Extracting generalizable skills from a single plan execution using abstraction-critical state detection

Khen Elimelech, Lydia E. Kavraki and Moshe Y. Vardi

***Abstract*— Robotic task planning is computationally challenging. To reduce planning cost and support life-long operation, we must leverage prior planning experience. To this end, we address the problem of extracting reusable and generalizable abstract skills from successful plan executions. In previous work, we introduced a supporting framework, allowing us, theoretically, to extract an abstract skill from a single execution and later automatically adapt it and reuse it in new domains. We also proved that, given a library of such skills, we can significantly reduce the planning effort for new problems. Nevertheless, until now, abstract-skill extraction could only be performed manually. In this paper, we finally close the automation loop and explain how abstract skills can be practically and automatically extracted. We start by analyzing the desired qualities of an abstract skill and formulate skill extraction as an optimization problem. We then develop two extraction algorithms, based on the novel concept of abstraction-critical state detection. As we show experimentally, the approach is independent of any planning domain.**

I. Introduction

To perform autonomously tasks such as rearrangement, manipulation, and navigation, robots must be able to plan their actions. Such planning is usually done in two levels: planning of discrete, high-level actions ("task planning" [1]), and of continuous robot motions ("motion planning" [2]); this work is concerned with the first level. Realistic robotic task planning is usually computationally challenging, especially considering complex robots, large planning domains, or complex task specifications. Hence, to support life-long operation and reduce future planning cost, it is important to be able to improve from successful planning experience. Yet, past problem solutions are typically not immediately reusable and should be adapted before becoming applicable.

To this end, in our preliminary work [3], we introduced a novel computational framework for automatic skill transfer, allowing to accumulate generalizable planning experience, which can later be reused in new contexts. With this approach, transfer is done in two stages: the "learning phase," in which we process successfully executed plans to build a library of cached *abstract skills*; and the "planning phase," in which we try to match abstract skills from the library to new planning problems, in order to accelerate their solution. This transfer approach is visually demonstrated in Fig. 1.

We investigated the "planning phase" in prior work [4], where we provided useful algorithms for planning with skills. Yet, there, we considered the library of skills to be given. Now, we wish to close the automation loop by addressing the "learning phase." Specifically, we want to explain how, given just a single successful plan execution, to extract generalizable abstract skills, to be added to the library.

Contribution: To allow automated skill extraction, we formulate the extraction problem as an optimization of an *abstract-skill score*. This concept, which we introduce and justify here, is used to balance the trade-off between the generalizability and instructiveness of the extracted abstract skill. We first formulate the problem of extracting a single abstract skill from each execution, and then extend it, to consider segmentation-based extraction of multiple skills. We also present and demonstrate practical solution algorithms for both problems, based on the novel concept of Abstraction-Critical State Detection (ACSD). These contribution are a crucial step for bringing the transfer framework from theory into practice and are independent of any planning domain.

Related work: We strongly emphasize that the contribution of this work is *qualitative*—explaining how to automate a previously-manual process. Our algorithms yield skills that can only be used in the context of our transfer framework. By such, this contribution cannot be directly compared to other techniques without comparing the entire framework, which is beyond the scope of this paper. For quantitative benefits and motivation, see [3], [4]. Further, as implied, abstract-skill extraction can be considered a form of "learning." However, our approach is categorically different than standard works on "skill learning" or "learning from demonstration" (e.g., [5], [6], [7], [8]): these techniques learn a policy, while we focus on plans; they require multiple expert demonstrations, and we—a single successful execution; they produce skills using neural networks, and we—using "abstraction keys" (as we shall explain); they achieve generalization thanks to multiple examples, and we—thanks to the abstract representation. Also, various "task planning" and "task and motion planning" [9] works suggest to reduce planning cost by planning in hierarchical abstract domains (e.g., [10], [11], [12]). We do not, however, consider planning in the abstract domain, nor address a planning problem. We only use abstraction as a tool to represent a skill (plan) in a generalizable manner. Finally, some works from computer vision (e.g. [13], [14]) also address the problem of "critical point detection" or "trajectory segmentation." They, however, view the problem from a purely geometric perspective and not in the context of planning. We detect critical states based on their effect on the abstraction, and aim to segment optimal abstract skills. Our approach also focuses on discrete traces (whether in geometric or symbolic domains) and not continuous trajectories.

Work on this paper was supported in part by NSF-IIS-1830549. The authors are with the Department of Computer Science, Rice University, Houston, TX 77005, USA. {elimelech,kavraki,vardi}@rice.edu

To appear at the 2023 IEEE International Conference on Robotics and Automation (ICRA). Preprint version.

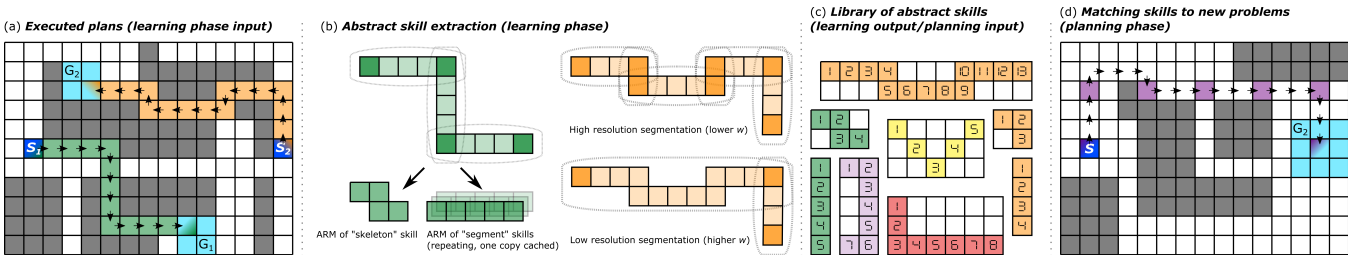

Fig. 1: Example of skill transfer using a "geometric abstraction key," which allow us to perform spatial transformations (translation, rotation, scaling) on state traces by specifying projection and reconstruction functions. (a) Successfully-executed plans in a grid-world domain, e.g., for robotic navigation tasks. (b) Segmentation of the state trace from each execution, to identify a "skeleton" of abstraction-critical states, as explained in Sec. IV-E; the segments and skeleton can each be abstracted (projected), to yield an abstract skill. (c) Abstract Road Maps (ARMs) representing a library of extracted abstract skills; this key's Information Function (IF), which expresses the "level of abstraction," is defined by the volume of each mini-grid. (d) Matching and reconstructing an ARM to guide the solution of a new planning problem.

II. Preliminaries

A. Abstraction keys for skill transfer

In this work, we refer to a successfully-executable task plan as a "skill." In previous work [3], [4], we presented and motivated a computational framework that allows to perform automated transfer of such skills between tasks and domains.

This transfer approach is based on two key ideas. First, instead of an action trace, a skill can be represented using a state trace, extracted from its execution; we refer to this state trace as the skill's "road map" RM. Second, road maps of such skills can be adjusted and transferred to new scenarios using "computational devices" called *abstraction keys*.

Definition 1: An *abstraction key* AK is a tuple $\left(\texttt{project}_p, \texttt{reconst}_p, \mathcal{P}\right)$ comprising a parametric state projection function $\texttt{project}_p\colon \mathcal{S} \to \Xi$; its inverse—a parametric state reconstruction function $\texttt{reconst}_p\colon \Xi \to \mathcal{S}$; and the parameter space $\mathcal{P}$, such that $p \in \mathcal{P}$.

The space Ξ is a key-induced abstraction of the state space $\mathcal{S}$. The set of valid parameters for projection of a state $\boldsymbol{S}$ is marked $\mathcal{P}_{\boldsymbol{S}}$, and the set of valid parameters for reconstruction of an abstract state $\boldsymbol{\xi}$ is marked $\mathcal{P}^{\boldsymbol{\xi}}$.

Using a chosen AK, transfer of a state trace (i.e., a skill's road map) is performed in two stages: by first projecting it into the (AK-induced) abstract domain Ξ, and then reconstructing it into the destination domain, in an attempt to match a new planning problem. A road map can thus be transformed and adjusted to a new problem by choosing different parameters during projection and reconstruction. Since the two stages are independent, the projected road map represents an *abstract skill*, which can be cached for reconstruction on-demand. Of course, to project and reconstruct road maps, we must provide valid parameter values.

Definition 2: Considering an abstraction key AK and a state trace $\boldsymbol{T}$, a parameter p is *valid for projection* if

$$p \in AK.\mathcal{P}_{\boldsymbol{T}}, \text{ where } AK.\mathcal{P}_{\boldsymbol{T}} \doteq \cap_{\boldsymbol{S}\in\boldsymbol{T}} AK.\mathcal{P}_{\boldsymbol{S}}. \quad (1)$$

Every abstraction key allows us to perform a certain *type* of transformation on states. In our previous works, we formulated and demonstrated the usage of several such keys. Among them are "symbol stripping" and "attention" [3], for transforming symbolic states; and "geometric keys" [4], used for spatial transformation of geometric states (as in Fig. 1).

B. Learning and planning from abstract skills

To reflect the two-phased skill transfer technique, abstract skills can be processed in two distinct phases. First, the "learning phase," in which we try to build a library of abstract skills from known and successful plan executions. Each such execution defines a state trace $\boldsymbol{T} = (\boldsymbol{S}_0, \ldots, \boldsymbol{S}_n)$. To extract an abstract skill from an execution, we should choose an abstraction key AK, a projection parameter p, and a road map $RM \subseteq \boldsymbol{T}$ to be projected. Then, an abstract skill $\boldsymbol{K} = (ARM, AK)$ with the Abstract Road Map $ARM \doteq \texttt{project}_p\left(RM\right)$ can be cached.

After accumulating a library of abstract skills from past experience, we can exploit it during the "planning phase," to solve efficiently new planning problems. To do so, during planning, we should search for an abstract skill (ARM, AK), for which exists a reconstruction parameter p that allows to appropriately reconstruct ARM into the new context; i.e., p for which the Reconstructed Road Map $RRM \doteq AK.\texttt{reconst}_p\left(ARM\right)$ matches the task goals, domain, and current state. If a match is found, we can use this RRM to guide the action search and efficiently solve the planning problem. In our prior work [4], we showed that such matching can be formulated as a constraint satisfaction problem and automatically solved. A visual example of such reconstruction is given in Fig. 1d.

C. Problem definition

The solution and benefits of "planning from abstract skills" were thoroughly formulated, discussed, and demonstrated in our prior work [4]. However, the extraction formulation provided before left us with practical questions, for which the library of abstract skills in that work had to be manually crafted. For example, we did not specify how to choose the abstraction key nor the parameter for projection. Considering different abstraction keys can result in abstract skills with widely different qualities; e.g., choosing a key that yields poor abstraction would limit the skill usefulness later on. We want to be able to make an informed choice for the keys, especially when multiple valid options are available.

Further, we also did not specify how to choose the road map RM to be projected. We would typically not want to use the entire state trace $\boldsymbol{T}$ from the original execution, as

this might unnecessarily restrict the quality of the extracted abstract skill, or prevent us from extracting skills altogether (when there is no valid parameter p for projecting the entire trace). *Automatically* choosing a road map that yields a useful and generealizable abstract skill is not trivial.

Thus, the overarching goal of this work is to explain how to *practically* learn abstract skills, and specifically: (i) understand skill qualities and what makes an abstract skill useful; (ii) provide a well-defined formulation of the problem of optimal-skill extraction; and (iii) provide practical extraction algorithms, to solve the formulated problem.

III. Instructiveness vs. applicability trade-off

Before we can explain how to extract abstract skills, we should describe the qualities of abstract skills that we would like to maintain. Thus, let us at this point consider a given abstract skill $\boldsymbol{K} \doteq (ARM, AK)$. Two qualities can be used to describe $\boldsymbol{K}$: *instructiveness* and *applicability*.

Instructiveness depends on the number of states the ARM contains and their "level of abstraction," from the perspective of AK. For intuition, using a highly-instructive abstract skill to solve a new planning problem is expected to be easier, by requiring a "smaller" transformation to reconstruct the ARM into the RRM, and/or less effort to follow the RRM and complete the solution. To put simply, this abstract skill's ARM is expected to highly overlap with the state trace of the final execution. Applicability describes in how many planning scenarios the abstract skill can potentially be applied, i.e., for how many scenarios we can match and reconstruct ARM, by finding a reconstruction parameter.

These qualities, however, are not practically measurable. As a proxy to understand their relationship and trends, we introduce the notion of *information*—an auxiliary function we can define to extend an abstraction key. Such a function shall consider the *type* of information the key abstracts during projection, and express the *amount* of it in the trace.

Definition 3: For an abstraction key AK, an *Information Function* $\texttt{IF}$ is a function $traces(\mathcal{S}) \cup traces(\Xi) \to \mathbb{R}_{\geq 0}$, where $traces(space) \doteq \cup_{n=0}^{\infty} space^n$, with the properties:
(i) Monotonically increasing, i.e.,

$$\forall \boldsymbol{T}_1 \subseteq \boldsymbol{T}_2 \implies \texttt{IF}(\boldsymbol{T}_1) \leq \texttt{IF}(\boldsymbol{T}_2).$$

(ii) Triangle inequality, i.e.,

$$\forall \boldsymbol{T}_1, \boldsymbol{T}_2,\ \texttt{IF}([\boldsymbol{T}_1, \boldsymbol{T}_2]) \leq \texttt{IF}(\boldsymbol{T}_1) + \texttt{IF}(\boldsymbol{T}_2).$$

(iii) Information consistency, i.e., $\forall \boldsymbol{T}_1, \boldsymbol{T}_2, p \in AK.\mathcal{P}$,

$$\texttt{IF}(\boldsymbol{T}_1) \leq \texttt{IF}(\boldsymbol{T}_2) \iff \texttt{IF}\left(\texttt{project}_p(\boldsymbol{T}_1)\right) \leq \texttt{IF}\left(\texttt{project}_p(\boldsymbol{T}_2)\right),$$

$$\texttt{IF}(\boldsymbol{T}_1) \leq \texttt{IF}(\boldsymbol{T}_2) \iff \texttt{IF}(\texttt{reconst}_p(\boldsymbol{T}_1)) \leq \texttt{IF}(\texttt{reconst}_p(\boldsymbol{T}_2)).$$

An $\texttt{IF}$ can often be intuitively defined for an abstraction key, to express an ARM "compactness;" e.g., as indicated in Fig. 1, for "geometric keys" the $\texttt{IF}$ may measure the volume of the trace's bounding box; for "symbol stripping" [3]—the number of symbols in the trace's states; for "attention"—the number of different propositions. According to this definition, more information or longer length of the ARM should express higher instructiveness of the abstract skill:

$$\uparrow \texttt{len}(ARM) \implies \uparrow \textit{Instructiveness}(ARM), \quad (2)$$

$$\uparrow \texttt{IF}(ARM) \implies \uparrow \textit{Instructiveness}(ARM). \quad (3)$$

Applicability can also be viewed as a function of the information and length of the ARM. For the abstract skill to be applicable to a planning problem, its ARM must be reconstructable into the relevant domain. Each state in the ARM adds a constraint to the reconstitution process—for the reconstructed state to not conflict with the task and domain. Further, more information in the ARM makes such constraints more restrictive, as it limits us only to scenarios in which it is relevant. Thus, abstract skills with longer ARMs and higher $\texttt{IDF}$ typically suffer from lesser applicability:

$$\uparrow \texttt{len}(ARM) \implies \downarrow \textit{Applicability}(ARM), \quad (4)$$

$$\uparrow \texttt{IF}(ARM) \implies \downarrow \textit{Applicability}(ARM). \quad (5)$$

From this discussion we may conclude the instructiveness and applicability are "opposite" qualities, i.e.,

$$\uparrow \textit{Instructiveness} \iff \downarrow \textit{Applicability}. \quad (6)$$

Thus, for a skill to be practically useful, it must express balance between the two qualities. Finding a very instructive skill that is hardly applicable, or a widely applicable skill that is hardly instructive, would not be useful. The relationship among the aforementioned measures is visualized in Fig. 2a.

IV. Optimal-skill extraction

Abstract-skill extraction requires choosing three elements: an abstraction key, a projection parameter, and a road map. While in Sec. III we described the qualities a *given* abstract skill, we now explain how to choose these elements, to control the qualities of the extracted skills.

A. Choosing the projection parameter

Consider first an abstraction key AK and a road map RM taken from a certain execution. To extract an abstract skill using this road map, we must choose a parameter $p \in \mathcal{P}_{RM}$ for projection. Next, we will build on the previous discussion to explain the basis for choosing this parameter. Since the road map is given, the extracted skill qualities are only determined by the information of the projected road map. Yet, we cannot control the information directly, but only via the choice of p. To assess how p affects the information of the projected road map, and thus guide us into making the optimal choice, we define the *abstraction score*.

Definition 4: For an abstraction key AK with an information function $\texttt{IF}$, the induced *abstraction score function* $\texttt{AbstScore}\colon traces(\mathcal{S}) \times \mathcal{P} \to [0,1]$ is defined as

$$\texttt{AbstScore}(RM, p) \doteq \frac{\texttt{IF}(RM) - \texttt{IF}\left(\texttt{project}_p(RM)\right)}{\texttt{IF}(RM)}. \quad (7)$$

For a road map RM, the maximal abstraction score value is referred to as its *potential* score and marked as

$$\texttt{AbstScore}^*(RM) \doteq \max_p \texttt{AbstScore}(RM, p). \quad (8)$$

The maximizing p is marked as p^*.

It is easy to see that a higher abstraction score corresponds to lower information of the projected (abstract) road map:

$$\uparrow \texttt{AbstScore}\,(RM, p) \iff \downarrow \texttt{IF}\left(\texttt{project}_p\,(RM)\right). \quad (9)$$

Hence, according to the relationship in (5), choosing p with a higher abstraction score leads to wider applicability:

$$\uparrow \texttt{AbstScore}\,(RM, p) \implies \uparrow \textit{Applicability}\left(\texttt{project}_p\,(RM)\right). \quad (10)$$

Yet, from the relationship (3), we know it also means reducing the abstract skill instructiveness:

$$\uparrow \texttt{AbstScore}\,(RM, p) \implies \downarrow \textit{Instructiveness}\left(\texttt{project}_p\,(RM)\right). \quad (11)$$

B. Choosing the road map

Now, let us assume the road map RM to be projected is not explicitly given. Instead, we have access to an entire state trace $\boldsymbol{T}$, taken from a successful execution. To extract an abstract skill, we do not necessarily have to use the entire trace, as this, as we shall see, might limit the qualities of the extracted skill. Hence, we want to explain the basis for choosing a subset of states from $\boldsymbol{T}$, to serve as the RM.

Extending a road map with additional states can only reduce the number of parameters that are valid for projection. We thus can intuitively conclude that the potential abstraction score is *monotonically decreasing*, i.e.,

$$\boldsymbol{T}_1 \subseteq \boldsymbol{T}_2 \;\Rightarrow\; \texttt{AbstScore}^*\,(\boldsymbol{T}_1) \geq \texttt{AbstScore}^*\,(\boldsymbol{T}_2) \quad (12)$$

This property means that we cannot possibly achieve a "better" abstraction of a road map (in terms of abstraction score) by extending it. Hence,

$$\uparrow \texttt{len}\,(RM) \implies \downarrow \texttt{AbstScore}^*\,(RM) \quad (13)$$

Clearly, $\texttt{len}\,(RM) = \texttt{len}\,(ARM)$. Hence, from combining (13) with the two relationships in (11) and (2), we learn that *choosing* a road map RM with a higher `length` leads to a *significantly* higher potential instructiveness of the skill:

$$\uparrow \texttt{len}\,(RM) \implies \uparrow\uparrow \textit{Instructiveness}\,(ARM^*), \quad (14)$$

where $ARM^* \doteq \texttt{project}_{p^*}\,(RM)$. Yet, from combining (13) with (10) and (4), we can also learn that this leads to a *significantly* lower potential applicability of the skill:

$$\uparrow \texttt{len}\,(RM) \implies \downarrow\downarrow \textit{Applicability}\,(ARM^*). \quad (15)$$

These relationships are visualized in Fig. 2b.

C. The abstract-skill score and problem formulation

As we see, "improving" each of our two decision variables (RM, as measured by its length, and p, as measured by the abstraction score) has non-trivial, conflicting effects on the instructiveness and applicability of the resulting abstract skill. We want to controllably choose RM and p to balance both qualities and produce the "optimal skill."

To do so, we wish to find an appropriate measure to quantify the overall quality of our choice. We want this measure to express the relationships we developed earlier. We also want to include a weight parameter $w \in [-1, 1]$, to help express preference for one of the qualities over the other. The function should hold the following properties: choosing a higher w increases the impact of the RM length and decreases the impact of the abstraction score, to encourage selection of instructive skills; choosing a lower w decreases the impact of the RM length and increases the impact of the abstraction score, to encourage selection of applicable skills. Altogether, these conditions yield the *abstract-skill score*:

$$\texttt{SkillScore}_w\,(RM, p) \doteq (\texttt{len}\,(RM) - 1)^{\frac{1+w}{2}} \cdot (\texttt{AbstScore}\,(RM, p) + 1)^{1-w} - 1. \quad (16)$$

Finally, using the abstract-skill score we can formulate the skill extraction problem as an optimization problem:

$$\begin{aligned} \texttt{given:}\;& \text{state trace } \boldsymbol{T} \doteq (\boldsymbol{S}_0, \ldots, \boldsymbol{S}_n),\\ & \text{a quality preference weight } w \in [-1, 1],\\ \texttt{find:}\;& \text{an abstraction key } AK,\\ & \text{a parameter } p,\\ & i_0, i_1 \in \{0, \ldots, n\}, \text{ s.t. } RM \doteq (\boldsymbol{S}_{i_0}, \ldots, \boldsymbol{S}_{i_1}),\\ \texttt{s.t.:}\;& p \text{ is a valid parameter choice for projecting } RM,\\ \texttt{to max:}\;& \texttt{SkillScore}_w\,(RM, p). \end{aligned} \quad (17)$$

After solving this problem, the returned skill would be $(AK.\texttt{project}_p\,(RM), AK)$, which we can cache in our abstract skill library. Note that to keep the formulation straightforward, we consider RM to be a segment of $\boldsymbol{T}$.

It is clear that for a given RM, choosing p with a higher abstraction score would always increase the abstract-skill score. We can hence measure the potential abstract-skill score

$$\texttt{SkillScore}^*\,(RM) \doteq \texttt{SkillScore}\,(RM, p^*), \quad (18)$$

and assume that for every RM, we would always prefer its abstraction-score-maximizing parameter p^* (from (8)). Yet, we recall that extending RM also constrains p^*, and lowers the potential abstraction score. Thus, the potential abstract-skill score (as a function of the RM length) has a parabolic

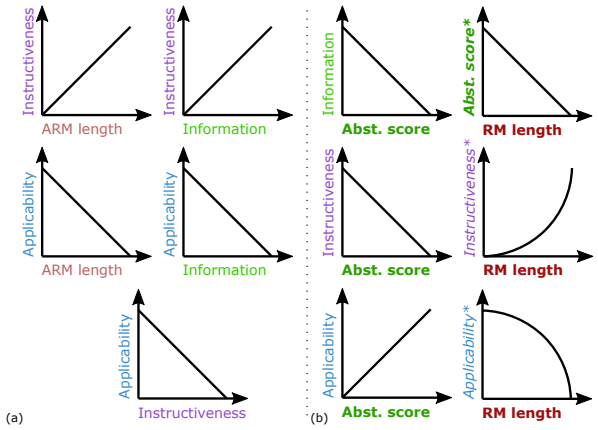

Fig. 2: (a) Measurable vs. immeasurable qualities of an abstract skill, and the instructiveness-applicability trade-off. (b) Controllable (bold) vs. uncontrollable qualities of an extracted abstract skill, and the road map length-potential abstraction score trade-off.

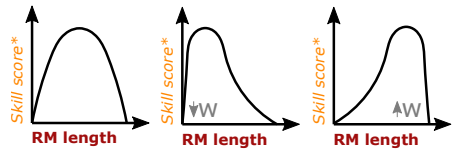

Fig. 3: The potential abstract-skill score as a function of $\texttt{len}\,(RM)$.

nature, as visualized in Fig. 3. As also shown, increasing w "pushes" this parabola to the right, and decreasing w—to the left. We also conclude that the choices of RM and p are interconnected, and we must consider the abstraction when choosing the road map. Attempting to pre-determine the road map without considering the abstraction score can lead to a sub-optimal skill, e.g., a skill with low abstraction score and low applicability (right side of the parabola), or make us unnecessarily conservative (left side of the parabola).

D. Extraction algorithm

To choose the optimal RM, we shall incrementally analyze segments of $\boldsymbol{T}$, and measure changes in the potential abstraction score. This would allow us to Detect the *Abstraction-Critical States (ACSD)*—states whose inclusion in RM would have an undesirable effect on the abstract-skill score. Practically, we suggest a "greedy" (steepest-ascent) hill climbing algorithm, to maximize the abstract-skill score. We start from the collection of all basic segments by coupling together each pair of consecutive states in the trace. We then try to extend incrementally and dynamically each such basic segment by merging into it additional states (from its right or left), as long as doing so increases its potential abstract-skill score. Once no more extensions are possible, we terminate and return the ARM of the segment with the maximal score. Such analysis can be done for every relevant abstraction key, to return the overall best skill. We can also return the "top-k" skills, if desired. These steps are summarized in Alg. 1a.

E. Segmentation-based multi-skill extraction

Until now, we considered extraction of a single skill from the trace by detecting the optimal segment. Though, with only a slight adjustment of the previous formulation, we can, in fact, perform a full segmentation of the trace, allowing us to learn multiple skills at once, with a single algorithm run.

1) Abstraction-critical skeleton: The approach is based on detecting a *skeleton* $RM_{skeleton}$ of abstraction-critical states (instead of just two). A skeleton is any sub-trace of states from $\boldsymbol{T}$, not necessarily consecutive, that contains both ends of the trace $\boldsymbol{T}$, i.e., $\boldsymbol{S}_0, \boldsymbol{S}_n$. A skeleton of length $m+1$ breaks $\boldsymbol{T}$ into m segments, which can be used to create individual skills. Nevertheless, *the skeleton, as it is a road map of states, can also be used to define a skill.* Intuitively, "skeleton skills" and "segment skills" are complementary to each other. Segments represent sequences of local transitions, while the skeleton consolidates each such sequence, to capture only major transitions. Thus, a segment skill can usually be used later as a building block for planning, providing a short-horizon, detailed guidance, while a skeleton can typically provide a long horizon, general guideline for the plan.

Algorithm 1: Abstract-skill extraction via ACSD.

Algorithm a `Single_Skill_Extract` (*state trace* $\boldsymbol{T}$ *of length* n*, preference weight* w*, abstraction key* AK)

$ARMs \leftarrow [\,]$
for i **from** 0 **to** $n-1$ **do** `/* every base segment */`
  $l, r \leftarrow i, i+1$
  $score,\, p^* \leftarrow AK.\texttt{SkillScore}^*_w\,(\boldsymbol{T}[l:r])$
  while $l > 0$ *or* $r < n$ **do** `/* try to extend */`
    $l\_score,\, l\_p^* \leftarrow AK.\texttt{SkillScore}^*_w\,(\boldsymbol{T}[l-1:r])$
    $r\_score,\, r\_p^* \leftarrow AK.\texttt{SkillScore}^*_w\,(\boldsymbol{T}[l:r+1])$
    if $score > l\_score$ *and* $score > r\_score$ **then**
      break `// extended to max`
    else if $l\_score > r\_score$ **then**
      $l--$ `// extend left preferred`
      $score,\, p^* \leftarrow l\_score,\, l\_p$
    else
      $r++$ `// extend right preferred`
      $score,\, p^* \leftarrow r\_score,\, r\_p$
  end
  $segment\_ARM \leftarrow AK.\texttt{project}_{p^*}\,(\boldsymbol{T}[l:r])$
  $ARMs.append((segment\_ARM,\, score))$
end
return $\operatorname{argmax} ARMs$ `// best-scoring skill ARM`

Algorithm b `Segmentation_Based_Extract` (*state trace* $\boldsymbol{T}$*, preference weight* w*, abstraction key* AK)

$SI \leftarrow (0, \ldots, \texttt{len}\,(\boldsymbol{T}) - 1)$ `// Skeleton Indices`
$curr\_score \leftarrow \texttt{seg\_score}(\boldsymbol{T}, SI, w)$ `// as in (19)`
`// perform mergers while score increases`
$rewards \leftarrow$ `merger_rewards` $(\boldsymbol{T},\, SI,\, curr\_score)$
while $\max(rewards) \geq 0$ **do**
  $index\_state\_to\_pop \leftarrow \texttt{index\_of\_max}(rewards) + 1$
  $SI.\texttt{pop}(index\_state\_to\_pop)$
  $curr\_score \leftarrow \texttt{seg\_score}(\boldsymbol{T}, SI, w)$
  $rewards \leftarrow$ `merger_rewards` $(\boldsymbol{T},\, SI,\, curr\_score)$
end
`// return ARMs of all extracted skills`
$ARMs \leftarrow [\,]$
for i **from** 0 **to** $\texttt{len}\,(SI) - 2$ **do**
  $segment \leftarrow \boldsymbol{T}[SI[i], \ldots, SI[i+1]]$
  $abs\_score,\, p^* \leftarrow AK.\texttt{AbstScore}^*\,(segment)$
  $ARMs.append(AK.\texttt{project}_{p^*}\,(segment))$
end
$abs\_score,\, p^* \leftarrow AK.\texttt{AbstScore}^*\,(skeleton)$
$ARMs.append(AK.\texttt{project}_{p^*}\,(skeleton))$
return $ARMs$

Procedure `merger_rewards` (*state trace* $\boldsymbol{T}$*, skeleton indices* SI*, current segmentation score* $curr\_score$)

$rewards \leftarrow [\,]$
for i **from** 1 **to** $\texttt{len}\,(SI) - 2$ **do**
  `// to merge, remove state from skeleton`
  $SI^- \leftarrow SI[0, \ldots, i-1, i+i, \ldots, end]$
  `// calc change in score after merger`
  $rewards.\texttt{append}(\texttt{seg\_score}(\boldsymbol{T}, SI^-, w) - curr\_score)$
end
return $rewards$

2) Problem formulation: The segmentation problem formulation is similar to the previous one. Here, we should find a skeleton (instead of just two indices) and projection parameters for each of the segments and the skeleton itself (instead of just one). The segmentation score (objective) is the norm of the vector of segment scores; e.g., the L^2 norm:

$$\|(ss_1, \ldots, ss_m)\|_2, \text{ where } ss_j \doteq \texttt{SkillScore}_w\,(RM_j, p_j) \tag{19}$$

The length of the skeleton does not need to user-specified,

as it would be determined by solving the optimization problem. Similarly to before, increasing w encourages longer segments with lower potential abstraction score over shorter segments with higher potential abstraction score. Hence, w can implicitly be used to determine the granularity of the extracted skeleton, as indicated in Fig. 4.

3) Segmentation algorithm: To find a solution, we formulate another hill climbing algorithm, which operates similarly to the previous algorithm. Here, we start from maximal segmentation of the trace into pairs of states, i.e., a skeleton containing all the states. However, here, instead of trying to iteratively extend a selected segment with a single state from either end, we check which, if any, pairs of consecutive segments can be merged, in order to best increase the overall score. Such merger is performed by removing the critical state between the segments from the skeleton. The algorithm terminates when no merger can further improve the score. The algorithm steps are summarized in Alg. 1b.

V. Experimental results

Earlier in this paper, in Fig. 1, we visualized the process of segmentation-based skill extraction in a geometric domain. In our prior work [4], we used a skill library similar to the one shown in Fig. 1 to demonstrate "planning from skills." Yet, in that work, the extraction of skills had to be manually done. Now, using the algorithms provided here, extraction of such skills becomes automatable (given appropriate executions).

As we mentioned, the approach is also relevant to non-geometric domains, as we show next. Thus, let us consider a general, symbolic domain with state space: $\mathcal{S} \doteq \{1,2,3\} \times \{x,y,z\} \times \{\bigstar, \spadesuit, \clubsuit\} \times \{\blacksquare, \blacksquare, \blacksquare\}$. A state space of this sort is especially relevant to manipulation planning domains, as it can express, e.g., types, locations, and amounts of objects. In this domain, we can use the "attention" abstraction key, previously formulated in [3]. This key allows us to achieve a compact skill ARM, by exporting to the parameter the "non-changing" state variables during projection. Thus, intuitively, the `IDF` for this key is simply the number of "changing" state variables. A demonstration of the segmentation results in this domain is provided in Fig. 4. Specifically, we can see that a "naive" extraction attempt on that trace, i.e., considering the entire trace as the road map, would fail (as there is no valid parameter for projection).

For a robust verification of the extraction technique, and to compare its algorithmic variations, we ran the the algorithms on numerous randomly-generated executions from the aforementioned domain. The results are summarized in Table I. These results corroborate our previous analysis. As we can see, for both single-skill and segmentation-based extractions, increasing the value of w leads to skills of higher length, and lower abstraction score. We also learn that the quality of skills returned from segmentation seem to be lower than the quality of a single extraction, since the first optimizes the total skill-quality. This could be mitigated by replacing the norm type in the segmentation score definition. In any case, segmentation holds the benefit of returning multiple skills at once, including the useful "skeleton skill."

$(1,x,\bigstar,\blacksquare) \triangleright (2,x,\bigstar,\blacksquare) \triangleright (1,x,\bigstar,\blacksquare) \triangleright (2,y,\bigstar,\blacksquare) \triangleright$
$(1,z,\bigstar,\blacksquare) \triangleright (1,y,\spadesuit,\blacksquare) \triangleright (1,y,\spadesuit,\blacksquare) \triangleright (1,y,\spadesuit,\blacksquare)$

(a) Original execution.

ARM of "skeleton" skill:

$(x,\bigstar,\blacksquare) \triangleright (z,\bigstar,\blacksquare) \triangleright (y,\spadesuit,\blacksquare) \triangleright (y,\spadesuit,\blacksquare)$, $p^* = (1,?,?,?)$, $\texttt{AbstScore}^* = 1$

ARMs of "segment" skills:

$(1,x,\blacksquare) \triangleright (2,x,\blacksquare) \triangleright (1,x,\blacksquare) \triangleright (2,y,\blacksquare) \triangleright (1,z,\blacksquare)$, $p^* = (?,?,\bigstar,?)$, $\texttt{AbstScore}^* = 1$

$(z,\bigstar,\blacksquare) \triangleright (y,\spadesuit,\blacksquare)$, $p^* = (1,?,?,?)$, $\texttt{AbstScore}^* = 1$

$(\blacksquare) \triangleright (\blacksquare) \triangleright (\blacksquare)$, $p^* = (1,y,\spadesuit,?)$, $\texttt{AbstScore}^* = 3$

(b) Segmentation-based abstract-skill extraction ($w = 0.8$).

ARMs of "skeleton" skill:

$(x,\bigstar,\blacksquare) \triangleright (x,\bigstar,\blacksquare) \triangleright (z,\bigstar,\blacksquare) \triangleright (y,\spadesuit,\blacksquare) \triangleright (y,\spadesuit,\blacksquare)$, $p^* = (1,?,?,?)$, $\texttt{AbstScore}^* = 1$

ARMs of "segment" skills:

$(1,\blacksquare) \triangleright (2,\blacksquare) \triangleright (1,\blacksquare)$, $p^* = (?,x,\bigstar,?)$, $\texttt{AbstScore}^* = 2$

$(1,x) \triangleright (2,y) \triangleright (1,z)$, $p^* = (?,?,\bigstar,\blacksquare)$, $\texttt{AbstScore}^* = 2$

$(z,\bigstar,\blacksquare) \triangleright (y,\spadesuit,\blacksquare)$, $p^* = (1,?,?,?)$, $\texttt{AbstScore}^* = 1$

$(\blacksquare) \triangleright (\blacksquare) \triangleright (\blacksquare)$, $p^* = (1,y,\spadesuit,?)$, $\texttt{AbstScore}^* = 3$

(c) Segmentation-based abstract-skill extraction ($w = 0.2$).

Fig. 4: Extraction of multiple abstract-skills from a single execution in a symbolic domain, using the "attention" abstraction key.

VI. Conclusion

This paper addressed the problem of extracting generalizable, abstract skills from successful task plan executions. This has been, up until this point, a practical gap in end-to-end automation of the skill transfer framework we established in previous work. We started by analyzing the desired qualities of an extracted skill. With the conclusions from this analysis, we were able to define the "abstract skill score" and formulate skill extraction as an optimization problem. We then provided an automatable algorithm for solving this problem, via detection of abstraction-critical states—states which bound segments of an execution from which we can extract skills with maximal score. We provided two variations of the formulation and algorithm, considering a single-skill extraction and full-segmentation-based multi-skill extraction, and demonstrated their differences experimentally. We also examined the effect of the quality preference weight, which allowed us to yield "more instructive" or "more applicable" skills. Evidently, the approach is useful in various planning domains, including symbolic domains (as used in our experiments) and geometric domains (as shown in Fig. 1). Finally, although the algorithms are clearly effective, we recognize that they might not achieve the globally-optimal skills. We plan to mitigate this aspect in future work by, e.g., considering non-myopic or stochastic hill-climbing.

TABLE I: Comparison of abstract-skill extraction techniques. Average results for 100 random symbolic traces of length 30.

w	*Algorithm*	$\texttt{len}(ARM)$	$\texttt{AbstScore}^*$	*# of skills*
$+.5$	Single skill	4.7	0.2	1 segment
	Segmentat.	2.6 (per seg.)	0.31 (per seg.)	21.6, skeleton
$-.5$	Single skill	2.2	0.8	1 segment
	Segmentat.	2.0 (per seg.)	0.4 (per seg.)	14.2, skeleton

REFERENCES

[1] Erez Karpas and Daniele Magazzeni. Automated planning for robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):417–439, 2020.

[2] Jean-Claude Latombe. *Robot motion planning*, volume 124 of *The Springer International Series in Engineering and Computer Science*. Springer New York, NY, 1991.

[3] Khen Elimelech, Lydia E. Kavraki, and Moshe Y. Vardi. Automatic cross-domain task plan transfer by caching abstract skills. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 06 2022.

[4] Khen Elimelech, Lydia E. Kavraki, and Moshe Y. Vardi. Efficient task planning using abstract skills and dynamic road map matching. In *International Symposium on Robotics Research (ISRR)*, 09 2022.

[5] Zuyuan Zhu and Huosheng Hu. Robot learning from demonstration in robotic assembly: A survey. *Robotics*, 7(2), 2018.

[6] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

[7] Peter Pastor, Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, and Stefan Schaal. Skill learning and task outcome prediction for manipulation. In *2011 IEEE International Conference on Robotics and Automation*, pages 3828–3834, 2011.

[8] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[9] 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(1):265–293, 2021.

[10] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477, 2011.

[11] William Vega-Brown and Nicholas Roy. Admissible abstractions for near-optimal task and motion planning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, page 4852–4859. AAAI Press, 2018.

[12] Danny Driess, Ozgur Oguz, and Marc Toussaint. Hierarchical task and motion planning using logic-geometric programming (HLGP). RSS Workshop on Robust Task and Motion Planning, 2019.

[13] Margret Keuper, Bjoern Andres, and Thomas Brox. Motion trajectory segmentation via minimum cost multicuts. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[14] Amin Hosseinpoor Milaghardan, Rahim Ali Abbaspour, and Christophe Claramunt. A geometric framework for detection of critical points in a trajectory using convex hulls. *ISPRS International Journal of Geo-Information*, 7(1), 2018.