Algorithm 2 Offline_step

Input: HiMM $Z = (X, T)$ with markings.
Output: Computed $(c_x^M, z_x^M)_{x\in\Sigma}$ for each MM M of Z
1: $\text{Optimal\_exit}(M_0)$ ▷ Run from root MM M_0 of T
2: $\text{Optimal\_exit}(M)$: ▷ Recursive help function
3: **for** each state q in $Q(M)$ **do**
4: **if** $q \in S_Z$ **then**
5: $(c_x^q)_{x\in\Sigma} \leftarrow 0_{|\Sigma|}$
6: **else**
7: Let M_q be the MM corresponding to q.
8: $(c_x^q, z_x^q)_{x\in\Sigma} \leftarrow \text{Optimal\_exit}(M_q)$
9: **end if**
10: **end for**
11: Construct $\hat{M}$
12: $(c_x^M, z_x^M)_{x\in\Sigma} \leftarrow \text{Dijkstra}(s(M), \{E_x\}_{x\in\Sigma}, \hat{M})$
13: return $(c_x^M, z_x^M)_{x\in\Sigma}$

(any states of Z). Let $\Psi(s_{\text{init}}, s_{\text{goal}})$ be the set of plans $u = (x_i)_{i=1}^N$ that ends at s_{goal} if starting at s_{init}, i.e., the induced trajectory $z = \{(q_i, x_i)\}_{i=1}^N$ to u starting at $q_1 = s_{\text{init}}$ exists (i.e., z is a trajectory) and $\psi(q_N, x_N) = s_{\text{goal}}$. Then, find a plan $\hat{u} \in \Psi(s_{\text{init}}, s_{\text{goal}})$ that minimises the cumulative cost

$$\min_{u\in\Psi(s_{\text{init}}, s_{\text{goal}})} C(z),$$

where z is the induced trajectory to u starting at s_{init}. We call such a $\hat{u}$ an *optimal plan* to the *planning objective* $(Z, s_{\text{init}}, s_{\text{goal}})$, and the induced trajectory $\hat{z}$ an *optimal trajectory* to $(Z, s_{\text{init}}, s_{\text{goal}})$. Moreover, a plan $u \in \Psi(s_{\text{init}}, s_{\text{goal}})$ is called a *feasible plan* to $(Z, s_{\text{init}}, s_{\text{goal}})$, and we say that $(Z, s_{\text{init}}, s_{\text{goal}})$ is *feasible* if there exists a feasible plan.

III. Hierarchical planning

In this section, we present our hierarchical planning algorithm, providing an overview in Section III-A, followed by details in Sections III-B to III-D. We fix an arbitrary planning objective $(Z, s_{\text{init}}, s_{\text{goal}})$ throughout this section.

A. Overview

The hierarchical planning algorithm finds an optimal plan u to $(Z, s_{\text{init}}, s_{\text{goal}})$. The algorithm is summarised by Algorithm 1 and consists of an offline step and an online step. The offline step computes optimal exit costs $(c_x^M)_{x\in\Sigma}$ and corresponding trajectories $(z_x^M)_{x\in\Sigma}$ for each MM $M \in X$ (line 2). This step needs to be done only once for a given HiMM Z. The online step then computes an optimal plan u to $(Z, s_{\text{init}}, s_{\text{goal}})$ using the result from the offline step. More precisely, it first reduces Z to an equivalent reduced HiMM $\bar{Z}$ (equivalence given by Theorem 1), pruning all irrelevant MMs of Z and replacing them with corresponding costs from the offline step, and then finds an optimal trajectory z to $\bar{Z}$ (line 4). Then, it expands z to an optimal trajectory for $(Z, s_{\text{init}}, s_{\text{goal}})$ from which we get the optimal plan u (line 5). The details of the offline and online step is given by Section III-B and Sections III-C to III-D, respectively, where lines 2, 4 and 5 in Algorithm 1 correspond to Algorithm 2, 4 and 5, respectively.

B. Offline Step

Towards a precise formulation of the offline step, we need the following notions. A state $q \in S_Z$ is *contained* in an MM $M \in X$ if q is a descendant of M in T (e.g., state 4 is contained in the MM labelled (2,1,1) in Fig. 3). A trajectory $z = \{(q_i, x_i)\}_{i=1}^N$ is an (M, x)-*exit trajectory* if every q_i is contained in M, $x_N = x$, and $q_{N+1} := \psi(q_N, x_N)$ (possibly empty) is not contained in M (hence, z exits M with x). The corresponding (M, x)-*exit cost* of z equals $\sum_{i=1}^{N-1} \chi(q_i, x_i)$ (the cost of (q_N, x_N) is excluded since the transition goes outside the subtree with root M). The *optimal* (M, x)-*exit cost*, denoted c_x^M, is the minimal (M, x)-exit cost,[3] and any such c_x^M is called an *optimal exit cost* of M. An (M, x)-exit trajectory that achieves the optimal (M, x)-exit cost is an *optimal* (M, x)-*exit trajectory*, and any such trajectory is called an *optimal exit trajectory* of M.

We now provide the details of the offline step. The offline step obtains $(c_x^M)_{x\in\Sigma}$ for each MM M in $Z = (X, T)$ recursively over the tree T. More precisely, let $M = (Q, \Sigma, \Lambda, \delta, \gamma, s)$ be an MM of Z. To compute $(c_x^M)_{x\in\Sigma}$, form the augmented MM $\hat{M}$ given by $\hat{M} = (Q \cup \{E_x\}_{x\in\Sigma}, \Sigma, \Lambda, \hat{\delta}, \hat{\gamma}, s)$. Here, $\hat{M}$ is identical to M except that whenever an input $x \in \Sigma$ would exit M ($\delta(q, x) = \emptyset$) then one instead goes to the added state E_x in $\hat{M}$. That is, $\hat{\delta}(q, x) := \delta(q, x)$ if $\delta(q, x) \neq \emptyset$ and $\hat{\delta}(q, x) = E_x$ otherwise (the values of $\hat{\delta}(q, x)$ for $q \in \{E_x\}_{x\in\Sigma}$ are immaterial). Furthermore, $\hat{\gamma}$ is given by $\hat{\gamma}(q, x) := c_x^q + \gamma(q, x)$ if $\delta(q, x) \neq \emptyset$ and $\hat{\gamma}(q, x) := c_x^q$ otherwise (again, the values of $\hat{\gamma}(q, x)$ for $q \in \{E_x\}_{x\in\Sigma}$ are immaterial). Here, $c_x^q = 0$ if $q \in S_Z$ is a state of Z, and $c_x^q = c_x^{M_q}$ otherwise, where M_q is the MM corresponding to q. Thus, intuitively, $\hat{\gamma}(q, x)$ reflects the cost of exiting q with x plus the cost of applying x from q in M. Note also that, by recursion (going upwards in the tree T), we may assume that all c_x^q are already known.

We can now obtain $(c_x^M)_{x\in\Sigma}$ by doing a shortest path search in $\hat{M}$ starting from s. We use Dijkstra's algorithm [6], [7] computing a shortest-path tree from s until we have reached all E_x, and thereby obtained $(c_x^M)_{x\in\Sigma}$ (with $c_x^M = \infty$ if we never reach E_x). We also save the corresponding trajectories $(z_x^M)_{x\in\Sigma}$ in $\hat{M}$ that we obtain for free from Dijkstra's algorithm. This procedure is performed recursively

[3] If no (M, x)-exit trajectories exists, then $c_x^M = \infty$ and any trajectory contained in M is said to be an optimal (M, x)-exit trajectory.

over the whole tree T to obtain $(c_x^M)_{x\in\Sigma}$ and $(z_x^M)_{x\in\Sigma}$ for all MMs M of Z. The algorithm is given by Algorithm 2, with correctness given by Proposition 1 and time complexity given by Proposition 2. See Fig. 3 for an example concerning the optimal exit costs.

Proposition 1: $c_x^M \in [0,\infty]$ computed by Algorithm 2 equals the optimal (M,x)-exit cost of Z.

Proposition 2: The time complexity of Algorithm 2 is

$$O(N[b_s|\Sigma| + (b_s+|\Sigma|)\log(b_s+|\Sigma|)]),$$

where b_s is the maximum number of states in an MM of Z, and N is the number of MMs in Z.

Remark 3: We stress that all the time complexity results in this section (Section III) are based on using Dijkstra's algorithm with a Fibonacci heap, due to the low time complexity, see [7] for details. However, for the systems in the simulations in Section IV, we use Dijkstra's algorithm with an ordinary priority queue (that has a slightly higher time complexity) since it is in practice faster for those systems.

Finally, in the remaining sections, for brevity, let a (q,x)-exit trajectory (cost) mean an (M,x)-exit trajectory (cost) if q corresponds to the MM M. If q does not correspond to an MM, i.e., q is a state of Z, then let the (q,x)-exit trajectory and cost be simply (q,x) and zero cost, with intuition that one can then only exit q with x by applying x directly. An optimal (q,x)-exit trajectory is defined analogously and the corresponding optimal (q,x)-exit cost c_x^q is as above (readily obtained from the optimal exits costs from Algorithm 2).

C. Online Step: Theory

We continue with the online step providing necessary theory in this section, while the algorithm is presented in Section III-D. To this end, let $U_1,\dots,U_n$ be the path of MMs in T from s_{init} to the root of T, that is, s_{init} is a state of the MM U_1, the corresponding node of U_i is a state of the MM U_{i+1}, and U_n equals the root MM of T. Similarly, let $D_1,\dots,D_m$ be the path of MMs of X in T from s_{goal} to the root of T (with s_{goal} being a state of the MM D_1). Note that there exist indices α and β such that corresponding nodes of U_α and D_β are states in the same MM of X. For brevity, let $B := D_\beta$. To get an optimal plan, we consider only the MMs of these two paths, see Fig. 3 for an example, formalised by the *reduced HiMM* in Definition 5 with equivalence to the original HiMM given by Theorem 1. For this result, we need the notion of a reduced trajectory and an optimal expansion. These two notions can be seen as complementary operations: the first reduces a trajectory to a subset of T, where the latter can be used to expand a reduced trajectory with respect to a subset of T to the whole tree.

Definition 4 (Reduced trajectory): Let $Z=(X,T)$ be an HiMM and consider a connected subset $\mathcal{M}$ of tree-nodes of T that includes the root of T. Let $z=(q_i,x_i)_{i=1}^N$ be a trajectory and define the reduced trajectory $z|_{\mathcal{M}}$ of z with respect to $\mathcal{M}$ as follows. For any state $q\in S_Z$ in some MM $M\in X$, let $\bar{q}$ be the reduced node of q with respect to $\mathcal{M}$; that is, $\bar{q}\in Q(\bar{M})$ contains q, where $\bar{M}$ is the MM in $\mathcal{M}$ with minimal path length to M in T. Define $\bar{z}_i=(\bar{q}_i,x_i)$ if $\bar{q}_{i+1}\neq\bar{q}_i$ and $\bar{z}_i=\emptyset$ otherwise. Then $z|_{\mathcal{M}}$ is the sequence of nonempty $\bar{z}_i$.

Algorithm 3 Optimal_expansion

Input: HiMM $Z=(X,T)$ and node-input pair (q,x).
Output: An optimal expansion z

```
return z ← Trajectory_expansion(q, x)
Trajectory_expansion(q, x):          ▷ Recursive function
if q ∈ S_Z then
    return (q, x).
else
    (q_1, x_1), ..., (q_m, x_m) ← z_x^{M_q}    ▷ z_x^{M_q} from offline step
    for each (q_i, x_i) do
        z_i ← Trajectory_expansion(q_i, x_i)
    end for
    return (z_1, ..., z_m)
end if
```

In words, $z|_{\mathcal{M}}$ equals all visible transitions seen in $\mathcal{M}$, where $\bar{q}$ gives us the best information of the state q with respect to $\mathcal{M}$, and $z|_{\mathcal{M}}$ changes whenever this $\bar{q}$ changes.

Next, an *optimal expansion* of a node-input pair $(q,x)\in Q_Z\times\Sigma$ of Z is defined by Algorithm 3, denote it by $E(q,x)$. An optimal expansion of a sequence $z=(q_i,x_i)_{i=1}^N$ of node-input pairs is on the form $(E(q_1,x_1),\dots,E(q_N,x_N))$. The following result justifies the notion:

Proposition 3: Let $Z=(X,T)$ be an HiMM and $(q,x)\in Q_Z\times\Sigma$. Then $E(q,x)$ is an optimal (q,x)-exit trajectory.

We now define the reduced HiMM $\bar{Z}$.

Definition 5 (Reduced HiMM): Let $Z=(X,T)$, s_{init} and s_{goal} be as above. Consider the HiMM $\bar{Z}=(\bar{X},\bar{T})$ where $\bar{X}=\{\bar{U}_1,\dots,\bar{U}_n\}\cup\{\bar{D}_1,\dots,\bar{D}_n\}$ and $\bar{T}$ is equal to the subtree of T consisting of the nodes $U_1,\dots,U_n$ and $D_1,\dots,D_m$ but replaced with $\bar{U}_1,\dots,\bar{U}_n$ and $\bar{D}_1,\dots,\bar{D}_n$ respectively. For brevity, let $\mathcal{M}=\{U_1,\dots,U_n\}\cup\{D_1,\dots,D_n\}$. The details of this construction are:

(i) Let $U_i=(Q,\Sigma,\Lambda,\delta,\gamma,s)$. Then $\bar{U}_i=(Q,\Sigma,\Lambda,\delta,\bar{\gamma},s)$, where $\bar{\gamma}(q,x)=$

$$\begin{cases}\gamma(q,x), & \delta(q,x)\neq\emptyset, (U_i\xrightarrow{q}M)\in T, M\in\mathcal{M}\\ c_x^q+\gamma(q,x), & \delta(q,x)\neq\emptyset, \text{otherwise}\\ 0, & \delta(q,x)=\emptyset, (U_i\xrightarrow{q}M)\in T, M\in\mathcal{M}\\ c_x^q, & \delta(q,x)=\emptyset, \text{otherwise.}\end{cases} \tag{1}$$

Here, c_x^q is as in Section III-B. The MM $\bar{D}_i$ is defined analogously (replace U_i and $\bar{U}_i$ with D_i and $\bar{D}_i$, respectively).

(ii) The reduced hierarchical transition function $\bar{\psi}$ is constructed according to Definition 3 for the HiMM $\bar{Z}$.

(iii) Let q be a node in $\bar{Z}=(\bar{X},\bar{T})$ where $q\in Q(X_j)$, $X_j\in\bar{X}$. Let $v=\delta(q,x)$. The reduced hierarchical output function $\bar{\chi}$ is defined as $\bar{\chi}(q,x)=\gamma_{X_j}(q,x)$ if $v\neq\emptyset$; $\bar{\chi}(q,x)=\gamma_{X_j}(q,x)+\bar{\chi}(w,x)$ if $v=\emptyset$ and $(W\xrightarrow{w}X_j)\in\bar{T}$ with $W\in\bar{X}$; and, $\bar{\chi}(q,x)=\emptyset$ otherwise.

We call $\bar{Z}$ the *reduced HiMM* (with respect to s_{init} and s_{goal}). Analogous to HiMMs, the *reduced cumulative cost* $\bar{C}(z)$ for a trajectory z of $\bar{Z}$ is $\bar{C}(z) = \sum_{i=1}^{N} \bar{\chi}(q_i, x_i)$ (if $\bar{\chi}(q_N, x_N) \neq \emptyset$, and $C(z) = +\infty$ otherwise), a plan that minimises $\min_{u \in \Psi(s_{\text{init}}, s_{\text{goal}})} \bar{C}(z)$ is an *optimal plan* and the induced trajectory is an *optimal trajectory* to the *reduced planning objective* $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$. We have the following key result:

Theorem 1 (Planning equivalence): Let $Z = (X, T)$ be an HiMM and $\mathcal{M} = \{U_1, \ldots, U_n\} \cup \{D_1, \ldots, D_m\}$. Then:

(i) Let z be an optimal trajectory to $(Z, s_{\text{init}}, s_{\text{goal}})$. Then, the reduced trajectory $z|_{\mathcal{M}}$ is an optimal trajectory to $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$.

(ii) Let z be an optimal trajectory to $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$. Then, an optimal expansion of z (expanded over Z) is an optimal trajectory to $(Z, s_{\text{init}}, s_{\text{goal}})$.

Theorem 1 (ii) says that, to look for an optimal trajectory to $(Z, s_{\text{init}}, s_{\text{goal}})$, one can look for an optimal trajectory to $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$ instead and then just expand it. This result is crucial for our planning algorithm, since it drastically reduces the search space. Also, (i) says that if there is no feasible trajectory to $(Z, s_{\text{init}}, s_{\text{goal}})$, then there is no feasible trajectory to $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$ either. Thus, we can exclusively consider the reduced planning objective $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$. For the planning algorithm, we also need the following result:

Proposition 4: Consider $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$. Then:

(i) An optimal trajectory $z = (q_i, x_i)_{i=1}^{N}$ to $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$ has a state q_i equal to $\text{start}(\bar{B})$.

(ii) Let $\text{start}(\bar{B})$ be a state of $\bar{D}_{k_1}$ and $\text{start}(\bar{D}_{k_1-1})$ be the start state of $\bar{D}_{k_2}$, and so on till $\text{start}(\bar{D}_{k_j})$ is the start state of $\bar{D}_1$. Then, provided $(\bar{Z}, \text{start}(\bar{B}), s_{goal})$ is feasible[4], an optimal trajectory to $(\bar{Z}, \text{start}(\bar{B}), s_{goal})$ is $w_1, w_2, \ldots, w_j$, where w_i is an optimal trajectory in $\bar{D}_{k_i}$ from the start state of $\bar{D}_{k_i}$ to the state corresponding to $\bar{D}_{k_i-1}$ (with $\bar{D}_0 = s_{goal}$).

D. Online Step: Algorithm

In this section, we provide the details of the online step, based on the theory in Section III-C. More precisely, to compute an optimal plan to $(Z, s_{\text{init}}, s_{\text{goal}})$, the algorithm first considers the reduced planning objective $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$, which by Proposition 4 (i) can be divided into obtaining an optimal trajectory to $(\bar{Z}, s_{\text{init}}, \text{start}(\bar{B}))$ and then combining it with an optimal trajectory to $(\bar{Z}, \text{start}(\bar{B}), s_{\text{goal}})$. An optimal trajectory to $(Z, s_{\text{init}}, s_{\text{goal}})$ is then obtained by expanding this trajectory, as given by Theorem 1 (ii). From this, we obtain an optimal plan to $(Z, s_{\text{init}}, s_{\text{goal}})$. The details are given below.

1) Solving $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$: To solve $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$, with procedure given by Algorithm 4, we first solve $(\bar{Z}, s_{\text{init}}, \text{start}(\bar{B}))$. To this end, we first reduce Z to $\bar{Z}$. We then note that there exists an optimal trajectory from s_{init} to $\text{start}(\bar{B})$ in $\bar{Z}$ that only goes through states in $\bar{U}_1, \ldots, \bar{U}_n$. Therefore, to solve $(\bar{Z}, s_{\text{init}}, \text{start}(\bar{B}))$, we only need to consider $\bar{U}_1, \ldots, \bar{U}_n$. Furthermore, only some states and trajectories in each $\bar{U}_i$ are relevant. Namely, for $\bar{U}_1$, the relevant states are the ones we might start from: s_{init} and $s(\bar{U}_1)$. From these states, the relevant trajectories are the ones that optimally exit $\bar{U}_1$, starting from the states.[5] For $\bar{U}_2$, the relevant states are $s(\bar{U}_2)$ and states in $\bar{U}_2$ that could be reached by exiting $\bar{U}_1$ (at most $|\Sigma|$ such states). From these states, the relevant trajectories are the ones that optimally exit $\bar{U}_2$ as well as the optimal trajectories to get to $\bar{U}_1$ (might be optimal to go back to $\bar{U}_1$). The other $\bar{U}_i$ are analogous to $\bar{U}_2$ except that: the relevant trajectories of $\bar{U}_\alpha$ also include the optimal trajectories to $\bar{B}$; and, for $\bar{U}_n$, we do not calculate the ones that optimally exit $\bar{U}_n$ (since this would only stop $\bar{Z}$). With this, we form a graph G where nodes (arcs) corresponds to the relevant states (trajectories), labelling each arc with the relevant trajectory and its cumulative cost. Searching in G using Dijkstra's algorithm, from s_{init} to $\bar{B}$, we find an optimal trajectory z_1 to $(\bar{Z}, s_{\text{init}}, \text{start}(\bar{B}))$. This procedure is Part 1 in Algorithm 4. We then get an optimal trajectory z_2 to $(\bar{Z}, \text{start}(\bar{B}), s_{\text{goal}})$ using Proposition 4 (ii), and Dijkstra's algorithm to search in each $\bar{D}_{k_i}$. This is Part 2 in Algorithm 4. Finally, we combine z_1 and z_2 to get an optimal trajectory

Algorithm 4 Reduce_and_solve

Input: $(Z, s_{\text{init}}, s_{\text{goal}})$ and $(c_x^M, z_x^M)_{x \in \Sigma, M \in X}$.
Output: An optimal trajectory z to $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$.

Part 1: Solve $(\bar{Z}, s_{\text{init}}, \text{start}(\bar{B}))$
$\bar{Z} \leftarrow \text{Reduce}(Z, s_{\text{init}}, s_{\text{goal}}, (c_x^M, z_x^M)_{x \in \Sigma, M \in X})$
Set G to an empty graph ▷ To be constructed
Add nodes to G
for $i = 1, \ldots, n$ **do** ▷ Consider $\bar{U}_i$
  Get states to search from and too, I and D, in $\bar{U}_i$.
  for $s \in I$ **do**
    $(c_d, z_d)_{d \in D} \leftarrow \text{Dijkstra}(s, D, \bar{U}_i)$
    Add arcs corresponding to $(c_d, z_d)_{d \in D}$ in G
  end for
end for
$z_1 \leftarrow \text{Dijkstra}(s_{\text{init}}, \bar{B}, G)$ ▷ End of Part 1
Part 2: Solve $(\bar{Z}, \text{start}(\bar{B}), s_{\text{goal}})$
Let z_2 be an empty trajectory ▷ To be constructed
$(M, c) \leftarrow (\bar{D}_{k_1}, \text{start}(\bar{B}))$
$g \leftarrow$ corresponding state in M to $\bar{D}_{k_1-1}$
while $c \neq s_{\text{goal}}$ **do**
  $z \leftarrow \text{Dijkstra}(c, g, M)$
  $z_2 \leftarrow z_2 z$ ▷ Add z to z_2
  $c \leftarrow start(g)$
  if $c \neq s_{\text{goal}}$ **then**
    Let $\bar{D}_{k_i}$ be the MM with state c
    $M \leftarrow \bar{D}_{k_i}$
    $g \leftarrow$ corresponding state in M to $\bar{D}_{k_i-1}$
  end if
end while ▷ End of Part 2
return $z \leftarrow z_1 z_2$

[4]This is true if $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$ is feasible, by (i).

[5]More precisely, if one can exit $\bar{U}_1$ with $x \in \Sigma$, starting from s_{init}, then we find a trajectory t_x that does this optimally. The relevant trajectories from s_{init} are then all the found t_x. The case for $s(\bar{U}_1)$ is analogous.

Algorithm 5 Expand

Input: Optimal trajectory z to $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$ and $(c_x^M, z_x^M)_{x\in\Sigma, M\in X}$.
Output: Executes/saves optimal plan u to $(Z, s_{\text{init}}, s_{\text{goal}})$.

```
Let u be an empty trajectory                    ▷ To be constructed
for (q, x) in z do
    t ← Plan_expansion(q, x)
    u ← ut                                      ▷ Concatenate u and t
end for
if save u then return u
end if
```

z to $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$. We get time complexity:

Proposition 5: The time complexity of Algorithm 4 is

$$O\big(|\Sigma|^2\text{depth}(Z) + |\Sigma|\text{depth}(Z)\cdot\log(|\Sigma|\text{depth}(Z))\big) + O\big([b_s|\Sigma| + b_s\log(b_s)]\cdot\text{depth}(Z)\big)$$

where the first (second) O-term is from Part 1 (Part 2), and b_s is the maximum number of states in an MM. In particular, with bounded $|\Sigma|$ and b_s, we get $O(\text{depth}(Z)\cdot\log(\text{depth}(Z)))$.

2) Solving $(Z, s_{\text{init}}, s_{\text{goal}})$: We get an optimal plan u to $(Z, s_{\text{init}}, s_{\text{goal}})$ by conducting an optimal expansion of the optimal trajectory z to $(\bar{Z}, s_{\text{init}}, s_{\text{goal}})$ from Algorithm 4. The optimal plan u can be executed sequentially or obtained at once, with procedure given by Algorithm 5. Here, Plan_expansion is identical to Trajectory_expansion in Algorithm 3 except line 4 that is changed to [return x] or [apply x] if one wants the full plan u at once or sequential execution, respectively. We get time complexity:

Proposition 6: Executing an optimal plan u sequentially using Algorithm 5 has time complexity $O(\text{depth}(Z))$ to obtain the next input in u. Obtaining the full optimal plan at once has time complexity $O(\text{depth}(Z)|u|)$, where $|u|$ is the length of u.

IV. Numerical evaluations

In this section, we consider numerical case studies to validate the hierarchical planning algorithm given by Algorithm 1. Case study 1 demonstrates the scalability of the algorithm. Case study 2 show-case the algorithm on the robot application introduced in the motivation.

A. Case Study 1: Recursive System

1) Setup: To validate the efficiency of Algorithm 1, we consider an HiMM Z constructed recursively by nesting the same MM repeatedly to a certain depth. More precisely, the MM M we consider has tree states $Q = \{1, 2, 3\}$ with start state 2 and three inputs $\Sigma = \{x, y, z\}$ with transitions depicted in Fig. 4 (left) and units costs (not depicted). The recursion step is then done by replacing state 1 and 3 with M, with result given by Fig. 4 (right). This procedure is then repeated for the recently added MMs until we have reached a certain depth (e.g., in Fig. 4, we would replace state 2, 4, 5 and 7 with M). This yields the HiMM Z.

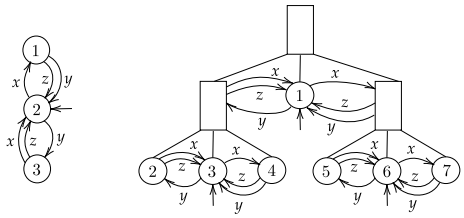

Fig. 4: MM M (left) and HiMM Z (right) with depth 2 for Case study 1.

We check the computing time of Algorithm 1 when varying the depth of Z. To this end, we let s_{init} and s_{goal} be opposite states in the HiMM (e.g., state 2 and 7 in Fig. 4) and compute the full optimal plan. For comparison, we also consider Dijkstra's algorithm for finding the optimal plan, applied to the equivalent flat MM.[6]

2) Result: The computing time is shown in Fig. 5 with $\text{depth}(Z)$ from 1 to 20. The online step finds optimal plans for all depths within milliseconds, while Dijkstra's algorithm is a bit faster for small depths ($\text{depth}(Z) \leq 6$) but takes several seconds for larger depths ($\text{depth}(Z) \geq 15$). In particular, for $\text{depth}(Z) = 20$, with about 2 million states, the online step finds an optimal plan in just 3.8 ms, compared to 108 s using Dijkstra's algorithm. Also, the offline step has a computing time comparable to Dijkstra's algorithm being slower for small depths (max 4 times slower), but negligible difference for large depths, and even slightly faster for $\text{depth}(Z) \geq 18$ (due to a slower increase). Hence, for large depth, even the computing time for the offline plus the online step is slightly faster than Dijkstra's algorithm.

B. Case Study 2: Robot Warehouse Application

1) Setup: We now consider the robot application introduced in the motivation, schematically depicted by Fig. 1. To formalise the example as an HiMM Z, we consider 10 warehouses ordered linearly, where the robot can move to any neighbouring house (e.g., to house 2 and 4 from house 3, and only to house 2 from house 1), at a cost of 100. This yields the MM M_1 corresponding to the top layer of Z, with house 1 as start state and inputs $\Sigma = \{\text{left}, \text{right}\}$.

Furthermore, each house is modelled as an MM M_2 having a single room. The room is a square 10×10 grid, where the robot at grid point (i, j) $(1 \leq i, j \leq 10)$ can move to any neighbouring (non-diagonal) grid point at a cost of 1 using inputs $\Sigma = \{\text{left}, \text{right}, \text{up}, \text{down}\}$. We also have a grid-point just outside the house, the entrance state, adjacent to $(1, 1)$. The robot can move between the entrance state and $(1, 1)$ with cost 1. From the entrance state, the robot can also exit M_2 by applying input $a \in \{\text{left}, \text{right}\}$. This a is then fed to M_1, which moves the robot to the corresponding neighbouring house. The MM M_2 has 101 states and four actions with the entrance state as start state.

Finally, at each grid-point inside the house, the robot has a work desk, modelled as an MM M_3, consisting of

[6]This flat MM can be obtained by simply checking all transitions and costs at each state in Z, and form the corresponding MM from this data.

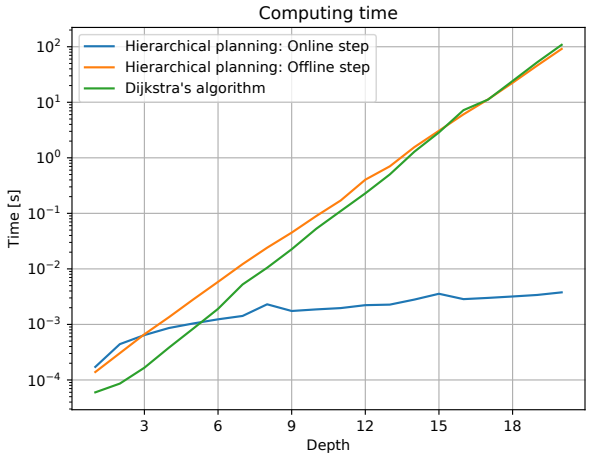

Fig. 5: Computing time for varying depth in Case Study 1.

9 laboratory test tubes arranged in a 3×3 test tube rack, where the robot can move between the test tubes or scan a tube using a robot arm. More precisely, at test tube (i, j) $(1 \leq i, j \leq 3)$, it can move the robot arm to any neighbouring tube (analogous to M_2) or scan the test tube, using inputs $\Sigma = \{\text{left}, \text{right}, \text{up}, \text{down}\} \cup \{\text{scan}\}$. When a test tube has been scanned, it remembers it and do not scan other tubes. Similar to M_2, we have an entrance state from which the robot can either enter the work desk (starting at test tube $(1, 1)$ and nothing scanned), or exit M_3 by applying input $a \in \{\text{left}, \text{right}, \text{up}, \text{down}\}$. This input a is then fed to M_2 which transition to the corresponding grid-point. From $(1, 1)$, we can also go back to the entrance state. All transition costs are set to 0.5 except the scanning, which costs 10. The MM M_3 has $9 \cdot 10 + 1 = 91$ states and inputs Σ, with entrance state as start state. The hierarchy of M_1, M_2 and M_3 yields Z.

2) Result: We set s_{init} to be the state where the robot is in house 1 at grid-point $(10, 10)$ having scanned test tube $(3, 3)$, and s_{goal} is identical to s_{init} except in house 10. That is, the robot has to move to house 10 and scan test tube $(3, 3)$ at grid-point $(10, 10)$. The online step finds an optimal plan in just 0.022 s compared to 3.4 s using Dijkstra's algorithm. Also, the offline step takes only 2.0 s, hence, even the offline plus online step is faster than Dijkstra's algorithm.

V. Conclusion

In this paper, we have considered a planning problem for an HiMM and developed an algorithm for efficiently computing optimal plans between any two states. The algorithm consists of an offline step and an online step. The offline step computes exit costs for each MM in a given HiMM Z. This step is done only once for Z, with time complexity scaling linearly with the number of MMs in Z. The online step then computes an optimal plan, from a given initial state to a goal state, by constructing an equivalent reduced HiMM $\bar{Z}$ (based on the exit costs), computing an optimal trajectory z for $\bar{Z}$, and finally expanding z to obtain an optimal plan u to Z. The online step finds an optimal trajectory z to $\bar{Z}$ in time $O(\text{depth}(Z) \log(\text{depth}(Z)))$ and obtains the next optimal input in u from z in time $O(\text{depth}(Z))$, or the full optimal plan u in time $O(\text{depth}(Z)|u|)$. We validated our algorithm on large HiMMs having up to 2 million states, including a mobile robot application, and compared our algorithm with Dijkstra's algorithm. Our algorithm outperforms the latter, where the partition into an offline and online step reduces the overall computing time for large systems, and the online step computes optimal plans in just milliseconds compared to tens of seconds using Dijkstra's algorithm.

Future work includes extending the algorithm to efficiently handle changes in the given HiMM (e.g., modifying an MM in the HiMM), and comparing it with other hierarchical methods. Another challenge is to extend the setup to stochastic systems.

References

[1] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM SIGSOFT Software Engineering Notes*, 23(6):175–188, 1998.

[2] H. Bast, et al. Route planning in transportation networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.

[3] O. Biggar, M. Zamani, and I. Shames. Modular decomposition of hierarchical finite state machines. *arXiv preprint arXiv:2111.04902*, 2021.

[4] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2008.

[5] J. Dibbelt, B. Strasser, and D. Wagner. Customizable contraction hierarchies. *Journal of Experimental Algorithmics (JEA)*, 21:1–49, 2016.

[6] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, dec 1959.

[7] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *25th Annual Symposium on Foundations of Computer Science, 1984.*, pages 338–346, 1984.

[8] D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[9] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[10] C. Ma and W. M. Wonham. Nonblocking supervisory control of state tree structures. *IEEE Transactions on Automatic Control*, 51(5):782–793, 2006.

[11] J. Maue, P. Sanders, and D. Matijevic. Goal-directed shortest-path queries using precomputed cluster distances. *ACM J. Exp. Algorithmics*, 14, 2010.

[12] George H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, 1955.

[13] I. Millington and J. Funge. *Artificial intelligence for games*. CRC Press, 2018.

[14] R. H. Möhring, et al. Partitioning graphs to speedup dijkstra's algorithm. *Journal of Experimental Algorithmics (JEA)*, 11:2–8, 2007.

[15] P. Schillinger, S. Kohlbrecher, and O. Von Stryk. Human-robot collaborative high-level control with application to rescue robotics. In *2016 IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2796–2802. IEEE, 2016.

[16] E. Stefansson and K. H. Johansson. Hierarchical finite state machines for efficient optimal planning in large-scale systems. *arXiv preprint arXiv:2212.03724*, 2023.

[17] O. N Timo, et al. Reachability in hierarchical machines. In *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*, pages 475–482. IEEE, 2014.

[18] X. Wang, Z. Li, and W. M. Wonham. Real-time scheduling based on nonblocking supervisory control of state-tree structures. *IEEE Transactions on Automatic Control*, 66(9):4230–4237, 2020.

[19] M. Yannakakis. Hierarchical state machines. In *IFIP International Conference on Theoretical Computer Science*, pages 315–330. Springer, 2000.