Learning Graph-based Representations for Continuous Reinforcement Learning Domains

Jan Hendrik Metzen

Robotics Group, University of Bremen
Robert-Hooke-Straße 5, 28359 Bremen, Germany
jhm@informatik.uni-bremen.de

Abstract. Graph-based domain representations have been used in discrete reinforcement learning domains as basis for, e.g., autonomous skill discovery and representation learning. These abilities are also highly relevant for learning in domains which have structured, continuous state spaces as they allow to decompose complex problems into simpler ones and reduce the burden of hand-engineering features. However, since graphs are inherently discrete structures, the extension of these approaches to continuous domains is not straight-forward. We argue that graphs should be seen as discrete, generative models of continuous domains. Based on this intuition, we define the likelihood of a graph for a given set of observed state transitions and derive a heuristic method entitled FIGE that allows to learn graph-based representations of continuous domains with large likelihood. Based on FIGE, we present a new skill discovery approach for continuous domains. Furthermore, we show that the learning of representations can be considerably improved by using FIGE.

1 Introduction

Reinforcement Learning (RL) allows autonomous agents to learn to improve their performance with experience in an unknown environment. However, typically representations for policies and value functions need to be carefully hand-engineered for the specific domain and learned knowledge is not efficiently reused in situations when an agent has to solve several different but related tasks. Representation learning for RL [7] and hierarchical RL [1] are approaches to alleviate these drawbacks.

Graph-based representations of the domain have been used as basis for both representation learning and hierarchical RL. For instance, Mahadevan and Maggioni [7] have learned useful internal representations called proto-value functions based on a graph representation of the domain. Such graphs have also been used to identify bottleneck states of the environment [8, 10, 13] which are a common basis for skill discovery in hierarchical RL. While these graph-based approaches have shown promising results in domains with discrete state and action spaces, extending them to continuous domains is not straight-forward. This is mainly due to the fact that graphs are intrinsically discrete structures and thus cannot directly model a continuous environment.

Previous work on graph-based approaches in continuous domains has thus either discretized the domain, i.e., placed graph nodes at a regular grid over the state space

[8], or placed graph nodes at a subset of the observed states [7]. While the former suffers from the curse-of-dimensionality, the later allows to exploit situations where the effective dimensionality of the state space is smaller. However, both approaches focus purely on covering the state space as uniformly as possible and neglect the dynamics of the environment. We argue that graph representations should take the dynamics into account since they can be seen as a model of the environment. That is, typical transitions encountered in the domain should be representable by the graph. The hypothesis evaluated in this paper is that a graph, which models the dynamics of its (continuous) environment well, will yield superior results with regard to representation learning and bottleneck identification. We propose a new heuristic called FIGE which allows to learn graph representations that explicitly aim at modeling the environment's dynamics.

The outline of the paper is as follows: In Section 2, we review graph-based RL methods and discuss how graph representations have been generated in these methods. In Section 3, we define the likelihood of a graph for a given set of transitions sampled according to the domain's dynamics. Thereupon, we propose the FIGE heuristic for learning graph representations of continuous environments, which is derived from the maximum graph likelihood formulation under simplifying assumptions. Furthermore, we compare FIGE with other graph learning heuristics empirically with regard to the graph likelihood. In Section 4, we propose and evaluate a new graph-based skill discovery method for continuous domains, which is based on the FIGE heuristic. Similarly, in Section 5, we present empirical evidence that Representation Policy Iteration [7] can benefit from using FIGE for graph generation in continuous domains. We summarize the results of this paper and provide an outlook in Section 6.

2 Graph-based Reinforcement Learning

A Markov decision process (MDP) M can be formalized as a 4-tuple $M = (S, A, P^a_{ss'}, R^a_{ss'})$ where S is a set of states, A is a set of actions, $P^a_{ss'} = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the 1-step state transition probability also referred to as the "dynamics", and $R^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$ is the expected reward. In RL, these quantities are usually unknown to the agent but can be estimated based on samples collected during exploration. If both S and A are finite, we call M a discrete MDP, otherwise we call it a continuous MDP. If for all $s \in S, a \in A$ there exists one $s' \in S$ with $P^a_{ss'} = 1$, the MDP is called deterministic otherwise it is a stochastic MDP. The goal of RL is to learn without explicit knowledge of M a policy π^* such that some measure of the long-term reward is maximized. Learning is often based on approximating the optimal action-value function $Q^*(s,a) = \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma \max_{a'} Q^*(s', a')]$, where $\gamma \in [0,1]$ is a discount factor.

One way of representing a (finite) MDP is using a weighted labeled multigraph $G = (V, E, w)$. Assuming knowledge of M, we would set $V = S$ and $E = \{(s,s')_a | \forall s, s' \in S, a \in A : P^a_{ss'} > 0\}$. In a state transition graph [7, 10], edge weights encode transition probabilities between nodes. If we set the weight of edge $(s,s')_a$ to $w^a_{ss'} = P^a_{ss'}$, a state transition graph is just an other way of representing the domain's dynamics. However, the graph-based view of the MDP is particularly suited for representation learning [7] and for identifying bottlenecks of the MDP [8–10, 12, 13], which is a common prerequisite for skill discovery in hierarchical RL. In small and discrete domains, learning state transition graphs from experience for unknown MDPs is straightforward: one graph node is created for each observed domain state and an edge is created for any pair of states between which a direct transition has been observed. For domains with continuous state spaces $S \subset \mathbb{R}^d$, the situation is more complicated because graphs are inherently discrete structures and thus, there cannot be a 1-to-1 correspondence between states and graph nodes since there exists an infinite number of states. Thus, several states need to be aggregated into one node, i.e., $V \subsetneq S$. Accordingly, one has to choose how many nodes there should be and where in the state space these nodes should be placed.

Prior work on choosing the positions of the graph nodes has mainly focused on covering the state space uniformly with nodes and neglected the domain's dynamics $P^a_{ss'}$. Among these approaches are: (a) a heuristic which forms a uniform *grid* of v_{num} nodes over the state space with a grid resolution of $\lfloor \sqrt[d]{v_{num}} \rfloor$ per dimensions. This approach has been used in the context of graph-based skill discovery, e.g., by Mannor et al. [8]. An obvious disadvantage is that the approach will not scale to domains with many dimensions. (b) The *on-policy sampling* heuristic (also denoted as "random subsampling" by Mahadevan and Maggioni [7]), which samples v_{num} graph nodes uniform randomly from the set of states S' encountered during exploration. The heuristic is on-policy, i.e., regions of the state space that are often visited by the sampling policy are represented by more graph nodes. (c) The *ε-net* heuristic, also denoted as "trajectory-based subsampling" [7], which aims at covering the "effective state space" as uniformly as possible. It follows a greedy strategy for finding a locally maximal set of graph nodes $V \subset S'$ with pairwise distance at least ε. The advantage of this approach compared to the on-policy sampling method is that the effective state space is covered more uniformly. In order to parametrize the heuristic by v_{num} instead of ε, one can perform binary search for a value of ε that yields a set of graph nodes with cardinality v_{num}.

3 FIGE: Force-based Iterative Graph Estimation

While the heuristics discussed in Section 2 focus on covering the state space uniformly, they do not take the domain's dynamics into account. Thus, for many valid state transitions $s \rightarrow s'$ of the domain, there may not be any pair of graph nodes $v_1, v_2 \in V$ such that $v_1 \rightarrow v_2$ is a good representation of $s \rightarrow s'$. Accordingly, the graph may not be able to capture the domain's dynamics $P^a_{ss'}$ accurately. In this section, we propose a generative process which defines how probable a set of observed transition has been generated from a transition graph. We then propose the heuristic FIGE which is derived from this generative process as maximum likelihood solution under simplifying assumptions.

3.1 Likelihood of Transition Graph

We propose to consider a graph as a generative model for transitions and to choose graph node positions such that the likelihood $p(T|G)$ of the resulting graph G for a set of observed transitions $T = \{(s_i, a_i, s'_i)\}^n_{i=1}$ becomes maximal. We consider transitions to have been sampled from the graph using the following generative process: (1) Sample a graph node $v \in V$ uniform randomly, i.e., $p(v) = 1/|V|$. (2) Sample a state s for a given node v according to $p(s|v) = N_b \exp(-\frac{1}{b^2} \|s - v\|^2_2)$ where b controls how closely centered $p(s|v)$ is on v and N_b is a normalization constant, which only depends on b. (3) Sample an action a uniformly from the action space A, i.e, $p(a|s) = p(a) = 1/|A|$. (4) Sample the "successor node" v' according the the graph's edge weights, i.e., $p(v'|v,a) = w^a_{vv'}$. (5) Finally, sample the successor state s' according to the distribution $p(s'|v',v,s) = N_b \exp(-\frac{1}{b^2}||s' - (v' - v + s)||_2^2)$ with the same b and N_b as before. This distribution encourages that the state transition $s \rightarrow s'$ is close to parallel to the given node transition $v \rightarrow v'$. This 5-step generative process can be derived as follows:

$$p(T|G) = \prod_{i=1}^{n} p((s_i, a_i, s'_i)|G) = \prod_{i=1}^{n} p_G(s_i) p(a_i|s_i) p_G(s'_i|s_i, a_i)$$

Under the independence assumptions $I = \{v \perp a|s; v' \perp s|v, a; s' \perp a|v', v', s\}$, we have

$$\begin{aligned} p_G(s'|s,a) &= \sum_v p_G(v, s'|s,a) = \sum_v p_G(v|s,a) p_G(s'|v,s,a) \\ &= \sum_v p_G(v|s,a) \sum_{v'} p_G(v', s'|v,s,a) = \sum_v p_G(v|s,a) \sum_{v'} p_G(v'|v,s,a) p_G(s'|v',v,s,a) \\ &\overset{I}{=} \sum_v p_G(v|s) \sum_{v'} p(v'|v,a) p_G(s'|v',v,s) \end{aligned}$$

Inserting this in $p(T|G)$ and using Bayes rule $p_G(v|s) = p(s|v)p(v)/p_G(s)$ yields

$$\begin{aligned} p(T|G) &= \prod_{i=1}^{n} p_G(s_i) p(a_i|s_i) \left[\sum_{v \in V} \frac{p(s_i|v)p(v)}{p_G(s_i)} \sum_{v' \in V} p(v'|v,a) p(s'_i|v',v,s_i) \right] \\ &= \prod_{i=1}^{n} \underbrace{p(a_i|s_i)}_{(3)} \left[\sum_{v \in V} \underbrace{p(v)}_{(1)} \underbrace{p(s_i|v)}_{(2)} \sum_{v' \in V} \underbrace{p(v'|v,a)}_{(4)} \underbrace{p(s'_i|v',v,s_i)}_{(5)} \right]. \end{aligned}$$

3.2 Method

Given the generative process discussed in the previous section, the optimal state transition graph for a given set of transitions T would be $G^* = \arg\max_G p(T|G)$. Unfortunately, solving this problem directly is hard; we propose the FIGE heuristic, which aims at finding close-to-optimal transition graphs iteratively and is computationally tractable. FIGE's update equations are derived from the maximum likelihood objective $G^* = \arg\max_G p(T|G)$ using two simplifying assumptions (see Appendix A): ($A1$) For each transition $(s,a,s') \in T$, assume $p(v'|v,a) = 1$ if $v = \mathrm{NN}_V(s) \wedge v' = \mathrm{NN}_V(s')$ else 0, where $\mathrm{NN}_V(s) = \arg\min_{v \in V} ||s - v||^2$. This assumption implies that whenever action a is executed in any state of the Voronoi cell $Vo(v) = \{s \in S | \mathrm{NN}_V(s) = v\}$ the successor state will be with probability 1 in $Vo(v')$. ($A2$) Assume $p(T|V) = \prod_v p(T|v)$. This assumption implies that the choice of the positions of the graph nodes $v \in V$ can be made independently. Both assumptions are typically oversimplifying; $A1$ is more oversimplifying for domains whose dynamics are less locally smooth. $A2$ on the other hand is more simplifying in strongly connected domains where many transitions from the Voronoi cell of one node to the Voronoi cell of another node occur. To account for some

Algorithm 1 Force-based Iterative Graph Estimation (FIGE)

```
1: Input: Transitions $T = \{(s_i, a_i, s'_i)\}_{i=1}^{n}$, parameters $v_{num}$, $K$
2: # Choose initial node positions $V$ from states in $T$ s.t. distance of closest pair is maximized
3: $V = \textsc{Initialize}(T, v_{num})$
4: for $i = 0$ to $K-1$ do
5:   for all $v \in V$ do
6:     $S^V[v] = \{s \mid \exists(s,a,s') \in T : \mathrm{NN}_V(s) = v\}$ # Observed states in Voronoi cell $Vo(v)$
7:     $F_S[v] = \textsc{Mean}(S^V[v]) - v$ # Sample representation force
8:     $T^{\rightarrow}(v) = \{\mathrm{NN}_V(s') - s' + s \mid \exists(s,a,s') \in T : \mathrm{NN}_V(s) = v\}$ # Transitions starting in $Vo(v)$
9:     $T^{\leftarrow}(v) = \{\mathrm{NN}_V(s) - s + s' \mid \exists(s,a,s') \in T : \mathrm{NN}_V(s') = v\}$ # Transitions ending in $Vo(v)$
10:    $F_G[v] = 0.5 \cdot [\textsc{Mean}(T^{\rightarrow}(v)) + \textsc{Mean}(T^{\leftarrow}(v))] - v$ # Graph consistency force
11:  $V = V + \alpha_i \cdot 0.5(F_S[V] + F_G[V])$ # Update node positions (vector notation)
12: # Count transitions from Voronoi cell $Vo(v)$ to Voronoi cell $Vo(v')$ under action $a$
13: $N^a_{vv'} = |\{(s,s') \mid \exists\,(s,a,s') \in T : \mathrm{NN}_V(s) = v \wedge \mathrm{NN}_V(s') = v'\}|$
14: $E = \left\{(v,v')_a \mid v,v' \in V\ a \in A : N^a_{vv'} > 0\right\}$ # Edge between $v$ and $v'$ labeled with action $a$
15: $w^a_{vv'} = N^a_{vv'} / \sum_{\tilde{v}} N^a_{v\tilde{v}}$ # Edge weights are empirical transition probabilities on graph
16: return $(V, E, w)$
```

of the errors made because of the oversimplifications of $A1$ and $A2$, FIGE iteratively refines the graph node positions by applying the derived update equations several times. Note that FIGE is a heuristic and no guarantee for converging to G^* is given.

FIGE is summarized in Algorithm 1: The set of graph nodes V is initialized such that it covers the set of states contained in T uniformly by, e.g., maximizing the distance of the closest pair of graph nodes (line 3). Afterwards, for K iterations, the graph nodes are moved according to two kind of "forces" that act on them: The "sample representation" force (line 6-7) pulls each graph node v to the mean of all states S^v for which it is responsible, i.e., the states s for which it is the nearest neighbor $\mathrm{NN}_V(s)$ in V. Thus, this force encourages node positions that capture the on-policy state distribution well and corresponds to an intrinsic k-means clustering. The "graph consistency" force (line 8-10) pulls each graph node v to a position where for all $(s,a,s') \in T$ with $\mathrm{NN}_V(s) = v$ there is a vertex v' such that $v' - v$ is similar to $s' - s$, i.e., both vectors are close to parallel. Thus, this force encourages node positions which can represent the domain's dynamics well. The nodes are then moved according to the two forces (line 11), where the parameter $\alpha_i \in (0,1]$ controls how greedily the node is moved to the position where the forces would become minimal. In order to ensure convergence of the graph nodes, α_i should go to 0 for i approaching K. If not explicitly stated, we use $\alpha_i = \lceil i/5 \rceil^{-1}$ and $K = 15$. An edge labeled with action a is added between two nodes v and v' if there exists at least one transition $(s,a,s') \in T$ with s being in the Voronoi cell of $Vo(v)$ and s in $Vo(v')$ (line 14). Furthermore, the edge weights are chosen as the empirical transition probabilities $\hat{P}^a_{vv'}$ from node v to v' under action a (line 15). It has recently been shown that this choice of edge weights is most robust under varying degrees of domain stochasticity and different exploration strategies of the agent [11].

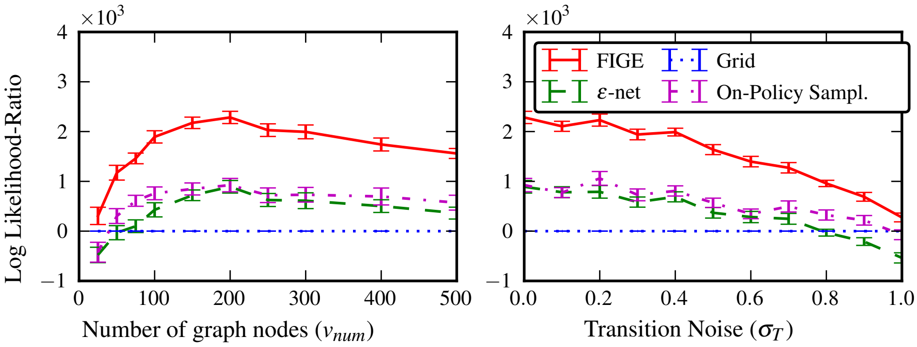

Fig. 1. Left graph: Graph Log Likelihood-Ratio relative to grid heuristic in deterministic mountain car for different values for the number of graph nodes. Right graph: Graph Log Likelihood-Ratio relative to grid heuristic in the stochastic mountain car domain for varying stochasticity σ_t and $v_{num} = 200$. Shown are mean and standard error of mean over 20 repetitions.

3.3 Evaluation: Graph Likelihood

In this section, we present an empirical comparison of the different heuristics with regard to the obtained likelihood of the generated graphs in the mountain car domain (compare Chapter 8.4 of Sutton and Barto [15]). In the left graph of Figure 1, the graph likelihood with $b = 0.02$ is evaluated for different heuristics and different number of graph nodes v_{num}. Since the likelihood depends on v_{num}, we plot the log of the ratio of the method's likelihood relative to the likelihood obtained by the data-independent grid heuristic. Note that the graph likelihood has been evaluated on test transitions T_{test} which were different from the training transitions T_{train} that were used for the optimization of graph node positions ($|T_{train}| = 5000$, and $|T_{test}| = 10^5$).

Regardless of v_{num}, the largest graph likelihood is achieved by FIGE and the worst one by the grid heuristic (for $v_{num} > 75$). The on-policy sampling heuristic performs slightly better than ε-net for $v_{num} < 150$ while both perform similarly for larger v_{num}. A possible explanation for the stronger deterioration of ε-net is that for small v_{num}, the minimal node distance ε gets larger than the typical distance of states and their successors and thus, $P^a_{ss'}$ cannot be represented in any part of the state space. In contrast, on-policy sampling allocates more graph nodes in densely sampled parts of the state space and thus allows to model at least these parts of the state space. FIGE can achieve a considerably larger graph likelihood than both by taking the domain's dynamics into account as well.

In a second experiment, we evaluate how robust the different heuristics are with regard to increasing stochasticity in the domain's state transition probability $P^a_{ss'}$. For this, we modify any transition from state s to s' governed by the domain's deterministic dynamics such that the i-the dimension of the actual successor state becomes $s'_i \leftarrow s'_i + \sigma_i(s'_i - s_i)$ with σ_i sampled uniformly from $[-\sigma_t, \sigma_t]$. Note that this is not purely observation noise since the actual internal state of the environment is altered. σ_t controls how "strong" the stochasticity of the domain is, with $\sigma_t = 0$ corresponding to the deterministic domain. The same amount of transition noise was also used for generating the test transitions T_{test}. The results are shown in the right graph in Figure 1. As expected, the grid-based heuristic deteriorates less with increasing stochasticity as it does not take the observed transitions into account. Nevertheless, the other data-dependent heuristics achieve better graph likelihood for $\sigma_t < 0.8$ with FIGE remaining the best heuristic for the whole investigated range of $\sigma_t \in [0,1]$. This shows that FIGE is also suited for stochastic domains.

4 Graph-based Skill Discovery

Next to continuous action spaces, scaling RL to real-world problems with large or continuous state spaces remains a challenge since the amount of experience the agent can collect is limited. One approach to alleviate this problem is hierarchical RL [1], which aims at dividing a problem into simpler subproblems, learning solutions for these subproblems, and encapsulate the acquired knowledge into so-called *skills* that can potentially be reused later on in the learning process. It has been shown that skills can help an agent to adapt to non-stationarity of the environment and to transfer knowledge between different but related tasks [3] and can increase the representability of the value function in continuous domains [5]. A major challenge in hierarchical RL is to identify what might constitute a useful skill, i.e., how the problem should be decomposed. Skills should be reusable, distinct, and easy to learn. The task of identifying such skills is called *skill discovery* [2, 4].

Most prior work on autonomous skill discovery is based on the concept of *bottleneck areas* in the state space. Informally, bottleneck areas have been described as the border states of densely connected areas in the state space [10] or as states that allow transitions to a different part of the environment [12]. Several heuristics have been proposed to identify bottlenecks. One class of heuristics are *frequency-based approaches* that compute local statistics of states like diverse density [9] and relative novelty [12]. An other class of heuristics that is typically more sample-efficient are *graph-based approaches* which are based on estimates of the domain's state transition graph (see Section 2). Graph-based approaches to skill discovery aim at partitioning this graph into subgraphs which are densely connected internally but only weakly connected with each other. Menache et al. [10] propose a top-down approach for partitioning the global transition graph based on the max-flow/min-cut heuristic. Şimşek et al. [13] follow a similar approach but partition local estimates of the global transition graph using a spectral clustering algorithm and use repeated sampling for identifying globally consistent bottlenecks. Mannor et al. [8] propose a bottom-up approach that partitions the global transition graph using agglomerative hierarchical clustering.

Relatively few works on autonomous skill discovery in domains with continuous state spaces exist. Mannor et al. [8] have evaluated their approach in the mountain car domain by uniformly discretizing the state space. However, such a uniform discretization is suboptimal since it does not scale well to higher dimensional state spaces. One skill discovery method that has been designed for continuous domains is "skill chaining" [5]. Skill chaining produces chains (or more general: trees) of skills such that each skill allows to reach a specific region of the state space, such as a terminal region or a region where an other skill can be invoked. Skill chaining requires to specify an area of interest (typically the terminal region of the state space) which is used as target for the skill at the root of the tree. For multi-task domains with several goal regions, it is unclear how the root of the skill tree should be chosen. In the next section, we present a generic algorithm for graph-based skill discovery in MDPs with continuous state spaces.

4.1 Approach: Skill Discovery by Clustering of Transition Graph

We adopt the *option framework* [14] for Hierarchical RL, in which skills are formalized as *options*: An option o consists of three components: the option's initiation set $I_o \subset S$ that defines the states in which the option may be invoked, the option's termination condition $\beta_o : S \rightarrow [0,1]$ which specifies the probability of option execution terminating in a given state, and the option's policy π_o which defines the probability of executing an action in a state under option o. In the options framework, the agent's policy π may in any state s decide not solely to execute a primitive action but also to call any of the options for which $s \in I_o$. If an option is invoked, the option's policy π_o is followed for several time steps until the option terminates according to β_o. The option's policy π_o is defined relative to an option-specific reward function R_o that may differ from the global external reward function. Skill discovery thus requires to choose I_o, β_o, and R_o.

For a given set of observed transitions T that have been sampled from the MDP, we can generate the state transition graph using any of the approaches discussed in this paper. Based on the generated transition graph, any of the graph-based skill discovery approaches for discrete MDPs could be used to identify skills. We adopt the concept of identifying densely connected subgraphs of the transition graph, which correspond to densely connected regions in the state space. In order to quantify to what extent the edges connecting two disjoint subgraphs form a bottleneck, a so-called *linkage criterion* is used. A linkage criterion is a function mapping two disjoint subgraphs $A, B \subset G$ onto a scalar, which is the larger the "stronger" the bottleneck between A and B in G is. In this paper, we adopt the off-policy $\hat{N}_{cut}$ linkage criterion that was proposed by Metzen [11]. The $\hat{N}_{cut}$ criterion is an approximation of the sum of probabilities that the agent transitions in one time step from a state in subgraph A to a state in subgraph B or vice versa. For identifying densely connected subgraphs of a graph G, we aim to determine a partition C^* of minimal cardinality of the graph nodes into disjoint sets c_i such that each induced subgraph does not contain a bottleneck. Formally: $C^* = \underset{C \in \mathscr{C}(V)}{\arg\min} |C|$ s.t. $\max_{c_i \in C, d_i \subset c_i} \hat{N}_{cut}(c_i \setminus d_i, d_i) \leq \psi$, with $\mathscr{C}(V)$ being the set of all possible partitions of V and ψ a parameter controlling the granularity of the clustering. Note that the maximization goes over all possible ways of splitting c_i into two parts d_i and $c_i \setminus d_i$ and the constraint guarantees that there is no bottleneck with $\hat{N}_{cut} > \psi$ in any of the c_i. Since finding the optimal solution for this problem is NP-hard, we use an approximate approach that is based on agglomerative hierarchical clustering and similar to the one proposed by Mannor et al. [8]. This algorithm starts by assigning each node into a separate cluster and afterwards merges greedily the pair of clusters with minimal linkage until no pair remains with a linkage below ψ. As proposed by Mannor et al., only clusters which are connected in G can be merged.

For learning an option o based on a newly discovered skill, we need to choose appropriate I_o, β_o, and R_o based on the identified partitioning C_G of the transition graph.

For this, the partition C_G of the transition graph is generalized to a partition of the entire state space C_S by a nearest-neighbor based approach, i.e., for all clusters $c_i \in C_G$: $C_S(c_i) = \{s \in S \mid \mathrm{NN}_V(s) \in c_i\}$. For each cluster A, one skill is generated for each adjacent cluster B, where A and B are adjacent if there exists $v_a \in A, v_b \in B$ and action a such that $(v_a, v_b)_a \in E$. The corresponding skill prototype $(I_{A\to B}, \beta_{A\to B}, R_{A\to B})$ is defined as:

$$I_{A\to B} = C_S(A) \qquad \beta_{A\to B}(s) = 0 \text{ if } s \in I_{A\to B} \text{ else } 1$$
$$R_{A\to B}((s,a,r,s')) = -1 \text{ if } s' \in (C_S(A) \cup C_S(B)) \text{ else } r_p,$$

where r_p is a parameter of the algorithm that determines the penalty for failing to fulfill a skill's objective. Additionally, for each cluster that contains nodes in which an episode has terminated, a special skill is created that can be invoked in any state of the cluster, terminates successfully when the episode terminates, and terminates unsuccessfully (i.e., obtains the penalty reward) if the clusters is left. Note that in contrast to Mannor et al. [8], the generalization of the graph partition to the entire state space allows to perform the learning of skills and higher-level policies in the original MDP and not in a discretized version of it.

4.2 Evaluation

In this section, we present an empirical evaluation of the proposed skill discovery approach in the 2D Multi-Valley environment, which is an extension of the basic mountain car domain. The car the agent controls is not restrained to a one-dimensional surface, however, but to a two-dimensional surface. This two-dimensional surface consists of $2 \times 2 = 4$ valleys, whose borders are at $(\pi/6 \pm \pi/3, \pi/6 \pm \pi/3)$. The agent observes four continuous state variables: the positions in the two dimensions (x and y) and the two corresponding velocities (v_x and v_y). The agent can choose among the four discrete actions `northwest`, `northeast`, `southwest`, `southeast` which add $(\pm 0.001, \pm 0.001)$ to (v_x, v_y). In each time step, due to gravity $0.004\cos(3x)$ is added to v_x and $0.004\cos(3y)$ to v_y. The maximal absolute velocity in each dimension is restrained to 0.07. The agent is faced with a multi-task scenario: in each episode, the agent has to solve one out of twelve tasks. Each task is associated with a combination of two distinct valleys; e.g., in task $(0,1)$ the agent starts in the floor[1] of valley 0 and has to navigate to the floor of valley 1 and reduce its velocity such that $\left|\left|(v_x, v_y)\right|\right|_2 \leq 0.03$. In each time step, the agent receives a reward of $r = -1$. Once a task is solved, the next episode starts with the car remaining at its current position and one of the tasks that starts in this valley is drawn at random. The current task is communicated as an additional state space dimension to the agent; the agent uses it for learning the top-level policy but ignores it during graph generation, graph clustering, and skill learning such that skills are reusable in different tasks.

We present an empirical comparison of the learning performance of the entire hierarchical RL architecture with different graph node selection heuristics as base for skill discovery. We compare the performance to two baselines: (i) a monolithic approach which learns a flat policy for every task without using skills, and (ii) the same hierarchical RL framework but with predefined skills prototype (I_o, β_o, R_o). These prototypes

[1] The floor of valley 0 corresponds to the region $((-1/6 \pm 2/15)\pi, (-1/6 \pm 2/15)\pi)$.

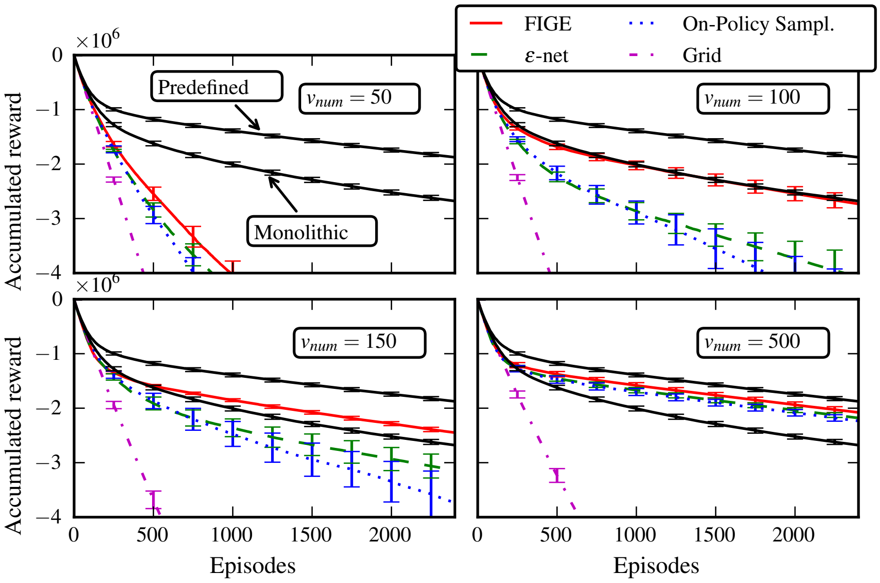

Fig. 2. Accumulated reward in the 2D Multi-Valley domain during 2400 episodes of learning for different graph generation heuristics and graph sizes. Baselines show performance of a monolithic learner and for the optimal predefined skill prototypes. Shown are mean and standard error of mean over 10 repetitions.

have been generated in the same way as those discovered using graph clustering but are based on the ground-truth partition of the domain into the 4 valleys. Thus, baseline (ii) presents probably an upper boundary for the performance that any skill discovery method can achieve within the given hierarchical RL architecture.

Skill discovery has been performed after $n = 10^5$ state transitions have been observed in the environment and graphs with $v_{num} \in \{50, 100, 150, 500\}$ nodes have been generated. These have been clustered with the approach presented in Section 4.1 for $\psi = -0.03$. Each option's value function has been represented by an CMAC function approximator consisting of 10 independent tilings with $7^2 \cdot 5^2$ tiles, where the higher resolutions have been used for the x and y dimensions. The penalty reward of an unsuccessful option has been set to $r_p = -1000$ and the value functions have been initialized to -100. For learning the compositional option π, a lower resolution of $5^2 \cdot 3^2$ tiles has been used and the value functions have been initialized to -1000. The discounting factor has been set to $\gamma = 1$ and all policies have been ε-greedy with $\varepsilon = 0.01$. The value functions were learned using Q-Learning and updated only for currently active options with a learning rate of 1. Episodes have been interrupted after 10^4 steps without solving the task and a new task was chosen at random. All parameters have been chosen based on preliminary investigations.

Figure 2 shows the accumulated reward obtained during the first 2400 learning episodes (the phase of learning during which the explorative bias provided by skills has the strongest impact) for different graph node selection heuristics and different number of graph nodes v_{num}. For too small v_{num}, e.g., $v_{num} = 50$, all graph node selection heuristic perform worse than learning a flat policy. Furthermore, one can see that the grid-based heuristic obtains poor results for any choice of v_{num}. When using many graph nodes ($v_{num} = 500$), no considerable differences between the other heuristics exist and the performance is considerably better than when learning a flat policy and only slightly worse than for predefined skills. However, for intermediate values of v_{num}, e.g., $v_{num} \in \{100, 150\}$, FIGE obtains significantly better results than the ε-net and the on-policy sampling heuristic ($p < 0.001$, Mann-Whitney U-test). Furthermore, the accumulated reward obtained by FIGE for $v_{num} = 150$ is considerably larger than when learning a flat policy, which is not the case for the other heuristics. In summary, FIGE allows to create skills that can provide a useful explorative bias during learning based on smaller graphs than other heuristics which allows to reduce computation time during graph clustering and learning.

5 Representation Learning

Representation Policy Iteration (RPI) is an approach proposed by Mahadevan and Maggioni [7] that aims at solving MDPs by jointly learning representations and optimal policies. In contrast to most other RL algorithms, RPI does not require an a-priori specification of basis functions. The main idea for learning basis functions is to first learn a state transition graph of the domain and to construct a symmetric diffusion operator on this graph. The normalized graph Laplacian $\mathscr{L} = D^{-1/2}(D - W)D^{-1/2}$ and the combinatorial Laplacian $L = D - W$ are examples for such diffusion operators—with W being the graph's symmetrized weight matrix and D being a diagonal matrix whose entries are the row sums of W. The smoothest eigenvectors of these operators, the so-called proto-value functions, are used as basis functions for representing value functions. Least-squares policy iteration (LSPI) as proposed by Lagoudakis and Parr [6] is used for control learning, i.e., for learning the parameters w^{π} of the action value function $Q^{\pi} = w^{\pi}\Phi$ of an ε-optimal policy π within the linear span of the basis functions Φ. In the original paper, at the end of each episode an additional set of samples is collected either on- or off-policy. We skip this additional sampling and use the samples collected during control learning also for representation learning to show that some graph node selection heuristics can deal with this better than others. In order to initialize representation and control learning, the agent explored the environment uniform randomly during the first 10 episodes. Thereupon, RPI was performed at the end of each episode and the policy obtained was followed ε-greedily.

RPI can also be used in MDPs with continuous state space. In such continuous domains, one challenge is to select the graph node position ("to subsample a set of states" in the terminology of Mahadevan and Maggioni). The authors discuss the usage of the on-policy subsampling and the ε-net heuristics; however we will show that considerable better results can be achieved by using FIGE. In RPI, each graph node is connected to its k nearest neighbor nodes in the euclidean space and the edge weight between nodes

x_i and x_j is $W(i,j) = \tau(i)\exp(-\|x_i - x_j\|_2^2/\kappa)$ where $\tau(i)$ and κ are parameters to be specified. Note that this way of connecting graph nodes has the potential disadvantage that proximity of nodes in the euclidean space does not necessarily imply that a transition between these nodes is possible, e.g., if an obstacle lies between those states. Choosing graph edges based on observed transitions between states (compare Section 2) lessens this issue and seems thus preferable. However, for consistency with the original approach of Mahadevan and Maggioni [7], we adhere the "euclidean" connectivity.

5.1 Evaluation: Mountain Car

In a first experiment, we evaluate the performance of RPI for different graph node selection heuristics and different degrees of transition noise σ_t in mountain car (compare Section 3.3) for $v_{num} = 50$. For all node selection heuristics, we obtained the best results when setting the number of proto value-functions equal to the number of graph nodes, i.e., $p_{num} = v_{num}$. Furthermore, in accordance with Mahadevan and Maggioni [7] we set the discount factor γ to 0.99 and the exploration rate to $\varepsilon = 0.01$. We used the normalized Laplacian $\mathscr{L}$ as graph operator. The results are shown in Figure 3. The left plot shows learning curves of RPI in the deterministic mountain car domain: RPI performs best when combined with FIGE and worst when combined with the grid heuristic while on-policy sampling and ε-net achieve approximately the same results (no significant differences). This can be attributed to the fact that due to the randomly chosen start states of episodes, the on-policy state distribution (over several episodes) does not vary too strongly over the effective state space. Thus, sampling from the on-policy distribution yields in this domain graph nodes that cover the effective state space close to uniform. The worse results of the grid heuristic show that even for low-dimensional domains, a uniform discretization can be detrimental. The right plot shows how the reward accumulated after 100 episodes changes for different degrees of stochasticity of the domain. In general, increasing transition noise seems to make the task easier as the performance increase for all heuristics; probably because the value function becomes smoother and thus better representable. However, the relative order of different methods remains the same. This reinforces that FIGE can be used in stochastic domains as well.

5.2 Evaluation: Octopus Arm

In a second experiment, we investigate the performance of RPI using FIGE for graph node selection in the octopus arm domain [16]. The specific task is depicted in the left plot of Figure 4: the agent has to control the octopus arm such that it moves two food items (small yellow circles) into its mouth (large red circle). The base of the arm is restricted and cannot be actuated directly. The agent may control the arm in the following way: elongating or contracting the entire arm, bending the first half of the arm in either of the two directions, and bending the second half of the arm in either of the two directions. In each time step, the agent can control the elongation, the first half, and the second half of the arm independently, resulting in 8 discrete actions. The agent observes the positions x_i, y_i and velocities $\dot{x}_i$, $\dot{y}_i$ of the food items and of 24 selected parts of its arm (denoted by small black dots in Figure 4) and the angle and angular velocity of the the arm's base. Thus, the state space is continuous and consists of 106 dimensions. Each

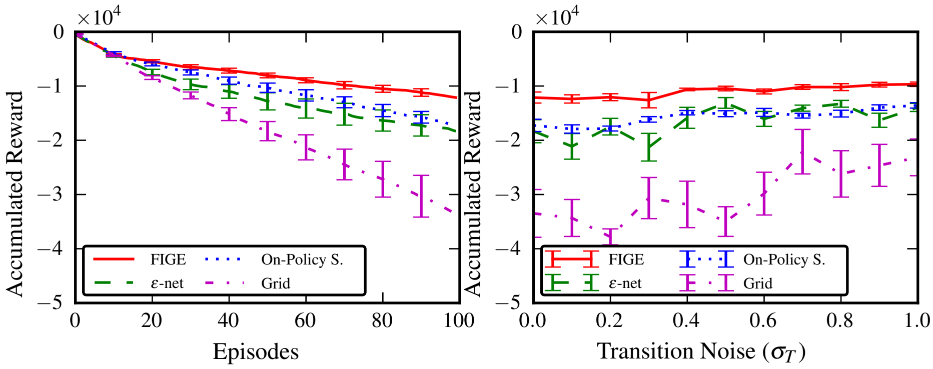

Fig. 3. Accumulated reward obtained by RPI in mountain car for different graph node selection heuristics. Left plot: learning curves in the deterministic domain. Right plot: reward accumulated after 100 episodes for different degrees of transition noise σ_T. Shown are mean and standard error of mean over 10 repetitions.

dimension is normalized such that its values fall into $[0,1]$. The agent obtains a reward of -0.01 per time step, a reward of 5 for moving the left food item into its mouth, and a reward of 7 for the right food item. The episode ends after 100 time steps or once both food items have been eaten. Because of the high-dimensional state space and the complex dynamics of the domain, the octopus arm problem is a challenging task.

We compare RPI combined with FIGE for $v_{num} = 75$ and $p_{num} = 5$ to LSPI using 75 radial basis functions (RBFs) as features ($\gamma = 0.99$, $\varepsilon = 0.01$). In the first 10 episodes, pure exploration without learning was conducted. The RBF centers c_i have been set to observed states such that the pairwise distance of the centers becomes maximal; the feature activation of center c_i for state s is computed as $\phi_i(s) = \exp(10||c_i - s||_2^2)$. The right plot of Figure 4 summarizes the results: using FIGE-based proto-value functions performs considerably better than standard RBF features; in particular, the agent learns in each run to move at least one food item into its mouth, which is not the case for LSPI. The main difference between the two approaches is that RBFs are local while proto-value functions can also capture more global properties. We suppose that since FIGE allows to capture the dynamics of a domain well, it allows to learn non-local proto-value functions that provide a useful bias to LSPI. In summary, the results suggest that FIGE can also support learning in high-dimensional problems.

6 Summary and Outlook

We have presented a new view on graph-based RL in continuous domains. Based on interpreting state transition graphs as generative models of the domain's dynamics, we have proposed a formulation for the likelihood of a graph for a given set of transitions. We have derived the new heuristic FIGE from the maximum likelihood objective under simplifying assumptions. FIGE allows to generate transition graphs that capture the domain's dynamics better than other heuristics. This is also reflected in the performance of representation learning and skill discovery methods that are built upon transition

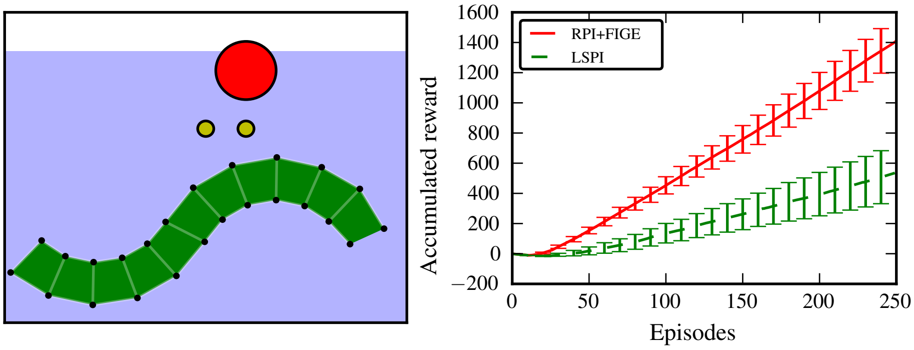

Fig. 4. Left plot: Visualization of the octopus arm task. Right plot: Accumulated reward obtained by LSPI and RPI using FIGE for graph node selection in the octopus arm domain. Shown are mean and standard error of mean over 20 repetitions.

graphs: in both kind of methods and across different domains, FIGE has achieved superior and more robust results than prior heuristics for transition graph generation. In general, our empirical results show that it makes a considerable difference how transition graphs are generated; for instance, using a grid-based discretization often had a catastrophic effect on the performance, even for low-dimensional domains.

FIGE is an offline, batch algorithm that requires considerable amounts of computation. This is less critical when it is combined with other offline approaches like RPI, LSPI, and non-incremental skill discovery based on graph-clustering, which are even more expensive in terms of computation. However, for making use of transition graphs in online methods like, e.g., OGAHC [11] for skill discovery, it would be highly desirable to develop an online method for graph generation that aims at similar objectives as FIGE. A further direction for future research would be to extend FIGE to domains with continuous action spaces and to use it for learning a policy representation that can be used within direct policy search approaches.

Acknowledgment This work was supported through a grant of the German Federal Ministry of Economics and Technology (BMWi, FKZ 50 RA 1217). The author would like to thank Yohannes Kassahun and the anonymous reviewers for many helpful comments.

References

1. Barto, A.G., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems 13(4), 341–379 (2003)
2. Digney, B.L.: Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments. In: From Animals to Animats: The 4th Conference on Simulation of Adaptive Behavior. pp. 363–372. Cambridge, MA (1996)
3. Digney, B.L.: Learning hierarchical control structures for multiple tasks and changing environments. In: 5th Conference on the Simulation of Adaptive Behavior. pp. 321–330 (1998)
4. Kirchner, F.: Automatic decomposition of reinforcement learning tasks. In: AAAI 95 Fall Symposium Series on Active Learning. pp. 56–59. Cambridge, MA, USA (1995)

5. Konidaris, G., Barto, A.G.: Skill discovery in continuous reinforcement learning domains using skill chaining. In: NIPS. vol. 22, pp. 1015–1023 (2009)
6. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. Journal of Machine Learning Research 4, 1107–1149 (2003)
7. Mahadevan, S., Maggioni, M.: Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. Journal of Machine Learning Research 8, 2169–2231 (2007)
8. Mannor, S., Menache, I., Hoze, A., Klein, U.: Dynamic abstraction in reinforcement learning via clustering. In: 21st International Conference on Machine Learning. pp. 560–567 (2004)
9. McGovern, A., Barto, A.G.: Automatic discovery of subgoals in reinforcement learning using diverse density. In: 18th International Conference on Machine Learning. pp. 361–368 (2001)
10. Menache, I., Mannor, S., Shimkin, N.: Q-Cut - dynamic discovery of sub-goals in reinforcement learning. In: 13th European Conference on Machine Learning. pp. 295–306 (2002)
11. Metzen, J.H.: Online skill discovery using graph-based clustering. Journal of Machine Learning Research W&CP 24, 77–88 (2012)
12. Şimşek, Ö., Barto, A.G.: Using relative novelty to identify useful temporal abstractions in reinforcement learning. In: 21st International Conference on Machine Learning. pp. 751–758 (2004)
13. Şimşek, Ö., Wolfe, A.P., Barto, A.G.: Identifying useful subgoals in reinforcement learning by local graph partitioning. In: 22nd International Conference on Machine Learning. pp. 816–823 (2005)
14. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence 112, 181–211 (1999)
15. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press (1998)
16. Yekutieli, Y., Sagiv-Zohar, R., Aharonov, R., Engel, Y., Hochner, B., Flash, T.: A dynamic model of the octopus arm. I. biomechanics of the octopus reaching movement. Journal of Neurophysiology 5, 291–323 (2005)

A Appendix: Derivation of FIGE's update equations

Let $T = \{(s_i, a_i s'_i)\}_{i=1}^{n}$ be a set of n transitions. We aim at finding a state transition graph G^* with v_{num} nodes such that $G^* = \arg\max_G p(T|G)$. We derive the FIGE update equations as maximum likelihood solutions for $p(T|G)$ under two simplifying assumptions.

(A1) For $(s,a,s') \in T$, assume $p(v'|v,a) = 1$ if $v = \mathrm{NN}_V(s) \wedge v' = \mathrm{NN}_V(s')$ else 0.

Assumption A1 allows to effectively decouple the likelihood $p(T|G)$ from the graph's edges and their weights $w^a_{vv'}$ such that it depends solely on the graph node positions and can thus be written as $p(T|V)$. By using assumption A1, we obtain:

$$\begin{aligned}\log p(T|G) &= \log \frac{1}{|A|^n|V|^n} \prod_{i=1}^{n} \left[\sum_{v\in V} p(s_i|v) \sum_{v'\in V} p(v'|v,a) p(s'|v',v,s_i) \right] \\ &\overset{A1}{=} \log \frac{1}{|A|^n|V|^n} \prod_{i=1}^{n} p(s_i|\mathrm{NN}_V(s_i)) p(s'_i|\mathrm{NN}_V(s'_i), \mathrm{NN}_V(s_i), s_i) \hat{=} \log p(T|V) \\ \log p(T|V) &= -n \log|A||V| + \sum_{i=1}^{n} \left[\log p(s_i|\mathrm{NN}_V(s_i)) + \log p(s'_i|\mathrm{NN}_V(s'_i), \mathrm{NN}_V(s_i), s_i) \right] \\ &= -n \log|A||V| + n \log N_b - \frac{1}{b^2} D\end{aligned}$$

with $D = \sum_{i=1}^{n} \left[\|s_i - \mathrm{NN}_V(s_i)\|_2^2 + \|(\mathrm{NN}_V(s_i') - \mathrm{NN}_V(s_i)) - (s_i' - s_i)\|_2^2 \right]$. For given V we create 2 partitions of T into v_{num} sets: $T^{\rightarrow}(v) = \{(s,s') | \exists (s,a,s') \in T : \mathrm{NN}_V(s) = v\}$ and $T^{\leftarrow}(v) = \{(s,s') | \exists (s,a,s') \in T : \mathrm{NN}_V(s') = v\}$. Furthermore, we create v_{num} sets $S^V(v) = \{s \mid \exists (s,a,s') \in T : \mathrm{NN}_V(s) = v\}$. For $|V| = v_{num} = const$, we can now maximize the log-likelihood $\log p(T|V)$ by minimizing D:

$$
\begin{aligned}
D &= \sum_{i=1}^{n} \left[\|s_i - \mathrm{NN}_V(s_i)\|_2^2 + 2\frac{1}{2} \|(\mathrm{NN}_V(s_i') - \mathrm{NN}_V(s_i)) - (s_i' - s_i)\|_2^2 \right] \\
&= \sum_{v} \Bigg[\sum_{s \in S^V(v)} \|s - v\|_2^2 + \frac{1}{2} \sum_{s,s' \in T^{\rightarrow}(v)} \|\mathrm{NN}_V(s') - v - s' + s\|_2^2 \\
&\qquad + \frac{1}{2} \sum_{s,s' \in T^{\leftarrow}(v)} \|v - \mathrm{NN}_V(s) - s' + s)\|_2^2 \Bigg]
\end{aligned}
$$

Each term of the outer sum corresponds to the contribution of node v's position to D; however, the terms cannot be minimized separately since they are coupled via $\mathrm{NN}_V(s)$ and $\mathrm{NN}_V(s')$. Minimizing them jointly is difficult because of the discontinuities of the nearest-neighbor terms. Thus, FIGE makes the following simplifying assumption:

(A2) Assume $p(T|V) = \prod_v p(T|v)$

This assumption implies that the couplings between the terms of D can be ignored and each v can be set greedily to the position where the respective term in the outer sum would become minimal as if all other $\tilde{v} \in V$ would remain unchanged. Finally, the greedy FIGE update equation which moves a node from position v_{old} to position v_{new} is

$$
\begin{aligned}
v_{new} = \arg\min_{v} \Bigg[& \sum_{s \in S^V(v_{old})} \|s - v\|_2^2 + \frac{1}{2} \sum_{s,s' \in T^{\rightarrow}(v_{old})} \|(\mathrm{NN}_V(s') - s' + s) - v\|_2^2 \\
& + \frac{1}{2} \sum_{s,s' \in T^{\leftarrow}(v_{old})} \|v - (\mathrm{NN}_V(s) - s + s')\|_2^2 \Bigg]
\end{aligned}
$$

In this, the first sum is minimized by choosing $v_{new} = v_a = \mathrm{MEAN}_{s \in S^V(v_{old})}(s)$, the second sum by choosing $v_{new} = v_b = \mathrm{MEAN}_{s,s' \in T^{\rightarrow}(v_{old})}(\mathrm{NN}_V(s') - s' + s)$, and the third by $v_{new} = v_c = \mathrm{MEAN}_{s,s' \in T^{\leftarrow}(v_{old})}(\mathrm{NN}_V(s) - s + s')$. By using forces that pull v_{new} to v_a, v_b, and v_c with the respective weights, we obtain the FIGE update rule

$$
v_{new} = v_{old} + \alpha \left[\frac{1}{2}(v_a - v_{old}) + \frac{1}{4}(v_b - v_{old}) + \frac{1}{4}(v_c - v_{old}) \right].
$$

Since $A2$ is oversimplifying, one sweep of the FIGE update equations will typically not find the maximum likelihood solution. Thus, FIGE performs several update iterations to account for couplings between nodes.