Optimal Clustering with Dependent Costs in Bayesian Networks

PAUL PAO-YEN WU*, Centre for Data Science, Queensland University of Technology, Australia
FABRIZIO RUGGERI, Institute of Applied Mathematics and Information Technology, Italian National Research Council, Italy
KERRIE MENGERSEN, Centre for Data Science, Queensland University of Technology, Australia

Background: Clustering of nodes in Bayesian Networks (BNs) and related graphical models such as Dynamic BNs (DBNs) has been demonstrated to enhance computational efficiency and improve model learning. It typically involves partitioning the underlying Directed Acyclic Graph (DAG) into cliques or optimising for some cost or criteria.
Objectives: Given a DAG, we focus on a critical but understudied aspect of optimal clustering involving cost dependency. This is where inference outcomes and hence clustering costs depend on both nodes within a cluster and the mapping of clusters that are connected by at least one arc.
Methods: We propose a novel algorithm called Dependent Cluster MAPping (DCMAP) which can, given an arbitrary, positive cost function, iteratively and rapidly find near-optimal, then optimal cluster mappings.
Results: DCMAP is shown analytically to be optimal in terms of finding all of the least cost cluster mapping solutions and with no more iterations than an equally informed algorithm. Demonstrated on a complex systems seagrass DBN with 9.91×10^9 and 1.51×10^{21} possible cluster mappings for 25 and 50 node configurations, it took 856 and 1569 iterations on average to find the first optimal solution, respectively.
Conclusions: The effectiveness of DCMAP enables future research in BN learning using optimisation, such as through enhancing computational efficiency or minimising entropy for learning. This is critically important as computation of marginal distributions or updating model parameters is NP-hard.

JAIR Associate Editor: Julia Handl

JAIR Reference Format:
Paul Pao-Yen Wu, Fabrizio Ruggeri, and Kerrie Mengersen. 2026. Optimal Clustering with Dependent Costs in Bayesian Networks. *Journal of Artificial Intelligence Research* 87, Article 4 (September 2026), 29 pages. DOI: 10.1613/jair.1.23620

1 Introduction

Bayesian Networks (BNs) and related methods including Dynamic BNs (DBNs) are a form of graphical model that are represented as Directed Acyclic Graphs (DAGs), where relationships between nodes (factors) are shown as arcs and quantified with conditional probabilities (Pearl 1988). They have been applied widely across domains including medicine, environment, economics and social sciences to provide explanatory decision support, integrating data and even expert knowledge (Kitson et al. 2023; Wu, Mengersen, et al. 2017). However, one of the key challenges associated with inference and learning of BNs and DBNs is computation, which is NP-hard (Murphy 2002; Park and Darwiche 2004). To this end, different approaches to clustering of BN nodes have been used in the context of supporting more computationally efficient inference of marginal distributions given evidence, and model

*Corresponding Author.

Authors' Contact Information: Paul Pao-Yen Wu, ORCID: 0000-0001-5960-8203, p.wu@qut.edu.au, Centre for Data Science, Queensland University of Technology, Brisbane, Queensland, Australia; Fabrizio Ruggeri, ORCID: 0000-0003-0615-4417, fabrizio@mi.imati.cnr.it, Institute of Applied Mathematics and Information Technology, Italian National Research Council, Milano, , Italy; Kerrie Mengersen, ORCID: 0000-0001-8625-9168, k.mengersen@qut.edu.au, Centre for Data Science, Queensland University of Technology, Brisbane, Queensland, Australia.

DOI: 10.1613/jair.1.23620

learning, to update the structure of the graph and/or the conditional probabilities associated with each node (Daly et al. 2011; Kojima et al. 2010; Lauritzen and Spiegelhalter 1988; Lu et al. 2021). Appendix A illustrates with a simple example how three different clustering approaches reduce computational cost to different degrees. A general approach to optimisation of BN clusters (i.e. with aribtrary criteria) can advance research in probabilistic graphical models by providing a benchmark for validating approximate inference methods, which are necessary in high-dimensional problems, and enable new methods for computation and inference such as by finding homogeneous or correlated regions in a non-homogeneous system (Lin et al. 2020; Wu, Caley, et al. 2018).

Given a DAG, optimal clustering of a BN seeks to minimise some cost or objective function associated with the mapping of BN nodes to clusters. Typically, each node is uniquely mapped to a cluster and the total cost is the sum of costs associated with each cluster. However, a key challenge is cost dependency. Consider a BN $H(\boldsymbol{X},\boldsymbol{A})$ with nodes $X_i \in \boldsymbol{X}$ and arcs (or edges) $\boldsymbol{A}$ that encode conditional dependence between parent and child nodes. Fundamentally, BN inference is underpinned by conditional probabilities and conditional independence, where the joint probability distribution over all nodes $\boldsymbol{X}$ given observations (i.e. evidence $\boldsymbol{E}$) is (Pearl 1988; Wu, Caley, et al. 2018):

$$P(\boldsymbol{X},\boldsymbol{E}) = \prod_{X_j \in \boldsymbol{X}} P(X_j|Par(X_j))\delta(X_j) \tag{1}$$

where $P(X_j|Par(X_j))$ is the conditional probability of node X_j given its parents, and $\delta(X_j)$ the hard or virtual evidence about individual nodes, which can be the inputs to or observations for inference (Mrad et al. 2015). The joint probability distribution arises from the multiplication of conditional probability distributions, each of which is potentially multi-dimensional, comprising a node (or variable) X_j and parent nodes $Par(X_j)$. Hence, intuitively: (i) clustering determines which conditional probability distributions are computed together, impacting computational complexity, and (ii) the computed outcomes of a given cluster are dependent on clusters connected by at least one arc via multiplication as per equation (1). This propagation of partial inference outcomes between clusters has been employed in approaches to BN and DBN inference such as Lauritzen and Spiegelhalter (1988) and Wu, Caley, et al. (2018). The dependency of the local cluster cost on the mapping of connected clusters is referred to as cost dependency, noting that the objective is to minimise the global cost which is a combination (typically additive) of local costs. However, finding optimal clusters (i.e. partitions) is NP-hard in general (Buluç et al. 2016) and cost dependency exacerbates this complexity.

The clustering problem can be formalised as one of assigning a unique integer cluster label $u(X_j) = k, k = 1,\ldots,m$ to each node $X_j \in \boldsymbol{X}$ of a BN with DAG $H(\boldsymbol{X},\boldsymbol{A})$ and conditional probability parameters $\boldsymbol{\theta} = \{P(X_j|Par(X_j)\}$; the total number of clusters m is determined through optimisation. Since it is the DAG nodes that are clustered, for notational simplicity, let $G\big(H(\boldsymbol{X},\boldsymbol{A}),u(\boldsymbol{X})\big)$ be the total cost, and the goal is to find the cluster mapping $u(\boldsymbol{X})$ (i.e. partitioning) that minimises this cost. Notate $G^*\big(H(\boldsymbol{X},\boldsymbol{A})\big)$ as the optimal cost, and assume that the total cost is the sum of individual cluster costs G_k:

$$G^*\big(H(\boldsymbol{X},\boldsymbol{A})\big) = \arg\min_{u(\boldsymbol{X})}\Big(G\big(H(\boldsymbol{X},\boldsymbol{A}),u(\boldsymbol{X})\big)\Big) = \arg\min_{u(\boldsymbol{X})}\left(\sum_{k=1}^{m} G_k\right). \tag{2}$$

Note that there may be multiple cluster mapping solutions $u_s^*(\boldsymbol{X})$ with cost G^*; cost dependency implies that G_k depends on the mapping of connected clusters.

1.1 Case Study

We use as a case study a DBN of a seagrass ecosystem (Wu, Caley, et al. 2018) to show how clustering can minimise computational cost. Seagrasses are a critical primary habitat for fish and many endangered species including the dugong and green turtle, and contribute USD$1.9 trillion per annum through nutrient uptake and cycling. However, better management of human stressors such as dredging requires the modelling of cumulative

effects arising from interactions between biological, ecological and environmental factors. The non-homogeneous DBN itself is complex and computationally intensive, and inference is infeasible within a standard (MCMC) framework for real-world decision support contexts. Furthermore, understanding areas of uncertainty in the system and designing data collection strategies to mitigate uncertainty is of utmost interest in managing such complex systems. This uncertainty can be characterised by entropy with clusters corresponding to scenarios of evidence (i.e. observed data) (Section 5.2). We focus on computation as a cost function G to demonstrate how the proposed algorithm can find clusters to minimise the computational complexity of inference, and discuss how our algorithm could also be used to minimise other cost functions such as entropy.

1.2 Existing Work

Clustering of BNs has been employed in a wide range of contexts, evolving from a focus on computation of marginal distributions given evidence such as the seminal work of Lauritzen and Spiegelhalter (1988) to predominantly causal discovery and graph learning (Kitson et al. 2023; Lu et al. 2021) in recent times. Almost all existing methods for inferring marginal distributions use heuristics based on the graph to form clusters, such as triangulation in join-tree inference (Lauritzen and Spiegelhalter 1988), conditioning sets (cutsets) in bucket elimination (Dechter 1999), factor graphs (Kschischang et al. 2001) and approximate inference with cutsets (Bidyuk and Dechter 2007). They help tackle the challenge of exponential growth in computational complexity with induced tree width (Lin et al. 2020). In some methods, such as mini-clustering (Mateescu et al. 2010), incremental region selection (Forouzan and Ihler 2015), clusters for selective filtering in a DBN (Albrecht and Ramamoorthy 2016), and triplet region construction (Lin et al. 2020), the heuristically formed clusters are refined through greedy search or local optimisation based on some function such as conditional entropy. Much of the more recent work focuses on approximate methods to tackle high-dimensional problems. However, explicit study of global optimisation of clusters and their impact on computation and inference is missing in this literature.

In contrast, the BN learning literature has seen some progress in clustering and inference towards global optimisation. There are three main approaches: (i) constraint based, that evaluate conditional independence structures i.e. if an arc exists or not, then its direction, (ii) scoring, often with heuristics to evaluate goodness of fit of a proposed network, and (iii) hybrid, combining constraints with heuristics (Huang and Zhou 2022) .

A number of approaches use clusters as a structure to restrict or constrain the local search space in a graph learning framework such as the seminal work of Friedman et al. (1999). They define clusters in a similar vein to clique-trees (Lauritzen and Spiegelhalter 1988) in a "restrict and maximise" framework to optimise a mutual-information based score of the learned network. Kojima et al. (2010) advance this approach, effectively relaxing the cluster tree requirement in Friedman et al. (1999) to a cluster graph to help address the issue of large clusters, which make local optimisation computationally infeasible. They used ancestral constraints to help limit the search space of clusters and the algorithm was shown to outperform greedy algorithms including Max-Min Hill Climbing (MMHC) and Hill Climbing (HC) with Tabu search. However, computational feasibility becomes a problem again when there are too many edges between clusters (Kojima et al. 2010).

Lu et al. (2021) also adapt the work of Friedman et al. (1999) for undirected graphs, using clusters based on proposed parents and children of a node X to constrain the search in an iterative framework based on A*, a heuristic-informed search algorithm. They show empirically that for small networks, exhaustive and in some cases optimal methods such as A* and SAT (Boolean SATisfiability) outperform greedy methods such as PC (Peter and Clark), which can suffer from convergence to local optima, and their adaptation of A* performs as well as greedy methods in larger networks. However, their approach is exponential in complexity with neighbourhood size (i.e. number of parents).

Zhang et al. (2018) present an alternate approach in which clusters are determined through agglomerative clustering using Pearson correlation as the distance metric; however, little is provided in terms of algorithm

performance and its convergence characteristics. Huang and Zhou (2022) and Gu and Zhou (2020) both develop comprehensive approaches that use similar agglomerative clustering based on mutual information and correlation as the distance metric, respectively. The former learns skeletons (an undirected version of the graph) for each cluster using their variant of the PC algorithm and log-likelihood ratio tests of conditional independence. Gu and Zhou (2020) use a score-based algorithm to learn partial DAGs for each cluster, which are then fused in the final step proposing edges between clusters. Huang and Zhou (2022) combine clusters with automated parameter tuning for turning skeletons into partial then fully directed acyclic graphs and hybrid greedy initialisation, obtaining empirical improvements in computational efficiency and more accurate DAGs. However, in both cases, the performance is dependent on the quality and size of the clusters.

Clearly, clustering has demonstrated utility to improve computational feasibility in BN inference of marginal distributions and learning BNs with varying degrees of optimality given a scoring function (i.e. cost function). However, none of the reviewed literature explicitly studied globally optimal clustering and its impact on inference and computation. Existing BN learning methods focus on inclusion or exclusion of edges, rather than scores that change given different mappings of nodes to clusters. Given the pervasive role clustering plays in BN research and the utility of dynamic programming and clustering for optimisation (Friedman et al. 1999; Kojima et al. 2010; Lu et al. 2021), this paper focuses on explicit, generalisable optimisation of clusters in the context of BN inference.

Note that there also exists substantial literature in graph partitioning such as repeated graph bisection, Kernighan and Lin algorithm, spectral bisection and multilevel partitioning (Bader et al. 2013; Buluç et al. 2016). Zheng et al. (2020), for instance, demonstrate how dynamic programming can be used to optimally and efficiently cluster nodes in the absence of cost dependency, applied to convolutional neural networks. However, the problem context for these approaches typically involves partitioning a graph into a set number of clusters or to satisfy some criteria for each cluster defined with respect to static node and/or edge weights. This differs significantly from the BN context and associated cost dependency between clusters.

1.3 Layout of Paper

We develop a novel algorithm called DCMAP (Dependent Cluster MAPping) that optimises BN cluster mappings given an arbitrary, non-negative scoring function which we refer to as a cost function in the presence of cost dependency. Such dependency arises in the BN context of using clustering to improve computational efficiency when inferring marginal distributions given evidence (Lauritzen and Spiegelhalter 1988) and in model learning (Friedman et al. 1999; Lu et al. 2021). A simple example is provided in Appendix A showing how DCMAP (least computation) and two other popular clustering approaches reduce computation. Noting as background the context of BN inference (Section 2.1) and introducing an illustrative example (Section 2.2), we first present an intuitive overview of DCMAP and its notation (Section 2.3), followed by its derivation from dynamic programming principles (Section 2.3.1 and 2.3.3). The algorithm is presented in detail in Section 3 and shown analytically to find all the optimal solutions such that an equally informed algorithm cannot do so in fewer iterations. An empirical demonstration is provided on the case study usage scenario of using clusters to minimise computation cost of inference for our complex systems seagrass DBN in Section 4. Reflections and suggestions for future work, including extension to other cost functions for optimisation such as entropy are provided in the Discussion (Section 5).

2 DCMAP Algorithm: Background and Overview

This section develops the foundational concepts for the proposed algorithm DCMAP and their relationship to BNs. It starts with some background on BN inference and an illustrative example of how clustering affects computation, provides an overview of the notation, and develops foundational concepts of layers, clusters and dynamic programming. Using these, a high-level conceptual presentation of the algorithm is provided along

with the derivation of the objective function. Finally, an estimate of the size of the search space is provided to contextualise the complexity of cluster mapping.

2.1 Background: BN Inference

In almost every usage scenario involving BNs including evaluation of what-if scenarios and model learning, it is necessary to compute marginal distributions for each node X given evidence $P(X|\boldsymbol{E})$ in the DAG $H(\boldsymbol{X}, \boldsymbol{A})$. This marginal distribution is obtained from (1) by marginalising (i.e. summing) out all other nodes (Wu, Caley, et al. 2018):

$$P(X|\boldsymbol{E}) = \frac{P(X, \boldsymbol{E})}{P(\boldsymbol{E})}$$
$$P(X, \boldsymbol{E}) = \sum_{X_i \in \boldsymbol{X} \setminus X} \prod_{X_i \in \boldsymbol{X}} P(X_i|Par(X_i))\delta(X_i) \tag{3}$$

where $P(X_i|Par(X_i))$ is the conditional probability of node X_i given its parents, $\setminus$ denotes set difference, and $\delta(X_i)$ is the evidence for node X_i. Evidence can be used to, among other things, incorporate observations and evaluate scenarios. The joint distribution (1) has $|\boldsymbol{X}|$ nodes or dimensions, and $\sum_{X_i}$ denotes a summation over all discrete states of node X_i which we denote as lowercase $x_{i1}, x_{i2}, \ldots$, effectively removing that dimension from the joint distribution through marginalisation. Note that $P(\boldsymbol{E})$ is a constant that normalises the resulting distribution (Lauritzen and Spiegelhalter 1988). Although the notation focuses on discrete probability distributions to minimise complexity, the proposed clustering algorithm is generalisable to continuous distributions since it ultimately relies only on an arbitrary non-negative cost function.

2.2 Illustrative Example

Intuitively, if the clustering cost function to be optimised (2) is dependent in some way on BN inference (3), then it will be affected by cost dependency due to the propagation of probabilities via parent-child arcs between clusters. To help illustrate this and the proposed algorithm, consider an example discrete BN DAG (Fig. 1). Assuming binary states for all nodes, the Conditional Probability Table (CPT) for each node with no parents (nodes A, B, C) is a 2×1 matrix, e.g. $P(A) = [p_a, p_{\bar{a}}]^T$ for states $A = a$ and $A = \bar{a}$. The CPT for F is a 2×2 matrix, i.e. $P(F|A) = \begin{bmatrix} p_{af} & p_{\bar{a}f} \\ p_{a\bar{f}} & p_{\bar{a}\bar{f}} \end{bmatrix}$ where p_{af} is the probability of state $F = f$ given state $A = a$. $P(D|A, B)$ and $P(G|D, E)$ are 2×4 matrices. Multiplication of CPTs (1) containing different nodes, i.e. variables, grows the dimensionality of the resultant matrix with associated exponential growth in computation; in the discrete case, the resultant matrix contains all possible combinations of node states. We ignore evidence terms for notational simplicity, noting that evidence can be easily incorporated using $\delta(X_i)$ with node X_i (1). Using (3), the posterior probability $P(F)$ is:

$$P(F) = \sum_{A,B,C,D,E,G} P(A)P(B)P(C)P(D|A, B)P(E|C)P(F|A)P(G|D, E) \tag{4}$$
$$= \sum_{A} P(F|A)P(A) \left(\sum_{D,E,G} P(G|D, E) \left(\sum_{B} P(D|A, B)P(B) \left(\sum_{C} P(E|C)P(C) \right) \right) \right) \tag{5}$$

Consider, as an example, a cost function based entirely on computational cost, and one approach where all nodes are in one cluster (4) where the product of conditional probabilities produces a matrix of 7 dimensions, one for each node A through G, with exponential growth in memory (2^7 joint probability values) and computation associated with multiplication (608 products) and marginalisation (126 sums). For example, $P(A)P(B)$ requires 4 multiplications, and $P(A)P(B)P(D|A, B)$ requires $4 + 8 = 12$. $\sum_A P(F|A)$ requires 2 summations.

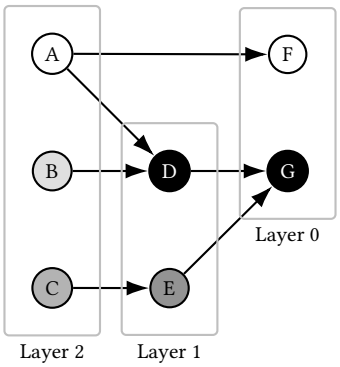


Fig. 1. Example DAG showing nodes, directed edges, and layers where shading denotes clusters obtained using DCMAP. Assume all nodes have binary states.

Contrast this with the cluster mapping shown through shading (e.g. A and F are in the same cluster, as are D and G; all others in a cluster of their own) in Fig. 1. We illustrate clustering with a simple approach where probabilities from an antecedent cluster are multiplied with those in the current cluster then marginalised using an arbitrary ordering. The resultant computation (5) has maximum dimensionality of 4 due to interspersed multiplication and marginalisation; recall marginalisation happened at the end in (4). In this example, clustering greatly reduces computation and memory cost. It also demonstrates cost dependency, since the computation cost required at a descendant cluster is dependent on the mapping and outcomes of antecedent clusters. Finally, it demonstrates the forwards and backwards nature of inference, such as forwards propagation from cluster E to cluster $\{D, G\}$, and backwards propagation from $\{D, G\}$ to cluster $\{A, F\}$. For the full example, see Appendix A.

2.3 Overview and Notation

In short, given a BN $H(\boldsymbol{X}, \boldsymbol{A})$, the task is to find one or more cluster mappings $U_s(\boldsymbol{X})$ that minimise the total cost as per equation (2). We first define layers (Section 2.3.1) and clusters (Section 2.3.2) as a way to decompose the DAG to enable step-by-step optimisation via dynamic programming (Section 2.3.3). Notation is summarised in Table 1.

2.3.1 Layers.

Definition 1. *Given a DAG $H(\boldsymbol{X}, \boldsymbol{A})$, define the layer $l(X_j) = l$ of node $X_j \in \boldsymbol{X}$ as the length of the longest path (i.e. number of arcs) following the direction of the arcs (parent to child) to the set of leaf nodes $X \in \boldsymbol{X}^{leaf}$, i.e., nodes with no child nodes.*

We assume that H is fully connected, since cluster mapping could be applied independently to each connected component. We use a simple breadth-first search (LaValle 2006) to compute $l(\boldsymbol{X})$ in $O(n)$ time (Algorithm 1).

Lemma 2.1. *Any pair of nodes in the same layer $(X_1, X_2) : l(X_1) = l(X_2) = l$ must be disjoint, i.e. there cannot be an arc between them.*

This can be easily shown through contradiction since if there were an arc from X_1 to X_2, then X_1 is a parent of X_2 and Algorithm 1 would have updated $l(X_1)$ to $l + 1$ at the next iteration on line 7, which means they cannot be from the same layer. Hence, all nodes in the same layer are disjoint (see Fig. 1 for an example). Additionally, since there are no cycles in a DAG, Algorithm 1 is guaranteed to terminate after at most $n = |\boldsymbol{X}|$ iterations.

Table 1. Summary of notation. Generally, we use lower case notation (e.g. g, u) for local calculation, upper case (e.g. G, U) for global search and optimisation, non-italicised (e.g. G, U) for data structures in our algorithm, and **bold** for a set. We notate $k = 1..n$ to denote $k = 1, 2, ..., n$ for brevity.

Symbol	Definition
$H(\boldsymbol{X}, \boldsymbol{A})$, n, $\boldsymbol{X}^{\text{leaf}}$	DAG H with n nodes $X \in \boldsymbol{X}$, leaf nodes $\boldsymbol{X}^{\text{leaf}} \in \boldsymbol{X}$, and arcs A.
$Par(), Child()$	These functions return the parent and child nodes / cluster / cluster-layers, respectively, of a given node(s) / cluster(s) / cluster-layer(s).
$G()$	Objective function cost given nodes and cluster mappings; G^* is the optimal cost (Section 2.3.3) and G_{min} is the current minimum global cost in the search.
$c()$	Transition cost for one step of dynamic programming (10); positive, non-zero function of nodes and associated conditional probabilities in a cluster-layer and child cluster-layer(s).
$l, \boldsymbol{X}_l$	Layer, $l = 0$ (i.e. leaf nodes), last layer l_{max}; L is the largest i.e. latest layer of a parent node (Section 2.3.1). $\boldsymbol{X}_l$ denotes nodes in layer l.
k, $\boldsymbol{X}_k$, $\phi(\boldsymbol{X}_k)$, $\pi(\boldsymbol{X}_k)$	Cluster label k with $\boldsymbol{X}_k$ denoting nodes in that cluster, $\phi(\boldsymbol{X}_k)$ and $\pi(\boldsymbol{X}_k)$ internal and link nodes (Section 2.3.1).
$kl, \boldsymbol{X}_{kl}$	Cluster-layer label and set of cluster-layer nodes, respectively (Section 2.3.3).
$u(X), U$	Cluster mapping of node X to integer cluster label k; U is a map for a vector of nodes, $\hat{\text{U}}$ a mapping proposal, and U^* a mapping with optimal cost G^*.
$J()$	Function to capture dependencies between a cluster-layer and its child cluster-layer(s) (Section 2.3.3).
$\hat{g}$	Heuristic estimate of the total cost (Section 2.3.3).
$h()$	Heuristic estimate of true cost $G()$ given remaining nodes; used to compute $\hat{g}$ above.
Q	Queue for prioritising the search; Q' are eligible queue entries for popping (Section 3).
b, Br_{lk}, Br_{pl}	Branch b of the search and data structures for branch linking Br_{lk} and branch updates Br_{pl} (Section 3).

Algorithm 1 Layer assignment algorithm; $Par(\boldsymbol{X}')$ denotes the union of all parent nodes of a set of nodes $\boldsymbol{X}'$.

```
1: procedure LAYER
2:     Initialise l ← 0, X' ← X^leaf, l(X) ← ∞
3:     l(X') ← 0
4:     while Par(X') ≠ ∅ do
5:         X' ← Par(X')
6:         l ← l + 1
7:         l(X') ← max (l(X'), l)
8:     end while
9: end procedure
```

Remark 2.1. *As the layer $l(X_1)$ is by definition the longest path to a leaf node, then $l(X_1) \geq l(X_2) + 1$ where X_2 is a child of X_1 (equivalently $X_1 \in Par(X_2)$); potentially, there exists a longer path from X_1 to a leaf node than $l(X_2) + 1$. In Fig. 1, node A and F are an example of this as $l(A) = 2$ and $l(F) = 0$.*

2.3.2 Clusters.

Definition 2. *Nodes are mapped to clusters using integer labels $u(X) = k$ where $1 \leq u(X) \leq m$ for m unique clusters, $m \leq n$. Denote the set of nodes in cluster k as $\boldsymbol{X}_k$. Denote $u'(X)$ as a proposed cluster mapping for node X.*

Clusters must be contiguous (Definition 3) to avoid possible introduction of cycles between clusters.

Definition 3. *A cluster $\boldsymbol{X}_k$ is contiguous iff given any two nodes $X_1, X_2 \in \boldsymbol{X}_k$, there are no parent-child paths $\boldsymbol{X}_p = \{X_1, \ldots, X', \ldots, X_2\}$ with a node X' mapped to a different cluster.*

BN inference assumes a DAG with no cycles to ensure convergence when propagating evidence (Pearl 1988). This extends to clusters (or cliques) to enable computationally efficient propagation of evidence between clusters

as part of BN inference (Lauritzen and Spiegelhalter 1988). As an example, if in Fig. 1 nodes B, G were in cluster 1 and D in cluster 2, then that breaks the contiguity of cluster 1 and introduces a cycle between cluster 1 and 2 via the path $\{B, D, G\}$. If instead nodes D, G were in cluster 1 and B in cluster 2, then contiguity is maintained and there are no cycles between cluster 1 and 2. Since the underlying DAG is acyclic, it is easy to see that, with contiguity, the resulting clusters are also acyclic.

Definition 4. *Define a cluster-layer kl as a subset of nodes $\boldsymbol{X}_{kl} \subseteq \boldsymbol{X}_k$ comprising all the nodes in layer l of cluster $\boldsymbol{X}_k$. Therefore, $\bigcup_{l=l_k^0}^{l_k^1} \boldsymbol{X}_{kl} = \boldsymbol{X}_k$ where l_k^0 and l_k^1 are the earliest and latest layers of cluster $\boldsymbol{X}_k$; hence $l_k^1 \geq l_k^0$.*

Definition 5. *Nodes in cluster $\boldsymbol{X}_k$ with at least one child node in a different cluster are denoted as link nodes $\pi(\boldsymbol{X}_k)$, and the remaining nodes are internal nodes $\phi(\boldsymbol{X}_k)$, i.e. $\boldsymbol{X}_k \backslash \pi(\boldsymbol{X}_k) = \phi(\boldsymbol{X}_k)$.*

In a similar vein, parent $Par(\boldsymbol{X}_k)$ or child clusters $Child(\boldsymbol{X}_k)$ must have at least one arc linking them to cluster $\boldsymbol{X}_k$, and parent $Par(\boldsymbol{X}_{kl})$ or child cluster-layers $Child(\boldsymbol{X}_{kl})$ also have at least one arc linking them to $\boldsymbol{X}_{kl}$. Note, a cluster mapping $u(\boldsymbol{X}_k)$ induces a unique mapping of link and internal nodes based on parent-child relationships between different clusters. However, the reverse is not necessarily true. Hence, our optimisation task focuses on finding $u(\boldsymbol{X}_k)$.

2.3.3 Dynamic Programming. We use dynamic programming to find optimal cluster mappings. Generally, dynamic programming is formulated as the optimisation of an objective function (or cost function) G over a series of decisions or actions u_k for $k = 0, \ldots, K-1$ where the optimal cost G^* over an interval $[k, K]$ is (Bellman and Dreyfus 1962):

$$G^*_{(k,K)}(x) = \inf_{u_k \in \mathcal{A}_k} \left[G_{(k,K)} \left(x_k, u_k, U^*_{(k+1,K-1)} \right) \right] \tag{6}$$

where $U^*_{(k+1,K-1)}$ is the optimal sequence of actions $u_{k+1}, \ldots, u_{K-1}$ for the interval $[k+1, K-1]$, $\mathcal{A}_k$ is the set of possible actions at stage k, x is the state at stage K as a consequence of optimal choice of u_k at state x_k, and inf is the infimum operator which returns the argument that gives the minimum expression. Intuitively, given the current state x_k and an optimal sequence of actions at stages $k+1$ to $K-1$ leading to state $x = x_K$, we choose the action u_k to take at the current stage to minimise the overall cost in going from stage k to K. This formulation is the most flexible as the cost function can vary (i.e time-varying in the literature, but we refer to as stage varying to avoid confusion with time in a DBN) across intervals $[k, K]$, and does not have to be additive. This is useful as there can be multiple clusters in a given layer of the DAG to cluster (Section 2.4). However, computational complexity grows in NP time with graph size and with the number of proposals $|\mathcal{A}_k|$ at each stage (LaValle 2006).

If the objective function G is additive across stages and independent of k (implying that the system is time invariant), then (6) simplifies to

$$G^*_{k+1}(x) = \inf_{u \in \mathcal{A}} \left[c(x, u) + G^*_k \left(f(\boldsymbol{x}, \boldsymbol{u}) \right) \right] \tag{7}$$

where $f(\boldsymbol{x}, \boldsymbol{u})$ is a sequence of states and actions, starting at stage $k = 0$, resulting in state x at stage k with optimal cost G^*_k, and c is the transition cost of enacting action u at state x. An additive and time-invariant system greatly reduces computational complexity. Optimisation algorithms are built around the objective function, and Equation (7) is the basis for many heuristic search techniques such as A*. A heuristic estimate of the remaining cost over stages $[k+1, K]$ is used in A* to prioritise the search and potentially reduce computation time, such as for BN learning (Lu et al. 2021).

2.4 High-Level Algorithm

The cost function being optimised through clustering can be arbitrary, but its computation differs from inferring marginal distributions for every node (3) as the latter requires forwards and backwards propagation, whilst the

former computes a global cost based on propagation in one direction, akin to scoring in BN learning (Section 1.2). This cost function could be the computational cost associated with cluster-based inference (equation 3), entropy, predictive error, or information criterion such as the Watanabe-Akaike Information Criterion (WAIC).

The proposed algorithm DCMAP uses layers l as dynamic programming stages in equation (6) for incremental optimisation of the cost, beginning at layer $l = 0$ (i.e. leaf nodes) and working backwards through the BN from earlier to later layers, culminating at $l_{\max}$. A queue Q is used to order the evaluation of possible cluster-layer mappings $\hat{\mathrm{U}}$, which are linked to connected cluster-layers, by lowest overall cost first. The overall cost includes the previous layer cost $G(\boldsymbol{X}_{l-1})$, the cost c of the proposed cluster-layer, and a heuristic estimate h of the remaining cost (11).

Conceptually, different cluster-layer mappings could be thought of as branches in the search tree, which are linked at layer l if they share the same mappings u at layers $l = 0, \ldots, l-1$. Intuitively, an efficient algorithm prunes branches that are dominated at the earliest opportunity to minimise computation time. These high level components of the algorithm are illustrated in Fig. 2.

2.5 Objective Function and Heuristic

Cost dependency and the potential for the parent $Par(X)$ of a child node $X \in \boldsymbol{X}_l$ to be at a layer $l' > l+1$ (Remark 2.1) violates the assumption of time invariance (7). Hence, a new objective function is derived that can accommodate this for the proposed cluster mapping algorithm (Fig. 2).

The structure of layers $\boldsymbol{X}_l$ remain fixed whilst cluster-layers $\boldsymbol{X}_{kl}$ change in a cluster-mapping setting. Hence, layers could be used as a structure for a time-invariant, objective function. Let the latest antecedent layer of a parent be L:

$$L = \max\Big(l\big(Par(\boldsymbol{X}_l)\big)\Big) \tag{8}$$

Substituting into the general objective function of Bellman and Dreyfus (1962) (6), the objective function over layers $[l, L]$ is:

$$G^*_{(l,L)}(x) = \inf_{u(\boldsymbol{X}_l)\in U_l}\Big[G_{(l,L)}\Big(\boldsymbol{X}_l, u(\boldsymbol{X}_l), U^*_{(l+1,L)}\Big)\Big] \tag{9}$$

where u describes a cluster mapping and U^* a series of optimal cluster mappings. Here, the propagation of probabilities during inference occurs between nodes in layer l and parent nodes in layers $l+1$ up to and including L, which means nodes at these layers need to be considered when finding the optimal cluster mappings at layer l. The objective function is thus stage-varying and, in this formulation, not necessarily additive. However, such a cost function is computationally expensive to evaluate.

As disjoint nodes are conditionally independent and nodes in a layer or cluster-layer are disjoint by Definition 4, we assume that the objective function cost of disjoint cluster-layers are independent and thus additive. Moreover, by definition of L (8) as the latest parent layers given nodes at layer l, an additive and time-invariant formulation for the objective function could be obtained by finding the lowest cost cluster-mapping (the infimum) over all intermediary cluster-layers. This replaces the $U^*_{l+1,L}$ term in (6) and makes G time invariant. Thus, (9) can be re-written as:

$$\begin{aligned} G^*(\boldsymbol{X}_L) &= G^*(\boldsymbol{X}_l) + \\ &\inf_{u(\boldsymbol{X}_{l+1,\ldots,L})} \sum_{l'=l+1}^{L}\sum_{k=1}^{m_{l'}} c\Big(\boldsymbol{X}_{kl'}, u(\boldsymbol{X}_{kl'}), Child(\boldsymbol{X}_{kl'}), u\big(Child(\boldsymbol{X}_{kl'})\big)\Big) \end{aligned} \tag{10}$$

where the optimal cost $G^*(\boldsymbol{X}_L)$ is only a function of the optimal cost at layer l and cluster-layer mappings between $l+1$ and L inclusive, notated as $u(\boldsymbol{X}_{l+1,\ldots,L})$, and c is a positive, non-zero cost that is a function of nodes, cluster mappings and implicitly associated conditional probabilities in a cluster-layer and child cluster-layer(s).

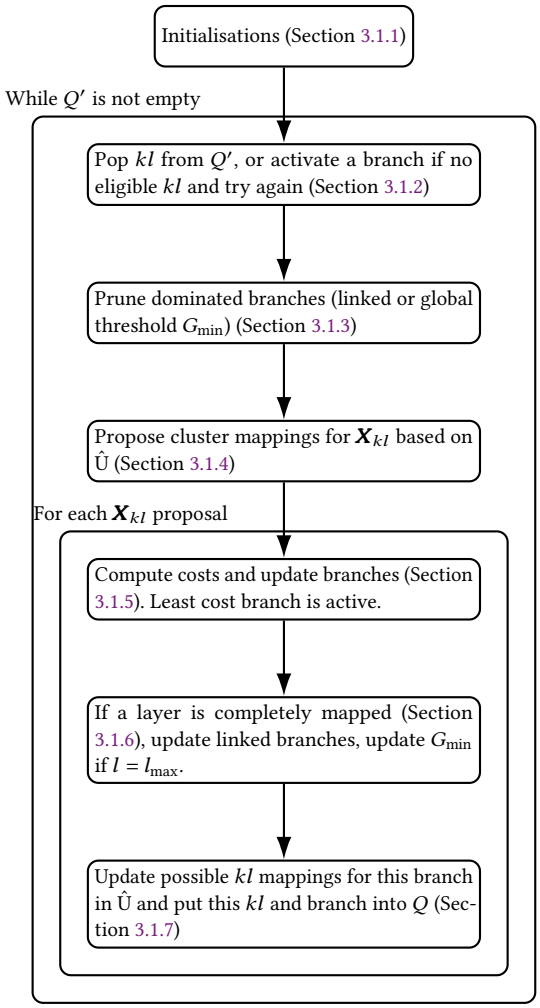


Fig. 2. Illustration of the high level components of the proposed algorithm DCMAP.

Note that $m_{l'}$ is the number of unique clusters at layer l'. In the BN inference context (Section 2.2 and (3), cost dependency arises from propagation between child cluster-layers $Child(\boldsymbol{X}_{kl'})$, their cluster mapping, and a parent cluster-layer kl' and its cluster mapping; potentially, they could be of the same cluster.

Equation (10) provides the basis for our algorithm. However, in algorithms like A* (LaValle 2006), potential actions (i.e. cluster mappings) are evaluated prospectively using the objective function at each iteration. This rapidly becomes infeasible in our cluster mapping context due to the combinatorial explosion of mappings over multiple layers $u(\boldsymbol{X}_{l+1,\ldots,L})$. One approach to overcome this challenge is to instead apply our objective function (10) retrospectively after potentially multiple iterations of the algorithm, to prune dominated branches (Section 3.2 and Lemma 3.2).

Definition 6. *A cluster mapping solution for branch i on layer l is dominated by another branch j iff $G_i(\boldsymbol{X}_l) > G_j(\boldsymbol{X}_l)$.*

Here, it is necessary to have an alternate means for prioritising cluster-layer proposals to update at each iteration. Two key heuristics are used: (a) an upper bound estimate of the total cost $\hat{g}$, and (b) adaptive constraint and release of the search space (Section 3.1.2).

Recall that $l_{\max}$ is the last layer of the DAG, hence $\hat{g}$ is an estimate of the total or global cost $G(\boldsymbol{X}_{l_{\max}})$, comprising the objective function cost for this branch $G(\boldsymbol{X}_{l-1})$, the cost for clusters in layer l, and a heuristic estimate of the remaining cost:

$$\begin{aligned}\hat{g}(\boldsymbol{X}_{kl}) = G(\boldsymbol{X}_{l-1}) + \\ \sum_{k \in M_l} c\Big(\boldsymbol{X}_{kl'}, u(\boldsymbol{X}_{kl'}), Child(\boldsymbol{X}_{kl'}), u\big(Child(\boldsymbol{X}_{kl'})\big)\Big) + \\ h\Big(\boldsymbol{X} \setminus \big(\boldsymbol{X}_{kl:k \in M_l} \cup \boldsymbol{X}_{0..l-1}\big)\Big)\end{aligned} \tag{11}$$

where M_l is a set of clusters at layer l that have been updated, $\setminus$ is set difference, and h is a heuristic cost that is an upper bound for all the remaining, non-updated nodes. We refer to nodes for which the cost has been calculated as updated or popped. Potentially, there may be some nodes remaining in layer l that have not been updated which are estimated as part of the h term (11). Note that it is only possible to update cluster-layers at layer l when all nodes in layer $l-1$ and earlier layers have been updated at least once.

2.6 Search Space Size

Using the concept of layers, we can approximate the size of the search space of our cluster mapping problem. Naively, a DAG with n nodes could be mapped to $k = 1..n$ clusters thus resulting in n^n possible mappings. However, due to the requirement for cluster contiguity (Definition 3), there are many infeasible proposals. A simple approach to ensuring contiguity is to either map a node to a new cluster unique to itself, or to an existing cluster of a child node (Section 3.1.4). The overall search space size η arises from the combination of layer by layer cluster mapping combinations, i.e., the product of products (12).

$$\eta = \prod_{l=1..l_{\max}} \prod_{X_i \in \boldsymbol{X}_l} \big(|Child(X_i)| + 1\big) \tag{12}$$

As might be expected, the search space grows exponentially with layers, and the out-degree of the DAG can also significantly grow the size of the search space (Buluç et al. 2016). In- and out-degree refer to the number of parent or child nodes, respectively (Buluç et al. 2016). For instance, the case study DBN (Wu, Caley, et al. 2018), with small world connectivity, has just 25 nodes but a search space of 9.9 billion mapping combinations. This study and corresponding DCMAP solution are described in Section 4.

3 Algorithm and Analysis

This section formalises the components of the algorithm in Fig. 2, and proves the optimality of the method.

3.1 Algorithm Components

Our method uses $\hat{g}(\boldsymbol{X}_{kl})$ to prioritise cluster-layer proposals to update (i.e. pop), and $G(\boldsymbol{X}_l)$ to facilitate local optimisation by identifying and cutting off further exploration of dominated solutions. It iterates from layer 0 (i.e. leaf nodes) through to $l_{\max}$, storing unique and non-dominated cluster mappings on separate search branches. The algorithm terminates if, given current cluster-layer proposals on the queue and additive costs, the current costs of these potential solutions exceed the least-cost solutions found so far. Potentially, near-optimal solutions can be returned at earlier iterations of the search.

Table 2. Definition of data structures. We use \$ to denote a named field within a data structure; for example, the layer field l within a popped queue entry q is denoted $q\$l$.

Data Structure	Definition
$\hat{\mathrm{U}} \in \mathbb{R}^{B\times n\times n}$	Proposed cluster mappings before updates, where $\hat{\mathrm{U}}[j,k,i]=1$ indicates a proposal of cluster k for node i on branch j.
$\mathrm{U} \in \mathbb{R}^{B\times n}$	Cluster mapping $\mathrm{U}[j,i]=k$ after updates are made at each iteration.
$\mathrm{Br}_{\mathrm{lk}} \in \mathbb{R}^{B\times l_{\max}\times B\times l_{\max}}$	$\mathrm{Br}_{\mathrm{lk}}[i,l,j,l]=1$ indicates a link at layer l between branches i and j.
$\mathrm{Br}_{\mathrm{pl}} \in \mathbb{R}^{B}$	Stores l^*+1 where l^* is the highest layer where all nodes have been updated at least once on branch b.
$\mathrm{G} \in \mathbb{R}^{B\times l_{\max}}$	Current objective function costs $G(\boldsymbol{X}_l)$ (10) for each branch and layer.
$\mathrm{G}_{\min}$	Current minimum total cost $G(\boldsymbol{X}_{l_{\max}})$.
Q, Q'	A queue with entries $q=\{b, u(\boldsymbol{X}_{kl}), l, \hat{g}(\boldsymbol{X}_{kl}), G(\boldsymbol{X}_l)\}$. We denote Q' as eligible entries for updating (popping) $q\in Q : \mathrm{Br}_{\mathrm{pl}}[b]\geq q\$l$, aka activated branches.

3.1.1 Data Structures and Initialisations. The main data structures are sparse matrices, which only store non-zero entries and assume zero otherwise for memory efficiency. The j^{th} row (corresponding to branch j), k^{th} column (cluster k) and i^{th} (node i) slice of a three-dimensional matrix $\mathcal{M}$ is indexed by $\mathcal{M}[j,k,i]$. The structures are defined in Table 2. Note the distinction between proposed versus realised cluster mappings with associated updated costs stored in $\hat{\mathrm{U}}$ and U, respectively. $\mathrm{Br}_{\mathrm{lk}}$ is used for pruning linked branches that are dominated and $\mathrm{Br}_{\mathrm{pl}}$ for tracking (using layers) which queue entries on that branch are eligible to be popped.

At initialisation, the objective functions is set to an heuristic estimate of the total cost so that no time is wasted exploring branches that exceed this cost. The possible cluster mappings for the leaf nodes are updated $\hat{\mathrm{U}}$ for branch 1, and these cluster-layers are put onto the queue for updating (Component Data Structures and Initialisations).

1: **procedure** INITIALISATIONS
2: $\quad \mathrm{G}[1,0..l_{\max}] \leftarrow h(\boldsymbol{X}_{l=0..l_{\max}})$, $\mathrm{G}_{\min} \leftarrow \mathrm{G}[1,l_{\max}]$
3: $\quad \hat{\mathrm{U}}\left[1,1..|\boldsymbol{X}_{\mathrm{leaf}}|,\boldsymbol{X}_{\mathrm{leaf}}\right] \leftarrow f\ (1..|\boldsymbol{X}_{\mathrm{leaf}}|)$
4: $\quad$ Put $\left\{1, \hat{\mathrm{U}}[1,1..|\boldsymbol{X}_{\mathrm{leaf}}|, \boldsymbol{X}_{\mathrm{leaf}}, 0, \hat{\mathrm{G}}(1,\boldsymbol{X}_{\mathrm{leaf}}), \mathrm{G}[1,0]\right\}$ onto Q
5: **end procedure**

3.1.2 Pop Queue or Activate Branch. Potentially, Q could be treated as a heap to maximise computational efficiency (LaValle 2006). We use $Pop()$ to refer to removing the lowest cost queue entry, choosing between equicost entries arbitrarily (Component Pop Queue or Activate Branch). If there are no eligible entries on the queue, i.e. no branch has $\mathrm{Br}_{\mathrm{pl}}\geq l$, then a branch is chosen for activation. This makes queue entries on that branch potentially eligible. Activation randomly switches between using $Peek()$ to return, but not remove, the lowest cost entry, or $Peek'()$ to return an entry at the lowest layer on the queue, choosing arbitrarily between entries at the same layer. This facilitates switching between best-first and breadth-first search (Theorem 3.1). After activation, DCMAP tries again to pop the queue if it is not empty; the algorithm only proceeds to the next step (Component Prune Linked Dominated Branches) if a queue entry was successfully popped.

3.1.3 Prune Linked Dominated Branches. Component Prune Linked Dominated Branches is essential to achieving computational efficiency in dynamic programming, and effectively removes any entries on the queue relating to that branch. Note that pruning of unlinked branches also happens based on the global cost (Section 3.1.6).

procedure POP QUEUE OR ACTIVATE BRANCH
  $q = \{b \leftarrow b, u(\boldsymbol{X}_{kl}), l, \hat{g}(\boldsymbol{X}_{kl}), \mathrm{G}\} \leftarrow Pop(Q')$
  if $q = \emptyset$ **then**
    $q \leftarrow Peek(Q)$ with probability α, otherwise $q \leftarrow Peek'(Q)$
    $\mathrm{Br}_{pl}(q\$b) \leftarrow |\mathrm{Br}_{pl}(q\$b)|$
  end if
end procedure

procedure PRUNE LINKED DOMINATED BRANCHES
  $\boldsymbol{B} \leftarrow j : \mathrm{Br}_{\mathrm{lk}}[b, l, j, l] = 1 \wedge \mathrm{G}[j, l] > 0$
  $\forall j \in \boldsymbol{B} : G[j, l] > \min(\mathrm{G}[\boldsymbol{B}, l], \mathrm{G}_{\min})$, prune j from $\mathrm{Br}_{\mathrm{pl}}$, Q
end procedure

3.1.4 Propose Cluster Mappings. The requirement for contiguous clusters (Definition 3) means that a node can only be mapped to a new cluster u_{new}, or the cluster of one or more child nodes $\mathbf{u}_{\text{exist}}$. For the former, we can ensure uniqueness and avoid accidentally mapping to an existing cluster by associating a unique label $1..n$ with each node. For the latter, define a priori a $n \times n$ matrix $S : S[i, j] = \{0, 1\}$, where 1 indicates that node X_i can potentially be mapped to the same cluster as node X_j. To ensure contiguity, Algorithm 2 works backward by identifying, for a given node X_i, which nodes X_j at layer $l(X_i)$ or earlier layers can be in the same cluster as X_i that are connected by a common parent node. Note that descendants are child nodes, child of child nodes and so on. For node X_j, $u(X_{i'}) : S[i', j] = 1$ are all the node clusters that X_j could be mapped to. To preserve contiguity, node X_j must be such that there are no paths $\boldsymbol{X}_p$ following the directed arcs between X_i and X_j containing any other nodes i.e. $\boldsymbol{X}_{p,ij} = \{X_i, X_j\}$ or $\boldsymbol{X}_{p,ij} = \emptyset$. The former implies X_j is a parent of X_i, and the latter is where both nodes are at the same layer, $l(X_j) = l(X_i)$.

Algorithm 2 Identifying nodes that can be in the same cluster as a given node X.

procedure SAMECLUSTER
  $S[1..n, 1..n] \leftarrow 0$
  for i in 1..n **do**
    $\boldsymbol{X'} \leftarrow \left\{Par(X_i), Descendants\big(Par(X_i)\big)\right\}$
    $S[i, j] \leftarrow 1, \forall X_j \in \boldsymbol{X'} : \max(|\boldsymbol{X}_{p,ij}|) = 2 \vee l(X_j) = l(X_i)$
  end for
end procedure

Lemma 3.1. *When applied iteratively and incrementally from layer* 0 *to* l_{max} *(Fig. 2), all feasible contiguous cluster mappings (Definition 3) can be generated using matrix S (Algorithm 2) to identify all nodes* X_j *that can be mapped to the same cluster as node* X_i*.*

PROOF. The assumption of iterating incrementally over $l = 0, 1, \ldots, l_{\max}$ means that at layer l, only nodes at layer l or earlier need to be considered for mapping to the same cluster. Additionally, Algorithm 2 only allows nodes X_i and X_j to be mapped to the same cluster if they are directly connected (i.e. a parent or child of the parent) with no paths going through an intermediary node; this guarantees contiguity, even if X_i and X_j are not in successive layers (Remark 2.1). At each iteration, feasible cluster mappings for node X_j are the clusters of nodes X_i associated with non-zero rows of $S[, j]$, which might be in the same layer or at a child layer. By Definition 1 of layers, there must exist at least one parent-child path between each node in one layer and the previous layer, hence, there are no contiguous cluster mappings that cannot be generated by iteratively and incrementally applying S from $l = 0$ to $l = l_{\max}$. □

We demonstrate Algorithm 2 to find S for the simple BN example in Fig. 1, comprising three layers and seven nodes:

$$\begin{bmatrix} & A & B & C & D & E & F & G \\ A & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ D & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ F & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ G & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{13}$$

For instance, node F (row F) could share the same cluster as nodes A, D and itself (Algorithm 2). Thus, when updating (popping) node D (column D) as the search progresses from layer 0 to 2, it could be mapped to the current cluster of node F or G or itself. Locally, this could lead to contiguous clusters involving D: (A, F, D, G), (A, F, D), (F, D), (D, G), (D).

The above constrained space of feasible cluster mappings could be pre-computed at initialisation and are enacted in Component Update Possible Cluster Mappings and Queue through $\hat{\mathrm{U}}$. Component Propose Cluster Mappings generates from $\hat{\mathrm{U}}$ and particular k and l the cluster-mappings for cost evaluation at each iteration. Intuitively, for a given branch and layer l, $\boldsymbol{Z}_1$ represents the nodes that cannot be in any other cluster $k' \neq k$, $\boldsymbol{Z}_3$ are nodes that have already been mapped to another cluster k', and $\boldsymbol{Z}_2$ are the nodes that could be in k or another cluster k'. Each of the N unique cluster mapping combinations are stored on separate branches, producing $N-1$ new branches, done by updating data structures and duplicating queue entries from the current branch b for new branches b'.

---

procedure PROPOSING CLUSTER MAPPINGS
  $\boldsymbol{Z} \leftarrow \{X \in \boldsymbol{X}_l : \hat{\mathrm{U}}[b, u(\boldsymbol{X}_{kl}), X] = u(\boldsymbol{X}_{kl})\}$
  $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\} \leftarrow$ GENERATECOMBOS$(\boldsymbol{Z}, \mathrm{U}, \hat{\mathrm{U}})$
  for i in 2..N **do**
    Make new branch b': Û,U, G, $\mathrm{Br}_{\mathrm{pl}}(b') \leftarrow -l$
    Duplicate $q' \in Q : q'\$b = b, q'\$l = l$ with $q'\$b = b'$
  end for
end procedure
procedure GENERATECOMBOS$(\boldsymbol{Z}, \mathrm{U}, \hat{\mathrm{U}})$
  $\boldsymbol{Z}_2 \leftarrow \boldsymbol{Z} \setminus \{\boldsymbol{Z}_1, \boldsymbol{Z}_3\} : \left(\mathrm{U}[b, \boldsymbol{Z}_3] \neq u(\boldsymbol{X}_{kl}), U[b, \boldsymbol{Z}_3] \neq 0\right), \left(\forall \hat{\mathrm{U}}[b, u \neq u(\boldsymbol{X}_{kl}), \boldsymbol{Z}_1] = 0\right)$
  $\boldsymbol{X}_{i=1..N} \leftarrow \left\{\boldsymbol{z}_j \cup \boldsymbol{Z}_1 \ \forall \boldsymbol{z}_j \in \mathrm{Combinations}\left(\binom{\boldsymbol{Z}_2}{1..|\boldsymbol{Z}_2|}, \boldsymbol{Z}_1\right)\right\}$
end procedure

Consider a simple example for a given branch and layer with the following possible cluster mappings for each node:

$$\hat{\mathrm{U}}\left[b, l(\boldsymbol{X}_i) = l\right] = \begin{bmatrix} k & X_1 & X_2 & X_3 \\ 2 & 0 & 1 & 1 \\ 3 & 1 & 1 & 1 \end{bmatrix} \tag{14}$$

The feasible combinations are:

$$u(X_1, X_2, X_3) = \{(3,2,2), (3,2,3), (3,3,2), (3,3,3)\}, \tag{15}$$

noting that X_1 can only be mapped to cluster 3; each combination is evaluated on a separate branch.

3.1.5 Compute Costs. Each cluster-mapping proposal is evaluated with respect to the objective function (Component Compute Costs) and the cluster mapping proposal is stored via U.

procedure COMPUTE COSTS
  Compute $\hat{g}(\boldsymbol{X}_i)$ and $\mathrm{G}[b', l] \leftarrow G(\boldsymbol{X}_l)$ (10) and (16)
  Update $\mathrm{U}[b', \boldsymbol{X}_i] = u(\boldsymbol{X}_{kl})$
end procedure

3.1.6 Layer Updates. When all the nodes in a single layer are mapped to clusters, it is possible to update $G^*(\boldsymbol{X}_L)$ (10) given the unique cluster mapping on this branch (Lemma 3.3). Component Layer Updates potentially: (i) makes queue entries at layer $l+1$ eligible to update via $\mathrm{Br}_{\mathrm{pl}}$, (ii) if $l = l_{\max}$, updates the global minimum objective function cost for global pruning, and (iii) links to other branches with the same cluster mapping for layer l for pruning dominated branches, which occurs at the beginning of each iteration.

procedure LAYER UPDATES
  if $\forall X \in \boldsymbol{X}_l : \mathrm{U}[b', X] \neq 0$ **then**
    Update $\mathrm{Br}_{\mathrm{pl}}(b') = l + 1$
    if $l = l_{\max}$ **then** $\mathrm{G}_{\min} = \min\left(\mathrm{G}_{\min}, \sum_{l=0}^{l_{\max}} \mathrm{G}[b', l]\right)$
    if $\forall b'' : \mathrm{Br}_{\mathrm{pl}}(b'') \geq l + 1 \wedge \mathrm{U}[b'', \boldsymbol{X}_l] = \mathrm{U}[b', \boldsymbol{X}_l]$ **then** LINK $(b', b'', l+1)$
  end if
end procedure

3.1.7 Update Possible Cluster Mappings and Queue. The final Component Update Possible Cluster Mappings and Queue uses the constrained set of possible cluster mappings (Section 3.1.4) when updating $\hat{\mathrm{U}}$ for parents of nodes in $\boldsymbol{X}_{kl}$. In addition, it places the possible cluster-layers associated with these parent nodes onto the queue Q.

procedure UPDATE POSSIBLE CLUSTER MAPPINGS AND QUEUE
  Generate $u\big(X \in Par(\boldsymbol{X}_i)\big) : u(X) = \{u_{\mathrm{new}}(X), \boldsymbol{u}_{\mathrm{exist}}(X)\}$ (Algorithm 2)
  $\hat{\mathrm{U}}\big[b', u\left(Par(\boldsymbol{X}_i)\right), Par(\boldsymbol{X}_i)\big] \leftarrow 1$
  Put $\{b', u(Par(\boldsymbol{X}_i), l, \hat{g}(\boldsymbol{X}_i), \mathrm{G}[b', l]\}$ onto Q
end procedure

3.2 Theoretical Analysis

This section proves analytically that DCMAP is optimal in the sense that it finds all least cost cluster mappings, and does so such that an equally informed algorithm cannot do so in fewer iterations. This is achieved through conditional optimality (derived in Section 3.2.1), the exploration of all possible contiguous cluster mappings through branching (Section 3.2.2), and combined to show the optimality of DCMAP through evaluating non-dominated branches, i.e. potential cluster mapping solutions, at the earliest opportunity (Section 3.2.3). An additional property is derived for super additivity of cost functions for a heuristic to further speed up the search (Appendix B).

3.2.1 Conditional Optimality. Foundational to DCMAP is equation (10) and conditional optimality of a layer cluster mapping, denoted as $u(\boldsymbol{X}_l) = U_l$ for short; the cluster mapping for layers $l' = 0, \ldots, l-1$ is denoted as $U_{<l} = U_{l-1}, U_{l-2}, \ldots$, and optimal mappings or costs are denoted with a superscript asterisk, e.g. G^*.

Definition 7. *Given a layer cluster mapping U_l, define $U^*_{<l}$ as a conditionally optimal or conditionally non-dominated cluster mapping iff $G^*(\boldsymbol{X}_l | U_l, U^*_{<l}) \leq G(\boldsymbol{X}_l | U_l, U^i_{<l})$ for all possible cluster mappings of earlier layers $U^i_{<l}$.*

Following Definition 6, we say that $U^*_{<l}$ dominates $U^i_{<l}$ if $G^*(\boldsymbol{X}_l | U_l, U^*_{<l}) < G(\boldsymbol{X}_l | U_l, U^i_{<l})$.

Lemma 3.2. *A globally optimal cluster mapping $U^\star = u^\star(\boldsymbol{X})$, where $G^\star(\boldsymbol{X}|U^\star) \leq G(\boldsymbol{X}|U^i)$ for all other cluster mappings U^i, is composed of conditionally optimal $G^*(\boldsymbol{X}_l|U_l, U^*_{<l})$, layer-based sub-mappings $(U_l, U^*_{<l})$ for each of $l = 0, \ldots, l_{max}$; i.e. every layer of U^* is conditionally optimal.*

Proof. Equation (10) was derived from dynamic programming to find globally optimal cluster mappings based on optimal mappings between layer l and its latest parent layer L. This could be re-written in terms of conditional optimality $G^*(\boldsymbol{X}_{l+1}|U_{l+1}, U^*_{<l+1})$ of a mapping at layer $l+1$ which, given a positive, non-zero cost $c(\cdot)$, can only be conditionally optimal if $G^*(\boldsymbol{X}_l|U^*_l, U^*_{<l})$ is also conditionally optimal, by definition.

$$G^*(\boldsymbol{X}_{l+1}|U_{l+1}, U^*_{<l+1}) = G^*(\boldsymbol{X}_l|U^*_l, U^*_{<l}) + \sum_{k=1}^{m_{l+1}} c\Big(\boldsymbol{X}_{kl+1}, u(\boldsymbol{X}_{kl+1}), Child(\boldsymbol{X}_{kl+1}), u\big(Child(\boldsymbol{X}_{kl+1})\big)\Big) \tag{16}$$

We now prove Lemma 3.2 by induction. By Definition 7, the globally optimal objective function cost is equal to the conditionally optimal cost of the last layer $G^\star(\boldsymbol{X}|U^\star) = G^*(\boldsymbol{X}_{l_{\max}}|U^*_{l_{\max}}, U^*_{<l_{\max}})$. Note that conditional optimality of $U^*_{<l_{\max}}$ is defined for a given cluster mapping at the specified layer, hence $U^*_{l_{\max}}$ refers to the cluster mapping at layer $l_{\max}$ that minimises G globally. By equation (16), a cluster mapping with conditional optimality $G^*(\boldsymbol{X}_{l+1}|U^*_{l+1}, U^*_{<l+1})$ at layer $l+1$ requires a conditionally optimal cluster mapping at layer l . Therefore, by induction, all layers of an optimal cluster mapping must themselves be conditionally optimal.

□

3.2.2 Branching. The exploration of all non-dominated cluster mapping solutions, one stored on each branch, is shown through Lemma 3.3.

Lemma 3.3. *DCMAP can generate all possible contiguous and feasible cluster mapping solutions with each unique mapping stored on a separate branch b. A cluster mapping is feasible if every node is mapped to exactly one cluster.*

Proof. It follows directly from Component Propose Cluster Mappings that, given potential cluster mapping proposals for layer l where $\hat{\mathrm{U}}[b, k,] = 1$, all feasible and unique cluster mappings are generated and stored on separate branches. Note that, on this branch, any nodes that have already been mapped to a cluster are excluded from the generated mappings via $\boldsymbol{Z}_3$ for feasibility; any nodes that must be mapped to k and have no other mapping proposals are captured via $\boldsymbol{Z}_1$ for feasibility; and combinations of inclusion and exclusion of the remaining nodes are captured via $\boldsymbol{Z}_2$.

From Lemma 3.1, an algorithm that iterates from $l = 0$ through to $l = l_{\max}$ using S from Algorithm 2 will generate all possible contiguous cluster mapping proposals over successive layers, which are stored in $\hat{\mathrm{U}}$ (Component Update Possible Cluster Mappings and Queue). Within each cluster-layer inside each of these layers, Component Propose Cluster Mappings generates all feasible and unique cluster mappings and stores them onto separate branches at each iteration (i.e. cluster-layer). Hence, it follows that DCMAP can generate all possible feasible and contiguous cluster mappings, with each unique mapping stored on a separate branch. □

3.2.3 Optimality. We show that DCMAP will find all the least cost cluster mapping solutions, and that this is done such that an equally informed algorithm cannot do so in fewer iterations in the absence of branch activation (i.e. $\mathrm{Br}_{\mathrm{pl}}(b') \leftarrow l$ in Component Propose Cluster Mappings). Then, we show the efficiency of branch activation when there are many branches of equal estimated total cost $\hat{g}$.

Theorem 3.1. *DCMAP will incrementally explore branches (i.e. cluster mapping solutions) from layer 0 through l_{max} to find every least cost $G^\star(\boldsymbol{X}, U^\star_i)$ solution.*

Proof. Each iteration of DCMAP corresponds to an exploration of a cluster-layer mapping kl. When an iteration produces a complete cluster mapping $\mathrm{U}[b', \boldsymbol{X}_l]$ for layer l that is the same as one or more branches

b'', a link is made between these branches in the same iteration. By Definition 7 of conditional optimality, any dominated branches sharing the same cluster mapping for that layer $\mathrm{U}[b', \boldsymbol{X}_l]$ can be pruned since by Lemma 3.2, an optimal cluster mapping comprises layer-by-layer conditionally optimal cluster mappings. Pruning of all dominated linked branches happens at the beginning of an iteration (Component Prune Linked Dominated Branches), so no iterations are wasted exploring a dominated branch.

In addition to conditional optimality, there is also outright global optimality as captured in $\mathrm{G}_{\min}$, which affects all branches including those that are not linked. This is updated in the same iteration as when a branch generates a lower cost cluster mapping for the entire BN (Component Layer Updates). As the objective function cost is non-negative, additive and time-invariant, any branches that exceed $\mathrm{G}_{\min}$ are globally dominated, even if they have not yet reached $l_{\max}$, since they cannot possibly lead to a lower cost solution. These are also pruned at the begining of each iteration (Component Prune Linked Dominated Branches).

By Lemma 3.3, DCMAP can generate all possible unique solutions on each branch, but all dominated branches and globally dominated branches are pruned in the process, leaving optimal solutions at algorithm termination, i.e. when the queue is exhausted. It follows that the optimal number of clusters m (2) are also determined. Branch activation, which only affects the eligibility of kl proposals for popping based on $U_{<l}$ already mapped layers (i.e. Q'), does not affect branch linking and pruning and thus optimality of solutions are preserved. □

Lemma 3.4. *In the absence of branch activation, let D be an algorithm no more informed than DCMAP that also searches cluster-layers using conditional optimality to find all optimal cluster mappings of a BN. Assume that the objective function and costs are identical: if a cluster-layer was popped by DCMAP, it was also popped by D.*

Proof. Suppose the contrary where there is at least one cluster-layer $\kappa\lambda$ on branch β popped by DCMAP but not by D. This can only happen if branch β was pruned by D as a dominated branch at an earlier iteration, otherwise D cannot be guaranteed to find least cost cluster mappings. If β was dominated at an earlier layer $l' < \lambda$, then DCMAP would have found this when completing the mapping of layer l' (Component Layer Updates), and pruned branch β at the earliest opportunity thus never popping $\kappa\lambda$, contradicting the assumption on $\kappa\lambda$ for DCMAP. If β is dominated at layer λ, this implies $\kappa\lambda$ completes layer λ and D can only determine this to be dominated by popping $\kappa\lambda$, contradicting the $\kappa\lambda$ assumption for D, hence completing the proof. □

Therefore, it is not possible to find, using an equally informed algorithm, all the least cost cluster mappings in fewer iterations than DCMAP in the absence of branch activation. Examples of alternate algorithms for cluster mapping include tree search algorithms like breadth first or depth first search (LaValle 2006). However, the number of iterations needed to find the first optimal solution is also important.

Consider the impact of pruning on search iterations. By Definition 7 of conditional optimality, pruning of dominated branches can only occur when a layer is completed (Component Layer Updates). Therefore, changing the order of updating of cluster-layers through branch activation in DCMAP has no effect on the total number of search iterations; it is already optimal (Lemma 3.4). However, branch activation, which evaluates one search branch to completion at a time, can potentially reduce the number of iterations to find the first least cost solution. This occurs as cluster-layers in a branch are updated in a semi-greedy ordering since each iteration evaluates cluster-layer proposals with the lowest $\hat{g}(\boldsymbol{X}_i)$ first (Component Compute Costs).

Branch activation is especially relevant when there are many queue entries of equal estimated global cost $\hat{g}$, which arises from the combinatorial explosion of possible cluster mappings. Consider an order of magnitude approximation of this. Specifically, assume on average $\overline{N}_K$ cluster-layer proposals $k'l'$ at $\overline{N}_P$ parent layers $l' > l$ over N branches (Component Update Possible Cluster Mappings and Queue). Note that whenever a new branch b' is made from b, queue entries for branch b are copied onto b'. As a result, the expected number of entries being placed on the queue $\overline{N}_T$ at layer l that share the same $\hat{g}$ as, on average, $\overline{N}_K\overline{N}_P$ other entries, is approximately:

$$\overline{N}_T = \overline{NN}_K\overline{N}_P. \tag{17}$$

Table 3. Distribution of in- and out-degree (degree and number of nodes with that degree) in the one time-slice DBN.

In-degree	0	1	2	4	5	6	7
#nodes	9	2	8	3	1	1	1
Out-degree	0	1	2	3	4	6	9
#nodes	2	14	3	3	1	1	1

Table 4. Distribution of in- and out-degree (degree and number of nodes with that degree) two time-slice DBN.

In-degree	0	1	2	3	4	5	6	7	8
#nodes	18	4	10	6	5	2	2	2	1
Out-degree	0	1	2	3	4	6	7	8	9
#nodes	3	27	7	6	2	1	1	1	2

For example (15), there are four cluster mappings ($N = 4$), $N_K = 2$ unique clusters for the first three proposals and one for the last one; assuming $\overline{N}_P = 2$, $N_T = 14 \approx \overline{N}_T = 16$.

Remark 3.1. *Comparing DCMAP with branch activation with no branch activation, it follows from (17) that the expected speed-up to update one layer with on average $\overline{N}_K$ clusters is $O(\overline{NN}_P)$ i.e. $O(\frac{1}{\overline{NN}_P})^{th}$ the number of iterations are needed.*

Note that this result holds irrespective of the objective function being used.

4 Case Study

We use as a case study the complex systems DBN model for seagrass resilience, described briefly in Section 1. This model, which has 25 discrete nodes per time-slice (Fig. 3) and 96 time slices (Wu, Caley, et al. 2018), captures biological, ecological and environmental dynamics and their interactions to predict the impact on resilience of a series of dredging stressors and its cumulative effects over time. It demonstrates small world or power law connectivity (Table 3 and 4) with a few nodes having comparatively many parent or child nodes, and the majority having few (Watts and Strogatz 1998); most nodes had two to four states, with a few having ten or more states. (Wu, Caley, et al. 2018) incorporated expert knowledge and multiple data sources (e.g. experimental and monitoring data) to inform the model; however, the computation time prohibited the use of Bayesian inference, which enables better data integration and estimation of uncertainty.

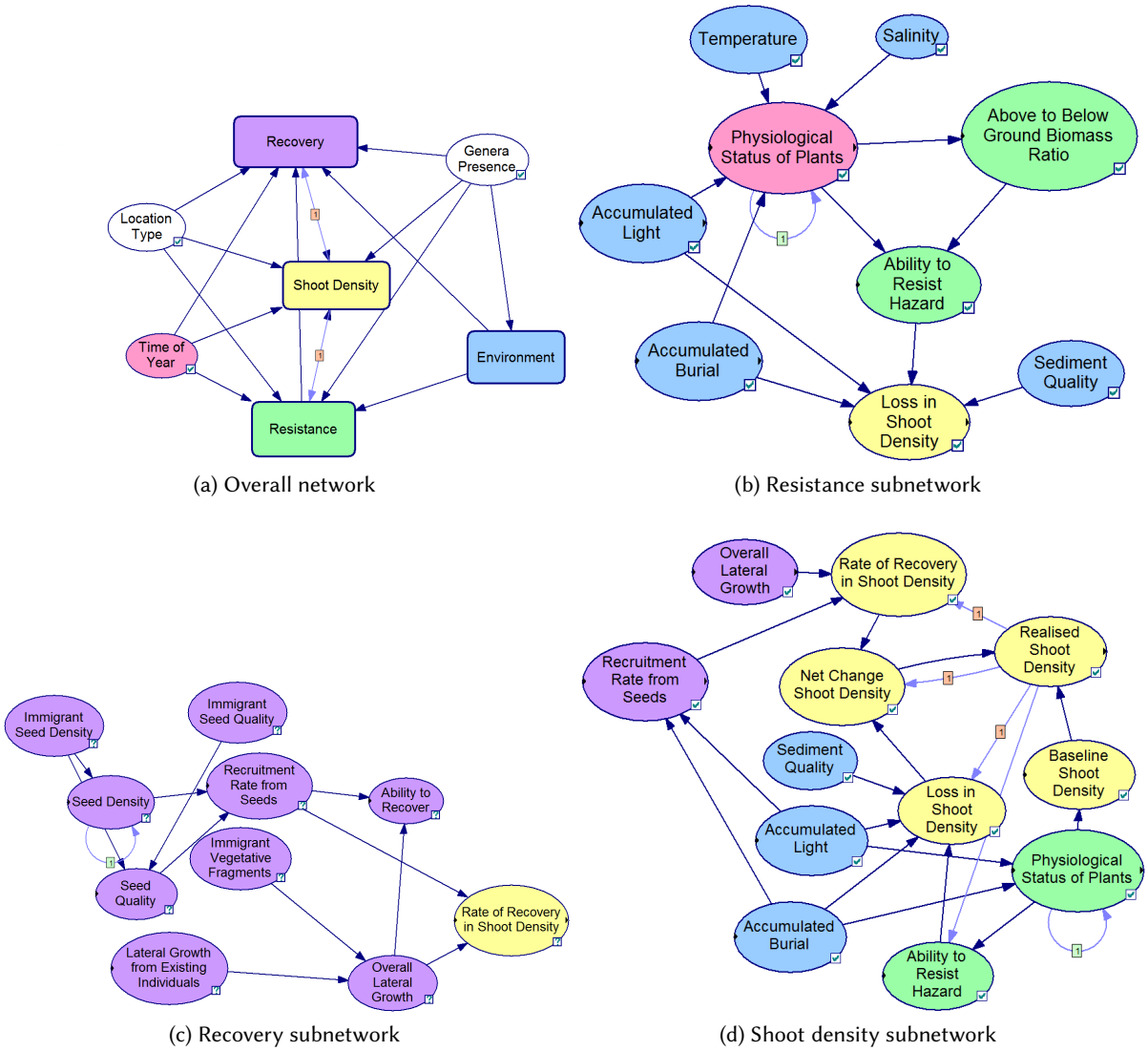
For the two time-slice version of this DBN with 50 nodes, the in- and out-degrees are even higher than the one time-slice DBN (Table 4).

4.1 Computation Cost as an Objective Function

The objective function is defined in terms of the computational cost associated with BN inference (3) as illustrated in Section 2.2. Recall that the objective function is defined recursively with respect to a cost function c (10), hence application of DCMAP to different use cases involves the definition of an appropriate cost function c.

Without loss of generality, consider a discrete BN. We formulate a backwards probability measure P^b_{kl} for computing inference akin to the rearrangement of Equation (3), which forms the basis of inference techniques such as variable elimination (Daly et al. 2011).

$$P^b_{kl} = \prod_{\forall X_i \in X_{kl}} \sum_{X_i} P\big(X_i | Par(X_i)\big) \prod_{X_{kl'} \in Child(X_{kl})} \sum_{\forall X \notin X_{kl'}} P^b_{kl'} \tag{18}$$



(a) Overall network

(b) Resistance subnetwork

(c) Recovery subnetwork

(d) Shoot density subnetwork

Fig. 3. Overall seagrass DBN model (Wu, Caley, et al. 2018) (Fig. 3a). Nodes are ovals and arrows denote conditional dependence between a parent and child node in the same time-slice. Where an arrow is labelled with a 1, the child node is in the next time-slice. Rounded rectangles denote subnetworks. Yellow nodes relate to loss and recovery in shoot density, purple nodes to recovery, green nodes to resistance, blue nodes to environmental factors and pink for all other nodes. The environment subnetwork is not shown as it is a disjoint subnetwork of the blue nodes, which are effectively inputs to the model.

P^b_{kl} comprises propagation from child cluster-layer $P^b_{kl'}$, marginalising (i.e. summing) out any nodes that are not in $\boldsymbol{X}_{kl}$, multiplication by the conditional probabilities of the nodes in $\boldsymbol{X}_{kl}$, and finally marginalisation

in preparation for propagation to the next antecedent cluster-layer. It is a recursive relation and for $l' = 0$, $P^b_{kl'} = 1$ since a cluster-layer comprising leaf nodes does not have child nodes, by Definition 1. In the context of $c\Big(\boldsymbol{X}_{kl'}, u(\boldsymbol{X}_{kl'}), Child(\boldsymbol{X}_{kl'}), u\big(Child(\boldsymbol{X}_{kl'})\big)\Big)$ (10), the limits of the products and sums incorporates the effect of cluster mappings u, and $P(X_i|Par(X_i))$ and $P^b_{kl'}$ are defined with respect to cluster-layer and child cluster-layer nodes.

Define $\mathcal{F}()$ as a normalised count of the number of clock cycles associated with performing all the multiplications and additions, where each multiplication is assumed to have a normalised count of 1 cycle, each addition 0.6 and each division (i.e. ratio) 3, based on Intel Skylake CPU latencies (Fog 2022). Hence,

$$c\Big(\boldsymbol{X}_{kl'}, u(\boldsymbol{X}_{kl'}), Child(\boldsymbol{X}_{kl'}), u\big(Child(\boldsymbol{X}_{kl'})\big)\Big) = \mathcal{F}\big(P^b_{kl}\big) \tag{19}$$

4.1.1 Results and Discussion. For one time-slice of this DBN, DCMAP finds a total of 4 optimal cluster mapping solutions, with an optimal cost of 1386.6 compared to a naive heuristic cost of 34444; the first optimal solution was found at iteration 865 and the last one at iteration 1782 (Fig. 4). Over 20 runs, the mean iteration for the first optimal solution was 856 (95% CI 852, 866). In contrast, the two time-slice DBN had 10 least cost solutions, the first found at iteration 1569 and last at iteration 6190 (Fig. 4), with an optimal cost of 6220.8 compared to the naive heuristic cost of 3148386. Over 20 runs, the mean iteration for the first optimal solution was 1568 (1566, 1581). An α probability value of 0.6 was selected to achieve a balance of best first and breadth first search with a slight preference to the former. These empirical results suggest a hypothesis that the complexity of finding the first optimal solution is $O(n^2)$ where n is the number of nodes, and that of finding all the least cost solutions is $O(an^2)$ for positive integer a; this particular case study suggests a value of $a = 3$. Additionally, DCMAP rapidly converged onto near-optimal solutions (Fig. 4), and regularly returned cluster mapping solutions (Fig. 4), on average returning a solution every 57 and 24 iterations for the one and two time-slice versions of the study.

Furthermore, we could compare the similarity of cluster mappings to optimal cluster mappings using nodes assigned to the same cluster rather than nodes assigned to the same cluster label, as labels could vary depending on its child clusters when assigning a node to an existing cluster (Section 3.1.4). This could be done by creating a $n \times n$ matrix where $Y[i, j] \in \{0, 1\}$ for each cluster mapping, where a 1 indicates that those two nodes are in the same cluster. This way, the similarity of a solution Y compared to an optimal solution Y^* could be computed as the dot product $Y \cdot Y^*$, divided by the number of ones in Y^*. Here, we use the maximum similarity between Y and the set of optimal solutions $\boldsymbol{Y}^*$. For the one and two time-slice and DBN, there was both rapid convergence in terms of cost (Fig. 4) and approximately 80% or more similarity (Fig. 5) to optimal solutions in just a few hundred iterations.

In addition to the empirical results above and theoretical computational efficiency of DCMAP (Lemma 3.4), the performance of DCMAP can be further contextualised through comparison with breadth first search using layers and clusters (Section 2.3.1 and 2.3.2). On a simple graph such as the illustrative example (Fig. 1), DCMAP and breadth first search require similar computation, finding all three optimal solutions at iterations 60, 99 and 100, versus iterations 54, 84 and 86, respectively. However, breadth first search does not scale well with graph complexity (i.e. size and layers) and is unable to find even one least cost solution for the one time-slice case study after 9600 iterations and the lowest cost solution was 1518.4; DCMAP found an optimal solution of cost 1386.6 at iteration 865 and had found all four optimal solutions by iteration 1782.

The size of the search space computed using (12) was 1.51×10^{21} and 9.91×10^9 for the two and one time-slice DBNs, respectively and, as might be expected given Lemma 3.3, all branch solutions are unique (i.e. no two branches with the same solution). Such exponential growth in search space size likely contributes to the inability for breadth first search to scale with more layers and nodes. In addition, such a large space of cluster mapping solutions contributes to a greater potential for benefit from optimisations, where the optimal solutions had 4%

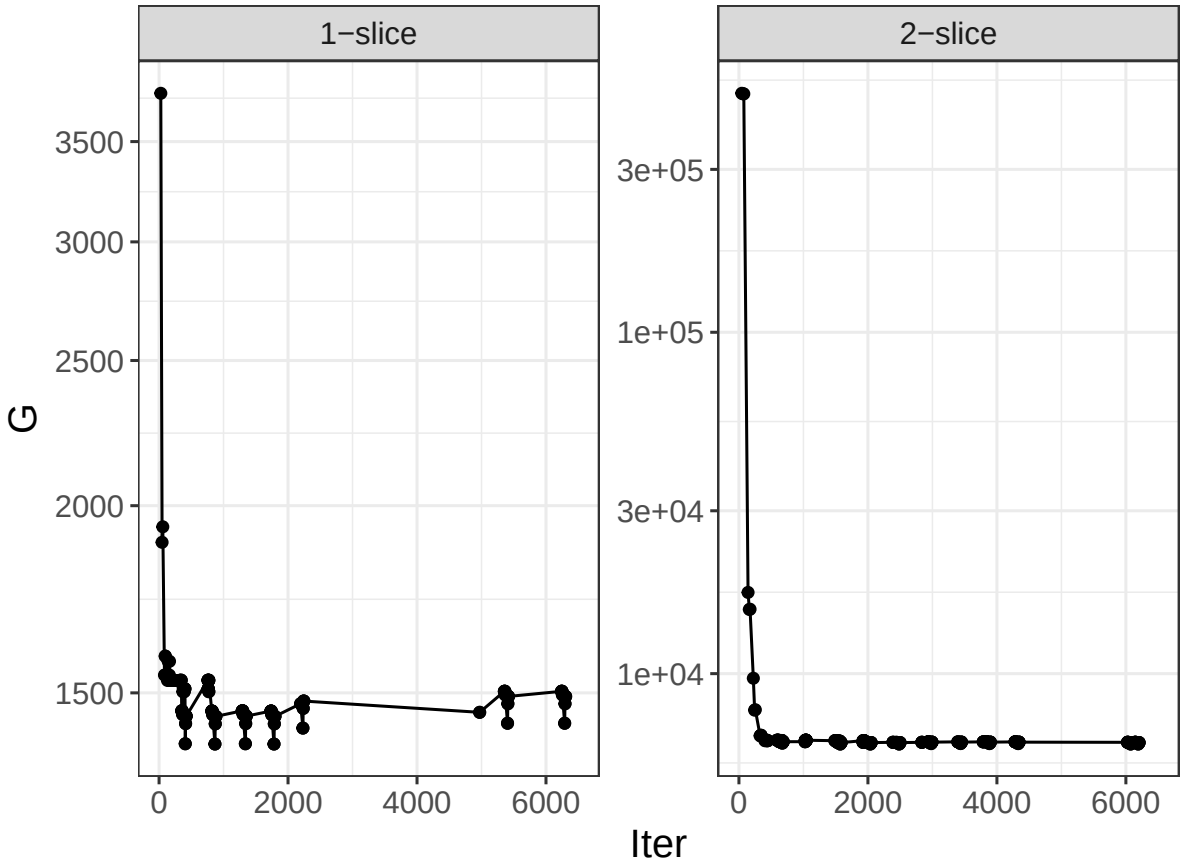


Fig. 4. Cost G of solutions found and the iteration of their discovery for both the one and two time-slice DBNs.

and 0.2% the cost of naive solutions, which corresponds to a computational speed-up of 22.8 and 426 in the one and two time-slice case studies, respectively. Therefore, DCMAP facilitates substantially more computationally efficient inference by using clustering in our complex systems seagrass DBN case study.

5 Discussion

Clustering has been applied to improve the computational feasibility of BN learning via constraints, scoring functions (i.e. using a cost function) or both (Huang and Zhou 2022), and the computational feasibilitiy of inference through heuristics and local optimisation or both (Forouzan and Ihler 2015). However, algorithms that explicitly focus on forming optimal clusters in the context of BN inference are missing. This incurs additional complexity in the form of cost dependency, where the local cost of a cluster is dependent on the mapping of connected nodes and the global cost is typically the sum of local costs. The proposed, general optimisation of BN clusters, i.e. with arbitrary criteria, advances the field by: (i) providing a benchmark for validating approximate methods against the globally optimal solution, (ii) enabling future research and development of cluster-based methods for inference

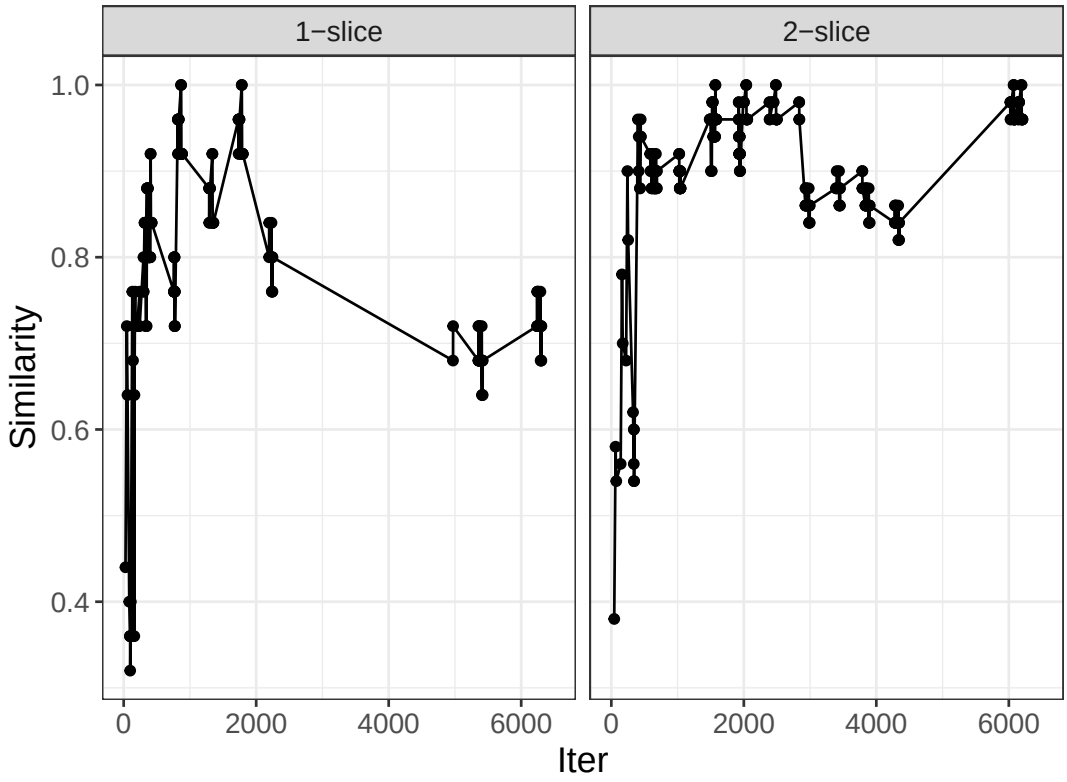


Fig. 5. Similarity of solutions to the set of optimal solutions and their iteration of discovery for both the one and two time-slice DBNs.

including anytime optimisation (Mateescu et al. 2010) and criteria such as entropy (Lin et al. 2020) (Section 5.2), and (iii) finding clusters of homogeneous regions in a system with changing dynamics (Wu, Caley, et al. 2018).

Previous BN literature has used decomposability of scoring functions, such as through additive costs, to allow for local search procedures and/or optimisation of the score function; some even use dynamic programming and graph layers (Friedman et al. 1999; Kojima et al. 2010; Lu et al. 2021). Our work builds upon the application of dynamic programming and layers in BNs to facilitate global optimisation with arbitrary non-negative cost functions and cost dependency. This latter property has not been explicitly studied in the existing BN literature.

The proposed algorithm, DCMAP, is proven to find all of the least cost cluster mapping solutions through conditional optimality of layers and branching (Section 3.2.3). In addition, it is theoretically shown that an equally informed algorithm cannot find these solutions in fewer iterations. This is important as optimal clustering is NP-hard in general (Buluç et al. 2016) and cost dependency would exacerbate this complexity. For instance, a breadth first search on the same one time-slice case study is unable to find any optimal solution even with ten times more iterations than DCMAP. This arises even though the number of iterations needed is similar to DCMAP for the simple illustrative BN (Fig. 1).

5.1 Case Study Discussion

We developed a case study of BN clustering where clusters were used as a means to compute (i.e. infer) the marginal probabilities of a complex systems seagrass DBN. By explicitly defining the cost function as the computational cost of inference given a cluster mapping, a speed-up of 22.8 and 426 times was achieved for a one and two time-slice version of the DBN, respectively. In itself, a substantial reduction of computational complexity through clustering is useful as BN inference plays a key role in prediction and model learning, and methods for that such as Markov Chain Monte Carlo (MCMC) (Gelman et al. 2013) can require thousands of iterations. MCMC can also help better quantify uncertainty for parameter and structure learning of BNs (Daly et al. 2011) and DBNs (Shafiee Kamalabad and Grzegorczyk 2020) through the Bayesian framework. Explicit optimisation of BN clusters using DCMAP could be a key enabler for future research in BN learning and updating, especially for complex systems DBNs like our seagrass case study.

Although our case study was for a discrete BN with between two and fourteen states per node with corresponding CPTs, the proposed DCMAP algorithm equally applies to discrete or continuous or hybrid BNs, DBNs, Object Oriented Bayesian Networks (OOBNs) (Koller and Pfeffer 2013) and Markov models in general, all of which can be represented as a DAG; the key adaptation required is the development of an application-specific cost function. Note that dynamic and templated models alike can be unrolled (Wu, Caley, et al. 2018) into a BN and cluster mapping performed on the result. In practice, for a k^{th} order dynamic model, running cluster mapping on k slices is sufficient to find optimal clusters that can then be replicated over all slices.

We have theoretically shown that an equally informed algorithm cannot find all optimal solutions in fewer iterations, further study is needed to better understand the computational complexity of DCMAP to find the first optimal solution and how it might vary with different BN graph structures and cost functions. Early results from the case study suggest a complexity of $O(n^2)$ iterations to find the first least cost cluster mapping and $O(an^2)$ iterations for all optimal solutions, where n is the number of nodes and a some positive integer. This aligns with previous findings of $O(n^2)$ complexity for clustering in the absence of cost dependency (Zheng et al. 2020). However, the computational complexity could potentially vary significantly depending on the cost function employed and also the structure of the network (e.g. dense versus sparse). Additionally, it could also vary with the α probability for switching between best first and breadth first search (Section 3.1.2), especially for near-optimal solutions. These are all avenues for future research.

Despite the theoretical optimality of DCMAP, scalability in practice is a challenge for large networks with thousands of nodes. Potentially, the template behind a DBN or OOBN like the two-slice DBN case study, and/or near-optimality could be investigated as approaches for large networks. Better understanding of the convergence characteristics and early termination of clustering, including in the impact of cost functions and network structures, is needed. Additionally, even though proposed cluster mappings are always unique (Lemma 3.3), label switching is an area for future study to better understand the extent of the issue and strategies for mitigation if needed.

5.2 Future Work: Entropy

In light of the generalisability of DCMAP given its ability to optimise for arbitrary, non-negative and dependent cost functions, one key area of future work is in the development of objective functions for specific usage scenarios. One such possibility is entropy, which for BNs is defined generally as the sum of the entropy associated with each scenario over all possible scenarios, where a scenario might be defined in terms of one or more nodes (Parhizkar et al. 2018; Scutari 2024). Consider a discrete BN. The entropy associated with evidence for a given

scenario indexed by k is:

$$H_k = -\sum_{k=1}^{n^k} p_k \sum_{i=1}^{n} \sum_{j=1}^{n_i^s} p_{ijk} log(p_{ijk}) \tag{20}$$

where we represent scenarios as clusters k, n^k is the total number of clusters, p_k is the marginal probability associated with that scenario, i indexes nodes and n is the total number of nodes, j indexes states where n_i^s is the number of states for node i, and p_{ijk} is the marginal probability of node i, state j and scenario k.

Consider a special case of clustering where specific evidence scenarios are to be applied at one or more nodes and at one or more points in time in a DBN, with the goal of minimising global entropy. Thus, each node can be assigned to one of two clusters: (cluster 1) observed using virtual evidence $\delta(X')$ (Pearl 1988) where each observation is independent, or (cluster 2) not observed. Note that virtual evidence provides greater flexibility in modelling uncertainty compared to hard evidence where a state is observed or not. Dependent cluster mapping is necessary to optimise over all possible combinations of observed nodes and their interdependencies, which arise from propagation of evidence for inferring marginal probabilities for each scenario p_{ijk} (20). DCMAP can approximate this marginal probability p_{ijk} for every node X_i via backwards propagation with evidence to layer l and no evidence from forward propagation:

$$p_{ijk} = \prod_{\forall X': l(X') \leq l} \sum_{X' \notin X_i} P\big(X'|Par(X')\big)\delta(X')P_{l+1}^f \tag{21}$$

where forwards propagation $P_{l+1}^f = \prod_{\forall X': l(X')>l} \sum_{X'} P\big(X'|Par(X')\big)$, which assumes no evidence since these nodes have not been mapped yet, and could be pre-computed. Due to independence of evidence, cluster proposal (Section 3.1.4) only involves assigning one node at a time $\boldsymbol{X}_{kl} = X_i$ to being observed or not, and p_k is equal to p_{ijk} in equation (21) in the absence of evidence for X_i. These enable the computation of an additive objective function G which sums entropy over all evidence scenarios k via H_k (20).

Clusters have been used to improve the computational efficiency of inference, and learn conditional probabilities and BN structures (Daly et al. 2011; Lu et al. 2021); potentially, optimal clusters can further enhance these tasks. In addition, using cost functions such as entropy, clusters could also have a practical interpretation. In non-homogeneous Hidden Markov Models or DBNs for instance, clusters could be interpreted as homogeneous portions of a non-homogeneous system, akin to changepoint detection or model switching (Boys et al. 2000; Shafiee Kamalabad and Grzegorczyk 2020). Potentially, optimisation of clusters with respect to entropy could contribute to new methods in BN learning and prediction, and is a promising area for future research that is enabled by DCMAP.

References

S. V. Albrecht and S. Ramamoorthy. 2016. "Exploiting causality for selective belief filtering in dynamic Bayesian networks." *Journal of Artificial Intelligence Research*, 55, 1135–1178.

D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner. 2013. *Graph Partitioning and Graph Clustering*. Vol. 588. AMS, Providence, Rhode Island.

A. Belfodil, A. Belfodil, and M. Kaytoue. 2019. "Anytime subgroup discovery in numerical domains with guarantees." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 500–516.

R. E. Bellman and S. E. Dreyfus. 1962. *Applied Dynamic Programming*. Princeton University Press.

B. Bidyuk and R. Dechter. 2007. "Cutset sampling for Bayesian networks." *Journal of Artificial Intelligence Research*, 28, 1–48.

R. J. Boys, D. A. Henderson, and D. J. Wilkinson. 2000. "Detecting homogeneous segments in DNA sequences by using hidden Markov models." *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 49, 2, 269–285.

A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. 2016. "Recent Advances in Graph Partitioning." In: *Algorithm Engineering: Selected Results and Surveys*. Ed. by L. Kliemann and P. Sanders. Springer International Publishing, Cham, 117–158. ISBN: 978-3-319-49487-6.

R. Daly, Q. Shen, and S. Aitken. 2011. "Learning Bayesian networks: approaches and issues." *The Knowledge Engineering Review*, 26, 02, 99–157. doi:doi:10.1017/S0269888910000251.

R. Dechter. 1999. "Bucket elimination: A unifying framework for reasoning." *Artificial Intelligence*, 113, 1–2, 41–85. doi:http://dx.doi.org/10.1016/S0004-3702(99)00059-4.

A. Fog. 2022. *Instruction Tables: : Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs*. Technical University of Denmark.

S. Forouzan and A. Ihler. 2015. "Incremental Region Selection for Mini-bucket Elimination Bounds." In: *UAI*, 268–277.

N. Friedman, I. Nachman, and D. Pe'er. 1999. "Learning Bayesian network structure from massive datasets: The" sparse candidate" algorithm." In: *Proc. Fifteenth Conference on Uncertainty in Artificial Intelligence, 1999*, 206–215.

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. 2013. *Bayesian Data Analysis*. (3rd ed.). Chapman and Hall, CRC, Boca Raton, Fla.

J. Gu and Q. Zhou. 2020. "Learning big Gaussian Bayesian networks: Partition, estimation and fusion." *Journal of machine learning research*, 21, 158, 1–31.

J. Huang and Q. Zhou. 2022. "Partitioned hybrid learning of Bayesian network structures." *Machine Learning*, 111, 5, 1695–1738.

N. K. Kitson, A. C. Constantinou, Z. Guo, Y. Liu, and K. Chobtham. 2023. "A survey of Bayesian Network structure learning." *Artificial Intelligence Review*, 56, 8, 8721–8814.

K. Kojima, E. Perrier, S. Imoto, and S. Miyano. 2010. "Optimal Search on Clustered Structural Constraint for Learning Bayesian Network Structure." *Journal of Machine Learning Research*, 11, 1.

D. Koller and A. Pfeffer. 2013. "Object-Oriented Bayesian Networks." *arXiv:1302.1554*.

F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. 2001. "Factor graphs and the sum-product algorithm." *Information Theory, IEEE Transactions on*, 47, 2, 498–519.

S. L. Lauritzen and D. J. Spiegelhalter. 1988. "Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems." *Journal of the Royal Statistical Society. Series B (Methodological)*, 50, 2, 157–224.

S. M. LaValle. 2006. *Planning Algorithms*. Cambridge University Press.

M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun. 2005. "Anytime Dynamic A*: An Anytime, Replanning Algorithm." In: *ICAPS*. Vol. 5, 262–271.

P. Lin, M. Neil, and N. Fenton. 2020. "Improved high dimensional discrete Bayesian network inference using triplet region construction." *Journal of Artificial Intelligence Research*, 69, 231–295.

N. Y. Lu, K. Zhang, and C. Yuan. 2021. "Improving causal discovery by optimal bayesian network learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 10. Vol. 35, 8741–8748.

R. Mateescu, K. Kask, V. Gogate, and R. Dechter. 2010. "Join-graph propagation algorithms." *Journal of Artificial Intelligence Research*, 37, 279–328.

A. Mrad, V. Delcroix, S. Piechowiak, P. Leicester, and M. Abid. 2015. "An explication of uncertain evidence in Bayesian networks: likelihood evidence and probabilistic evidence." *Applied Intelligence*, 43, 4, 802–824. doi:10.1007/s10489-015-0678-6.

K. P. Murphy. 2002. "Dynamic Bayesian networks: representation, inference and learning." Ph.D. Dissertation. University of California, Berkeley.

T. Parhizkar, S. Balali, and A. Mosleh. 2018. "An entropy based bayesian network framework for system health monitoring." *Entropy*, 20, 6, 416.

J. D. Park and A. Darwiche. 2004. "Complexity results and approximation strategies for MAP explanations." *Journal of Artificial Intelligence Research*, 21, 101–133.

J. Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Francisco.

M. Scutari. 2024. "Entropy and the Kullback–Leibler Divergence for Bayesian Networks: Computational Complexity and Efficient Implementation." *Algorithms*, 17, 1, 24.

M. Shafiee Kamalabad and M. Grzegorczyk. 2020. "Non-homogeneous dynamic Bayesian networks with edge-wise sequentially coupled parameters." *Bioinformatics*, 36, 4, 1198–1207.

D. J. Watts and S. H. Strogatz. 1998. "Collective dynamics of small-world networks." *Nature*, 393, 6684, 440–442.

P. P.-Y. Wu, M. J. Caley, G. A. Kendrick, K. McMahon, and K. Mengersen. 2018. "Dynamic Bayesian Network inferencing for non-homogeneous complex systems." *Journal of the Royal Statistical Society, Series C*, 67, 2, 417–434.

P. P.-Y. Wu, K. Mengersen, K. McMahon, G. A. Kendrick, K. Chartrand, P. H. York, M. A. Rasheed, and M. J. Caley. 2017. "Timing anthropogenic stressors to mitigate their impact on marine ecosystem resilience." *Nature communications*, 8, 1, 1263.

Y. Zhang, J. Liu, and Y. Liu. 2018. "Bayesian network structure learning: the two-step clustering-based algorithm." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 1. Vol. 32.

S. Zheng, X. Zhang, D. Ou, S. Tang, L. Liu, S. Wei, and S. Yin. 2020. "Efficient scheduling of irregular network structures on CNN accelerators." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39, 11, 3408–3419.

A Simple Example Comparison of Clustering and Computation Cost

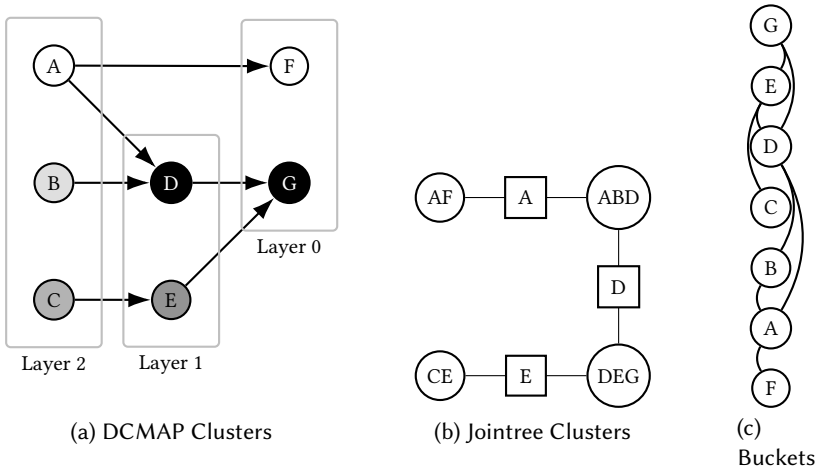
To illustrate how clustering can support DAG inference, consider an example BN (Fig. 6a) and the fundamental task of finding posterior distributions, specifically $P(X, \boldsymbol{E}), \forall X \in \boldsymbol{X}$, Equation (3). Assuming binary states for all nodes, the Conditional Probability Table (CPT) for each node with no parents (nodes A, B, C) is a 2×1 matrix, e.g. $P(A) = [p_a, p_{\bar{a}}]^T$ for states $A = a$ and $A = \bar{a}$. For E and F the CPT is a 2×2 matrix, e.g. $P(F|A) = \begin{bmatrix} p_{af} & p_{\bar{a}f} \\ p_{a\bar{f}} & p_{\bar{a}\bar{f}} \end{bmatrix}$ where p_{af} is the probability of state $F = f$ given state $A = a$. $P(D|A, B)$ is a 2×4 matrix. Multiplication of CPTs (1) containing different nodes, i.e. variables, grows the dimensionality of the resultant matrix with associated exponential growth in computation; in the discrete case, the resultant matrix contains all possible combinations of node states. We ignore evidence terms for notational simplicity, noting that evidence can be easily incorporated using $\delta(X_j)$ with node X_j (1). Using (3), the posterior probability $P(F)$ is:

$$P(F) = \sum_{A,B,C,D,E,G} P(A)P(B)P(C)P(D|A,B)P(E|C)P(F|A)P(G|D,E) \tag{22}$$

$$P(F) = \sum_{A} P(F|A)P(A) \sum_{B} P(B) \sum_{C} P(C) \sum_{D} P(D|A,B) \sum_{E} P(E|C) \sum_{G} P(G|D,E) \tag{23}$$

The computation of (22) produces a matrix of 7 dimensions, one for each node A through F, with corresponding exponential growth in memory (2^7 joint probability values) and computation associated with multiplication and marginalisation, aka product and sum. In contrast, the maximum dimensionality of (23) obtained through Bucket Elimination (BE) (Dechter 1999) is 4, occurring at bucket D (Fig. 6c). Intuitively, (22) can be rearranged, because of commutativity, to minimise computation by marginalising (summing) out nodes as early as possible (23). BE in effect employs dynamic programming to optimally compute the posterior marginal probability of a given target node, specified as part of a node ordering. In Fig. 6c, the specified order is (top to bottom) G, E, D, C, B, A, F.

Assume that the addition of two numbers has a computational cost of 0.6 compared to 1 for multiplication and 3 for divison (i.e. ratio), based on floating point operation latencies for the Intel Skylake CPU as an example



(a) DCMAP Clusters (b) Jointree Clusters (c) Buckets

Fig. 6. Example network showing nodes, directed edges, and layers (Fig. 6a). Shading denotes clusters (Fig. 6a) obtained using DCMAP, whereas the jointree (Lauritzen and Spiegelhalter 1988) comprises clusters (circles) and sepsets (squares) (Fig. 6b), and Bucket Elimination (Dechter 1999) uses buckets (Fig. 6c). Assume all nodes have binary states.

(Fog 2022)). Consider the computation of $\lambda_G(D,E) = \sum_G P(G|D,E)$ in bucket G of Fig. 6c, which is the last term in (23). Nominally, this comprises multiplication of 1 by $P(G|D,E)$ which, given every possible combination of binary states, requires $2^3 = 8$ multiplication operations and 4 addition operations (one per combination of D and E states). Thus, the output $\lambda_G(D,E)$ is computed at a cost of $8 + 6 \times 0.4 = 10.4$, and only has two dimensions (i.e. D and E) and a joint probability distribution size of 2^2. Bucket E computes $\lambda_E(C,D) = \sum_E P(E|C)\lambda_G(D,E)$, requiring $2^2 \times 2$ multiplication operations since there is an overlap in the E dimension between $P(E|C)$ and $\lambda_G(D,E)$. Marginalisation again needs 4 addition operations. The total computational cost is 64.4 to find $P(F)$ (23); finding the posterior for all seven nodes would incur a cost around 448, depending on input node ordering for BE.

As might be expected, other exact inference methods including Pearl's belief propagation and polytrees, and (Lauritzen and Spiegelhalter 1988)'s widely-used clustering algorithm share similarities with BE as all evaluate (3) exactly (Dechter 1999). However, clustering (Lauritzen and Spiegelhalter 1988) has a distinct advantage as it does not require a pre-specified node ordering and has less computational cost compared to BE when finding the posterior $P(X,\boldsymbol{E})$ for every node. This is achieved by performing inference in two steps: (i) forward-backward propagation along the jointree (Fig. 6b), and (ii) marginalisation of clusters to get $P(X,\boldsymbol{E})$; note that the posterior distribution given evidence $P(X|\boldsymbol{E})$ is obtained by normalising $P(X,\boldsymbol{E})$. The jointree, comprising clusters connected by sepsets, is derived from the DAG (Fig. 6a) using a heuristic process known as triangulation. forward-backward propagation involves: (a) projection from one cluster to a connected cluster via the sepset, and (b) absorption of the ratio of the new to old sepset into the receiving cluster. For example, projection of cluster AF to ABD (Fig. 6b) involves computing $\phi_A = \sum_A \phi_{AF}$, $\phi_{AF} = P(F|A)P(A)$ comprising one nominal multiplication by $P(A)$ then another by $P(F|A)$, then marginalisation for a cost of $2 + 4 + 2 \times 0.6 = 7.2$. Absorption into ABD updates $\phi_{ABD} = \phi_{ABD} \frac{\phi_A}{\phi_A^{\text{old}}}$, $\phi_{ABD} = P(D|A,B)$ incurring a ratio and multiplication cost of $2 \times 3 + 8 = 14$. After forward-backward propagation, the posterior for A is obtained from cluster AF by $\sum_F \phi_{AF}$ for a cost of $2 \times 0.6 = 1.2$; overall the cost to infer all nodes is just 132.8, which is substantially less than BE.

The above examples of BE and clustering (Lauritzen and Spiegelhalter 1988) show how cluster-like structures (e.g. buckets, clusters, sepsets) can facilitate more computationally efficient inference in a DAG; note that these structures differ from our definition of a cluster (Section 1). Computational efficiency is critical for MCMC-based inference (Gelman et al. 2013), as there is a need to run potentially thousands or hundreds of thousands of iterations of the models. Importantly, MCMC can help better quantify uncertainty for parameter and structure learning of BNs (Daly et al. 2011) and DBNs (Shafiee Kamalabad and Grzegorczyk 2020). Potentially, explicit optimisation of clusters using criteria associated with inference, represented as a cost, could further improve computational efficiency, make tractable MCMC for complex models, or facilitate new methods for inference such as finding homogeneous portions of a non-homogeneous system. However, this is a non-trivial task as the local cost of a cluster is dependent on the cluster mapping of connected nodes.

B Super-additive Objective Functions

A further consideration is that heuristics could also be applied to filter the proposal of cluster mappings (k,l) in addition to the prioritisation of cluster-layers for popping via $\hat{g}$. The former are typically contingent on the specific objective function being optimised. For instance, consider optimisation of DBN computation cost, where including an extra node in a cluster requires multiplication and marginalisation operations (Section 2.2) that exceed the cost of doing so separately, so super-additivity might apply:

Definition 8. *Using(10), we say that a cost function is super-additive if for any two nodes $\{X_1, X_2\}$ in the same layer l:*

$$c\left(\{X_1,X_2\},\ldots\right) > c\left(X_1,\ldots\right) + c\left(X_2,\ldots\right). \tag{24}$$

It follows from Definition 8 that:

$$
\begin{aligned}
&c\left(\{X_1, X_2\}, \ldots\right) + c\left(X_3, \ldots\right) > \\
&c\left(X_1, \ldots\right) + c\left(X_2, \ldots\right) + c\left(X_3, \ldots\right) \\
&c\left(\{X_1, X_2, X_3\}, \ldots\right) > c\left(\{X_1, X_2\}, \ldots\right) + c\left(X_3, \ldots\right) \\
&\text{therefore, } c\left(\{X_1, X_2, \boldsymbol{X}', \boldsymbol{X}''\}, \ldots\right) > \\
&c\left(X_1, \boldsymbol{X}', \ldots\right) + c\left(X_2, \boldsymbol{X}'', \ldots\right)
\end{aligned} \tag{25}
$$

where $\boldsymbol{X}', \boldsymbol{X}''$ are arbitrary sets of nodes in the same layer as X_1, X_2; potentially they could be empty sets.

Lemma B.1. *Given a super-additive cost function and any pair of root nodes $\{X_1, X_2\}$ in layer l, the least objective function cost solution $G(\boldsymbol{X}_L)$ has X_1 and X_2 in separate clusters.*

Proof. The layer costs at layers $l' > l$ are unaffected by the cluster mapping of X_1, X_2 as they do not have any parent nodes by definition. Therefore, by equation (25), any branch that contains both X_1 and X_2 in the same cluster (left-hand side) must have an objective function cost $G(\boldsymbol{X}_L)$ greater than a branch with X_1 and X_2 in separate clusters (right-hand side), completing the proof. □

Hence, for super-additive cost functions, splitting root nodes in the same layer across clusters always produces a lower cost solution. This can be used to filter cluster-layer mapping proposals as a heuristic.

C Simple Example

When run to algorithm termination (i.e. not possible to find a better solution), DCMAP takes 315 iterations to find all three optimal solutions with total cost of 54.0 and cluster mappings of:

$$
k = \begin{bmatrix}
A & B & C & D & E & F & G \\
2 & 6 & 7 & 2 & 3 & 1 & 2 \\
1 & 6 & 7 & 2 & 3 & 1 & 2 \\
5 & 6 & 7 & 2 & 3 & 1 & 2
\end{bmatrix}.
$$

Note that all three optimal solutions place nodes D and G in the same cluster (cluster 2, shaded black in Fig. 6). D is not only the node with the highest in-degree, but also connects the top part of the graph with the middle and, via node G, the bottom part of the graph. Thus it is feasible for D and G to be in the same cluster. Nodes B, C and E (shades of gray in Fig. 6) are all in individual clusters, corresponding to root (B, C) nodes or a node along a linear chain (E). As the objective function is based on computation cost, it is expected that root nodes are best assigned to their own clusters (Lemma B.1). The key variation between the three least-cost cluster mappings is the mapping of node A which can either be part of cluster 2 with D and G, part of cluster 1 with node F, or in an independent cluster of its own. These solutions were discovered at iterations 35, 182, and 276 on branches 21, 27 and 104, respectively. With a cost of 54.0, this is much less than the naive heuristic solution cost of 85.8.

However, near-optimal solutions were returned at earlier iterations:

$$
k = \begin{bmatrix}
A & B & C & D & E & F & G \\
1 & 6 & 2 & 1 & 2 & 1 & 2 \\
1 & 6 & 7 & 1 & 2 & 1 & 2 \\
2 & 6 & 7 & 2 & 2 & 1 & 2
\end{bmatrix}.
$$

A near-optimal solution that was within 6.7% of the final optimal cost was discovered at iteration 9 and had a cost of 57.6. This was refined to a cost of 57.0 at iterations 14 and 21, suggesting that the algorithm was frequently returning improved cluster mapping solutions.

Although the earlier solutions are quite different to the optimal solution, the solution at iteration 21 starts to follow the trend of the optimal solution with D, G and A in the same cluster, with other nodes mostly in their

own clusters. This suggests that finding near-optimal or even the first optimal solution could potentially be very rapid. As might be expected, however, finding all optimal solutions could be expensive (Belfodil et al. 2019; Likhachev et al. 2005). In addition, near-optimal solutions could produce diverse cluster mappings, although they converge towards optimal mapping patterns fairly rapidly.

Received 05 July 2026; accepted 16 August 2026