An Object-oriented Spatial and Temporal Bayesian Network for Managing Willows in an American Heritage River Catchment

Lauchlin Wilkinson
Faculty of IT,
Monash Univ., AUS

Yung En Chee
School of Botany,
Univ. of Melbourne, AUS

Ann E. Nicholson
Faculty of IT,
Monash Univ., AUS

Pedro Quintana-Ascencio
Department of Biology,
Univ. of Central Florida, USA

Abstract

Willow encroachment into the naturally mixed landscape of vegetation types in the Upper St. Johns River Basin in Florida, USA, impacts upon biodiversity, aesthetic and recreational values. To control the extent of willows and their rate of expansion into other extant wetlands, spatial context is critical to decision making. Modelling the spread of willows requires spatially explicit data on occupancy, an understanding of seed production, dispersal and how the key life-history stages respond to environmental factors and management actions. Nicholson et al. (2012) outlined the architecture of a management tool to integrate GIS spatial data, an external seed dispersal model and a state-transition dynamic Bayesian network (ST-DBN) for modelling the influence of environmental and management factors on temporal changes in willow stages. That paper concentrated on the knowledge engineering and expert elicitation process for the construction and scenario-based evaluation of the prototype ST-DBN. This paper extends that work by using object-oriented techniques to generalise the knowledge organisational structure of the willow ST-DBN and to construct an object-oriented spatial Bayesian network (OOSBN) for modelling the neighbourhood spatial interactions that underlie seed dispersal processes. We present an updated architecture for the management tool together with algorithms for implementing the dispersal OOSBN and for combining all components into an integrated tool.

1 INTRODUCTION

The highly-valued Upper St. Johns River in Florida, USA has been the focus of considerable restoration investment (Quintana-Ascencio et al., 2013). However, woody shrubs, primarily Carolina willow (*Salix caroliniana* Michx.), have invaded areas that were historically herbaceous marsh (Kinser et al., 1997). This change to the historical composition of mixed vegetation types is considered undesirable, as extensive willow thickets detract from biodiversity, aesthetic and recreational values. Overabundance of willows reduces local vegetation heterogeneity and habitat diversity. People also prefer open wetlands that offer a viewshed, navigable access and scope for recreation activities such as wildlife viewing, fishing and hunting.

Managers seek to control the overall extent of willows, their rate of expansion into other extant wetland types and encroachment into recently restored floodplain habitats. Spatial context is critical to decision-making as areas differ in terms of biodiversity, aesthetic and recreational value, "invasibility" and applicable interventions. For instance, vegetation communities that are intact or distant from willow populations (seed sources) are less susceptible to invasion. With respect to interventions, mechanical clearing is restricted to areas where the substrate can support heavy machinery; prescribed fire depends on water levels and the quantity of "burnable" understorey vegetation.

Modelling willow spread requires spatially explicit data on willow occupancy, an understanding of seed production, dispersal, germination and survival, and how the key life-history stages respond to environmental factors and management actions. Data and knowledge on these pieces of the puzzle are available from ecological and physiological theory, surveys, field and laboratory experiments and domain experts.

State-transition (ST) models are a convenient means of organising information and synthesising knowledge to represent system states and transitions that are of

management interest. We build on recent studies that combine ST models with BNs to incorporate uncertainty in hypothesised states and transitions, and enable sensitivity, diagnostic and scenario analysis for decision support in ecosystem management (e.g. Bashari et al., 2009; Rumpff et al., 2011). Our approach uses the template described by Nicholson and Flores (2011) to explicitly model temporal changes in willow stages.

Nicholson et al. (2012) outlined the architecture of a management tool that would integrate GIS spatial data, a seed dispersal model and a state-transition dynamic Bayesian network (ST-DBN) for modelling the influence of environmental and management factors on temporal changes in willow stages. That paper described the knowledge engineering and expert elicitation process for the construction and scenario-based evaluation of the prototype ST-DBN. This paper extends that work, using object-oriented techniques to generalise the knowledge organisational structure of the willow ST-DBN and to construct an object-oriented spatial Bayesian network (OOSBN) for modelling the neighbourhood spatial interactions that underlie seed dispersal processes. We present an updated architecture for the management tool that incorporates GIS data and the new ST-OODBN and OOSBN structures, together with algorithms for implementing the dispersal OOSBN and for combining all components into an integrated tool.

2 BACKGROUND

Dynamic Bayesian Networks (DBNs) are a variant of ordinary BNs (Dean and Kanazawa, 1989; Nicholson, 1992) that allow explicit modelling of changes over time. A typical DBN has nodes for N variables of interest, with copies of each node for each *time slice*. Links in a DBN can be divided into those between nodes in the same time slice, and those in the next time slice. While DBNs have been used in some enviromental applications (e.g. Shihab, 2008), their uptake has been limited.

State-and-transition models (STMs) have been used to model changes over time in ecological systems that have clear transitions between distinct states (e.g., in rangelands and woodlands, see Bestelmeyer et al., 2003; Rumpff et al., 2011). Nicholson and Flores (2011) proposed a template for state-transition dynamic Bayesian networks (ST-DBNs) which formalised and extended Bashari et al.'s model, combining BNs with the qualitative STMs.

The influence of environmental and management factors on the main willow stages of management interest and their transitions is shown in our updated version of the Nicholson et al. (2012) ST-DBN (see Figure 1). For each cell (spatial unit), data on attributes such as soil, vegetation type and information about landscape position and context is supplied from GIS data. This data provides inputs to parameterise the ST-DBN and dispersal model. A cell size of 100m x 100m (1 ha) was chosen to represent a modelling unit. This reflects the resolution of available spatial data for environmental attributes, makes the computational demand associated with seed dispersal modelling feasible, and is a reasonable scale with respect to candidate management actions. A time step of one year was considered appropriate given the willow's growth and seed production cycle (Nicholson et al., 2012).

Seed production depends on the size and number of reproductive (adult) stems *within* each cell. However, *Seed Availability*, the amount of seed available for germination within a cell, depends on willow seed production and dispersal from *surrounding* cells. As these processes are not accounted for in the ST-DBN (Figure 1), a key focus of this paper is the development and integration of an object-oriented spatial Bayesian network (OOSBN) to model the neighbourhood spatial interactions that underlie this process.

The purpose of the integrated tool is to synthesise current understanding and quantify important sources of uncertainty to support decisions on *where*, *when* and *how* to control willows most effectively. The ST-DBN models willow state transitions and characteristics in response to environmental and management factors within a single spatial unit and time step. For coherent, effective and well-coordinated landscape-scale management however, we want to be able to predict willow response across *space* (at every cell) in the target area and across *time* frames of management interest (e.g. 10-20 years). Such predictions can then be mapped and also aggregated across the target area to produce evaluation metrics for managers. Such a tool would enable managers to "test", visually compare and quantitatively evaluate different candidate management strategies.

This real-world management problem is naturally described in terms of hierarchies of components that include similar, repetitive structures. Object-oriented (OO) modelling has obvious advantages in this context. We apply OO techniques to generalise the knowledge organisational structure of the willow ST-DBN and design and construct the seed production and dispersal spatial network.

3 AN ST-OODBN FOR WILLOWS

Various authors have advocated the use of OO modelling techniques to: a) help manage BN complexity via abstraction and encapsulation, b) facilitate the

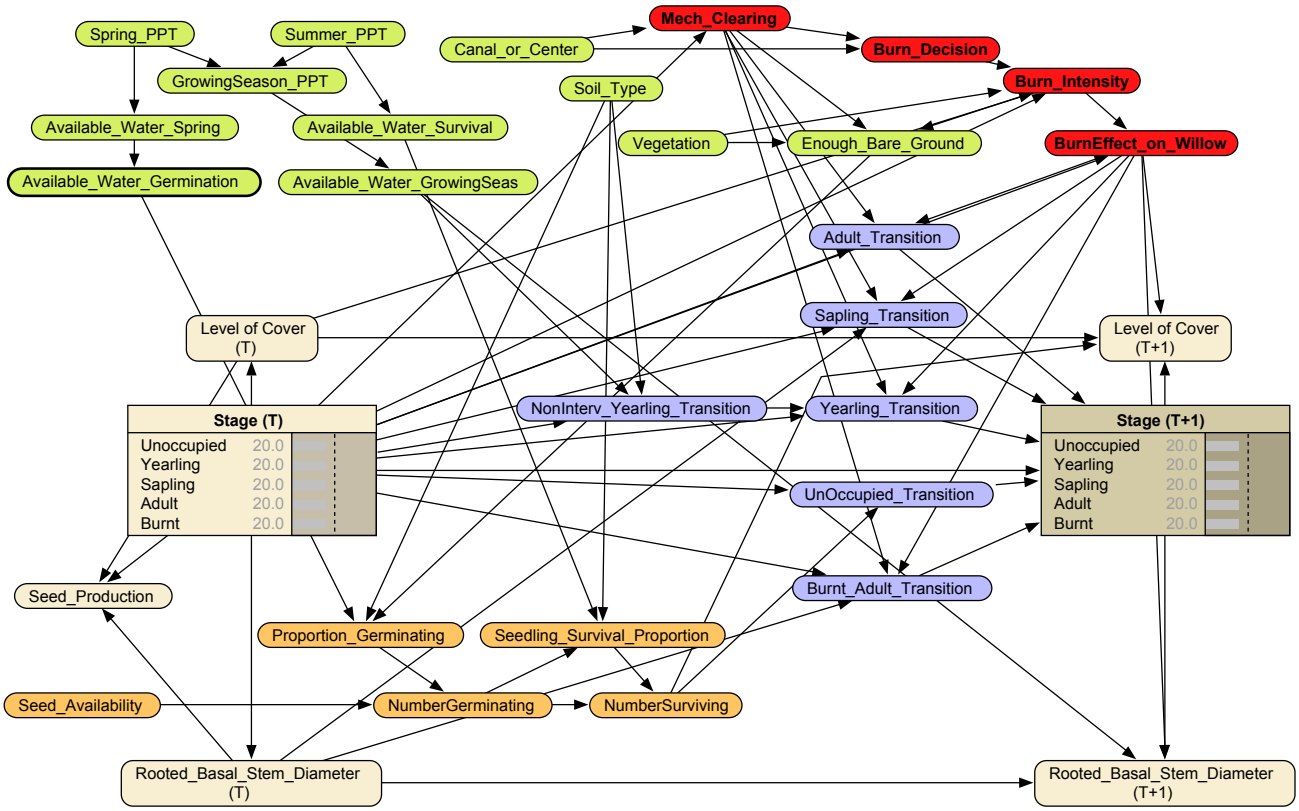


Figure 1: State-and-transition dynamic Bayesian network (ST-DBN) for modelling the response of key willow stages to environmental factors and management actions within a single spatial unit. The willow stages of management interest are: unoccupied, yearling, sapling, adult and burnt adult. Colours indicate: (i) aspects of willow state in tan; (ii) seed availability, germination and seedling survival processes in orange; (iii) environmental factors in green; (iv) management options in red; and (v) willow state-transitions in purple.

construction of classes of objects that are internally cohesive and potentially more reuseable, and c) formalise interfaces prior to integration (Koller and Pfeffer, 1997; Neil et al., 2000; Kjærulff and Madsen, 2008; Korb and Nicholson, 2010; Molina et al., 2010). However, examples in ecological and environmental management are scant (Molina et al., 2010; Carmona et al., 2011; Johnson and Mengersen, 2012).

We follow the definition of OOBNs used in Kjærulff and Madsen (2008), and implemented in the Hugin BN software package. A standard BN is made up of ordinary nodes, representing random variables. An OOBN class is made up of both nodes, and objects, which are instances of other classes. Thus an object may *encapsulate* multiple sub-networks, giving a composite and hierarchical structure. Objects are connected to other nodes via some of its own ordinary nodes, called its *interface* nodes. The rest of the nodes are not visible to the outside world, thus hiding information detail, another key OO concept. A class can be thought of as a self-contained 'template' for an OOBN object, described by its name, its interface and its hidden part. Finally, interface nodes are divided into input nodes and output nodes. Input nodes are the root nodes within an OOBN class, and when an object (instance) of that class becomes part of another class, each input node may be mapped to a single node (with the same state space) in the encapsulating class. The output nodes are the only nodes that may become parents of nodes in the encapsulating class. When displaying an OOBN, we show Hugin[1] screen shots, where input nodes are indicated with a dotted and shaded outline, and output nodes with a bold and shaded outline.

We converted the ST-DBN (Figure 1) into a ST-OODBN as follows. Using the five conceptual categories of nodes from the original network as a guide, the network was split into two abstract class types. The first represents abstract influencing factors and consists of three sub-classes that define *Environmental Conditions*, *Management Options* and the germination and seedling survival *Processes Factors*. The second type represents state transitions of willows

[1]Note that Hugin OOBNs do not support inheritance, so there are no super-classes or sub-classes.

(see Figure 2) and defines five sub-classes, *TransitionFromUnoccupied*, *TransitionFromYearling*, *TransitionFromSapling*, *TransitionFromAdult* and *TransitionFromBurntAdult*. Figure 3 illustrates the definition of the *TransitionFromYearling* class.

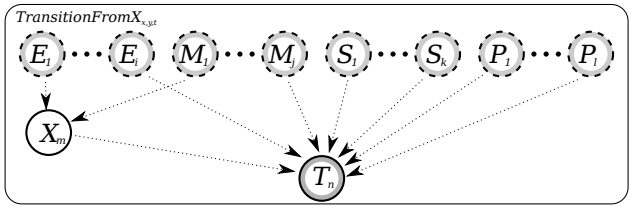


Figure 2: An abstract OOBN class for state transitions. Each implementation of a state transition output node T_n is defined by a combination of environmental conditions $E_{1...i}$, management options $M_{1...j}$, previous state variables $S_{1...k}$, process factors $P_{1...l}$ and any number of X_m hidden nodes. The only required input is the node that defines the previous state, all others are optional and are based on the implementing class. Dotted arrows indicate possible connections between input, hidden and output nodes.

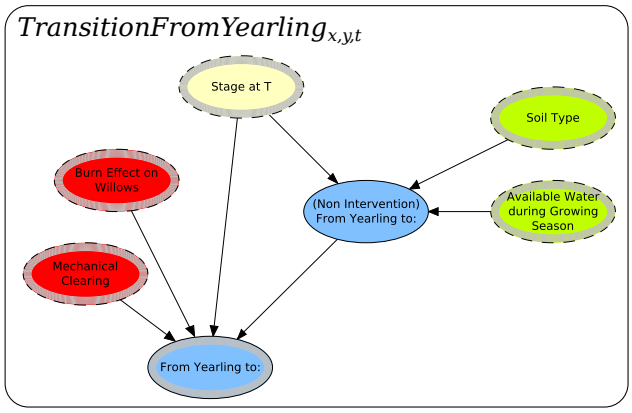


Figure 3: The *TransitionFromYearling* implementation of the abstract *TransitionFromX* type showing *Management Options* in red, *State Variables* in yellow and *Environmental Conditions* in green. The output *Transition* node *Transition from Yearling to:* is shown in blue with a bold and shaded outline.

These classes are instantiated as objects within a ST-OODBN class and when integrated with the seed dispersal OOSBN (described below) defines the complete ST model over a single time step (Figure 4). Recasting the network as a ST-OODBN makes the knowledge organisational structure explicit, whilst allowing network complexity to be hidden and integration efforts to focus on the interfaces between components of the network. Note that in the ST-OODBN (Figure 4) *Seed Availability* is an input node. *Seed Production* and its spatial dispersal is modelled by a separate OO network, which we present next.

4 AN OOSBN FOR SEED PRODUCTION AND DISPERSAL

S.caroliana flowers in early spring and produces very large numbers of small seeds (∼165,000 per average adult)that disperse by wind and water. Seed production is modelled by the *Willow Seed Production* OOBN (Figure 5), which is embedded in a broader seed dispersal model described below.

The number of seeds produced by an adult is given by the product of the number of *Inflorescences*, the number of *Fruits per inflorescence* and the number of *Seeds per fruit*. *Fruits per inflorescence* and *Seeds per fruit* are defined by distributions estimated from empirical data. The number of *Inflorescences* increases as a function of adult size (represented by *Rooted Basal Stem Diameter*) and this relationship has also been estimated from empirical data.

Cover is the percentage of a 1 hectare cell that is occupied by willows and *Average Canopy Area* is modelled as a function of *Rooted Basal Stem Diameter*. Together these two variables provide an estimate of the number of reproductive stems. Overall seed production within a cell, *Seeds per Hectare*, is then simply the product of the seed production per stem, by the number of reproductive stems. This *Willow Seed Production* OOBN models seed production processes explicitly rather than implicitly as in the ST-DBN prototype (Figure 1); an example of iterative and incremental knowledge engineering.

Willow seeds do not exhibit dormancy and have only a short period of viability – those that fail to germinate in the year they are produced are lost. The amount of seed available for germination within a cell depends on seed production and dispersal from surrounding cells. Thus, neighbourhood seed production and dispersal in combination with environmental and management factors determines patterns of willow spread and colonization.

Our approach to modelling seed dispersal is phenomenological rather than mechanistic. Wind-mediated seed dispersal is calculated using the Clark et al. (1999) dispersal kernel:

$$SD_{x,y}^{x',y'} = SP_{x',y'} \times \frac{1}{2\pi\alpha^2} e^{-(\frac{d}{\alpha})} \quad (1)$$

where $SD_{x,y}^{x',y'}$ is the number of seeds arriving at cell (x, y) from those produced at a cell (x', y'); it is the product of seed produced $SP_{x',y'}$ and an exponential

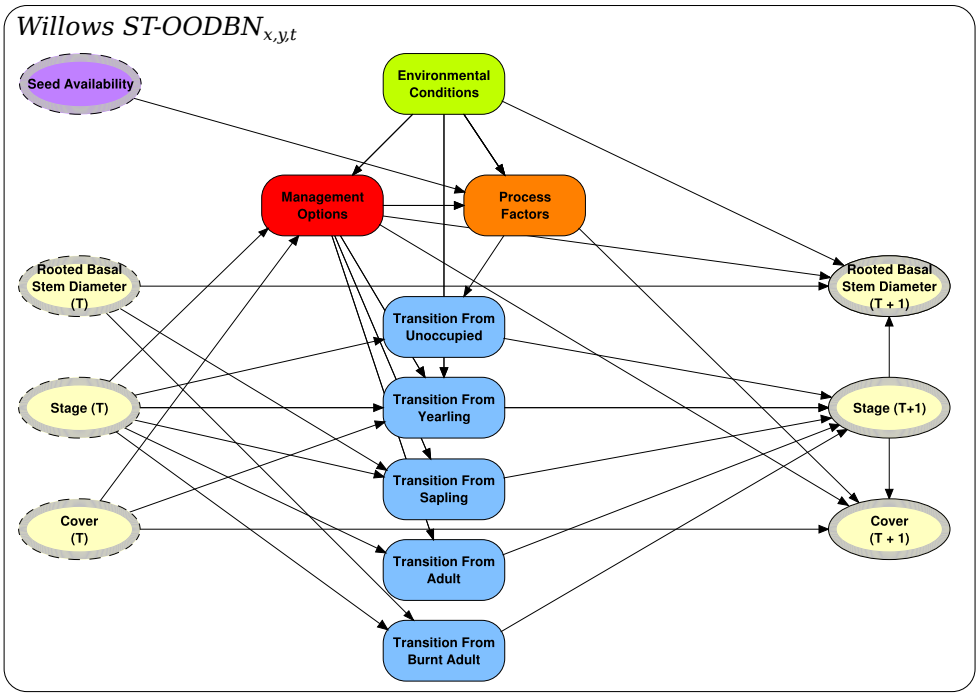


Figure 4: The resultant ST-OODBN class showing the four input nodes (*Seed Availability*, *Rooted Basal Stem Diameter (T)*, *Stage (T)* and *Cover (T)*) along with the *Environmental Conditions*, *Management Options*, *Process Factors* and five *TransitionFromX* objects that define the outputs (*Stage (T+1)*, *Rooted Basal Stem Diameter (T+1)* and *Cover (T+1)*) after one time step. Input nodes are illustrated with a dotted and shaded outline, instances of OOBN classes as round cornered boxes and output nodes with a bold and shaded outline.

kernel where d is the distance between cells (x, y) and (x', y'), and α is a distance parameter. To simulate stochasticity in dispersal events, α is a random variable that can be sampled from distributions designed to reflect the expected nature of dispersal (e.g. short versus long distance dispersal) (Fox et al., 2009). This seed dispersal model is captured within the *WindDispersalKernel*$_{x',y',t}$ OOBN (Figure 5), where the input *Distance* node is set for the particular (x, y) and α is set as a discretised normal distribution with a mean of 1 and a variance of 0.25.

For our purposes, we want to compute the seed availability $SA_{x,y}$ for a target cell (x, y), which is the sum of the seeds dispersed to it from every cell in the study area:

$$SA_{x,y} = \sum_{x',y' \in Area} SD_{x,y}^{x',y'} \qquad (2)$$

A naive BN model of this additive function would mean a *Seed Availability* node with all the *Seeds Dispersed* nodes (one for every cell) as its parents! For a study area with width w cells, height h cells, a *Seed Availability* node discretized to n states, and the *Seeds Dispersed* node discretized to m states, the CPT for *Seed Availability* would include $n \times m^{w \times h}$ probabilities –clearly infeasible.

From an ecological perspective, however, not *all* cells within a study area are expected to contribute towards final *Seed Availability* at any given cell. Indeed, because the number of seeds dispersed from a seed producing cell declines exponentially with increasing distance from that cell, we can make a simplifying assumption that after a certain distance, the number of seeds dispersed is effectively negligible. As a starting point we assume a circular region of influence for the target cell, defined by a dispersal mask with radius r. So for instance, a radius of eight cells (800 meters) implies $\pi 8^2$, or $\sim$201 cells providing parents to the final *Seed Availability* node. This is still far too many, particularly as we are using a standard BN software package with discrete nodes and exact inference.

However, since *Seed Availability* is a simple additive function, we use the simple modelling trick of adding the *Seed Availability* from each cell to the cumulative seed availability so far (via node *Cumulative Seed Availability*). This is equivalent to repeatedly divorcing parents to reduce the size of the state space. We can think of this is as sequentially scanning over the spatial dimension in a similar way to rolling out a DBN over time. We call this a spatial Bayesian network (SBN), and the object-oriented variety an OOSBN.

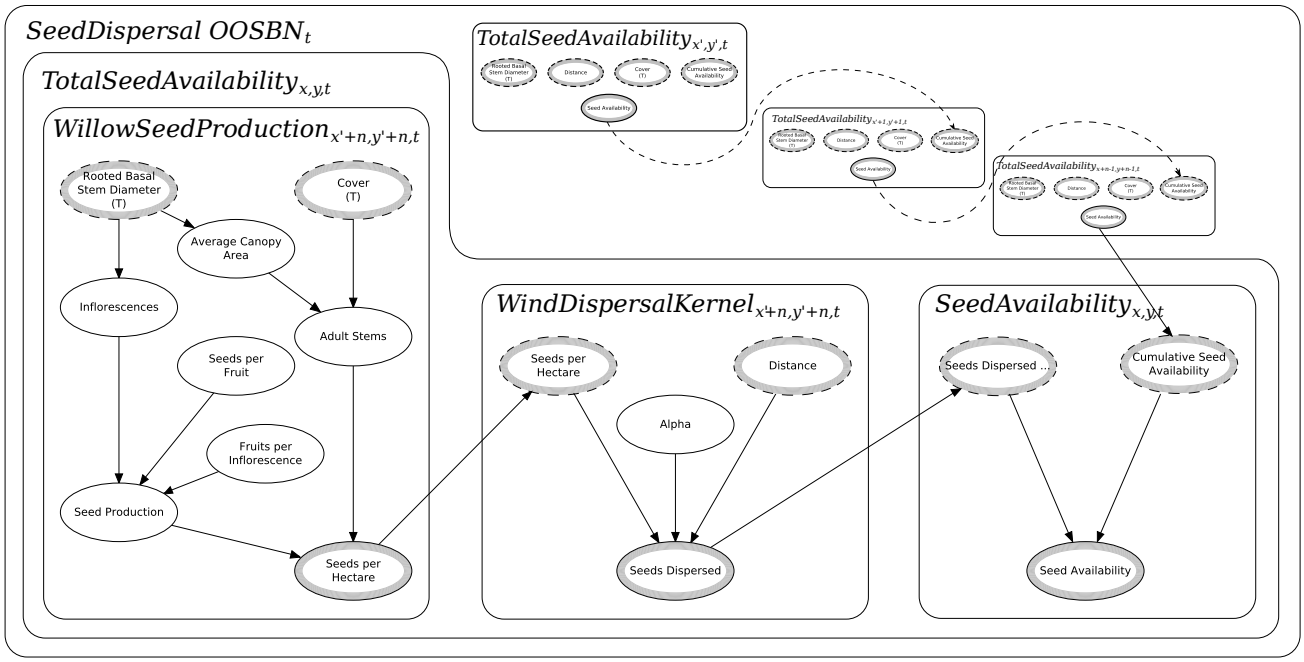


Figure 5: The OOSBN architecture showing how *Cover*, *Rooted Basal Stem Diameter*, *Cumulative Seed Availability* and *Distance* at locations (x', y') to $(x'+n, y'+n)$ are combined to provide total seed availability for each cell at location (x, y) at time t. Dashed arcs indicate that nodes are connected via the seed availability *PTLayer*. Multiple TotalSeedAvailability objects are illustrated within the SeedDispersal OOSBN reflecting that the total seeds available at a given (x, y) is dependant on seeds being produced in multiple locations. These synthetic TotalSeedAvailabilty objects are shown as collapsed objects, hiding their private nodes, and showing just the four input nodes that define the seeds dispersed from each producing cell to the target cell.

Our approach to integrating the seed dispersal makes use of such an OOSBN, that is run πr^2 times (i.e. once for every cell (x', y') within dispersal range) for each cell (x, y) in the study area to disperse and then sum the seed availability at each cell. The dispersal mask is flexible and can be designed to take on different shapes to reflect potentially important influences on wind dispersal such wind direction, wind strength and terrain characteristics. However, in the results given Section 6, we use a radius of eight cells.

Overall, our approach here is to mitigate the problem of large CPT sizes by turning the problem in to one of computation time. This has the added benefit of potentially being able to be computed easily in parallel and thus regaining some computation efficiency.

5 Integrating the ST-OODBN with the OOSBN

Figure 6 is an abstract representation of the system architecture, specifically, the interactions between the GIS layers and the OO networks as the tool is used for prediction at yearly intervals over the required management time frame.

For each cell in the GIS, there is conceptually one state-transition network (*ST-OODBN*) and one seed production and dispersal spatial network, (*OOSBN*). In practice, we do not store all these as separate networks, but rather re-use a single network structure, whose input nodes are re-parameterised for each cell, for each time step.

For the first time step, the system takes GIS data, which represents the initial conditions of the study area. These are stored in an internal data structure, *PTLayer*, that combines the spatial structure of the GIS, with distributions for the (discrete) nodes in the networks. *PTLayers* are used to store and pass the spatially referenced prior distributions of input nodes and posterior distributions of output nodes for both the *OOSBN* and *ST-OODBN* (Figure 6). Each *PTLayer* contains a number of fields, one for each of the node states of the linked input and output nodes. Each field stores the probability mass of the corresponding node state.

The *PTLayer* distributions are used to set the priors for the input nodes at each time step t. Then inference (belief updating) is performed within the OO network, producing new posterior probability distri-

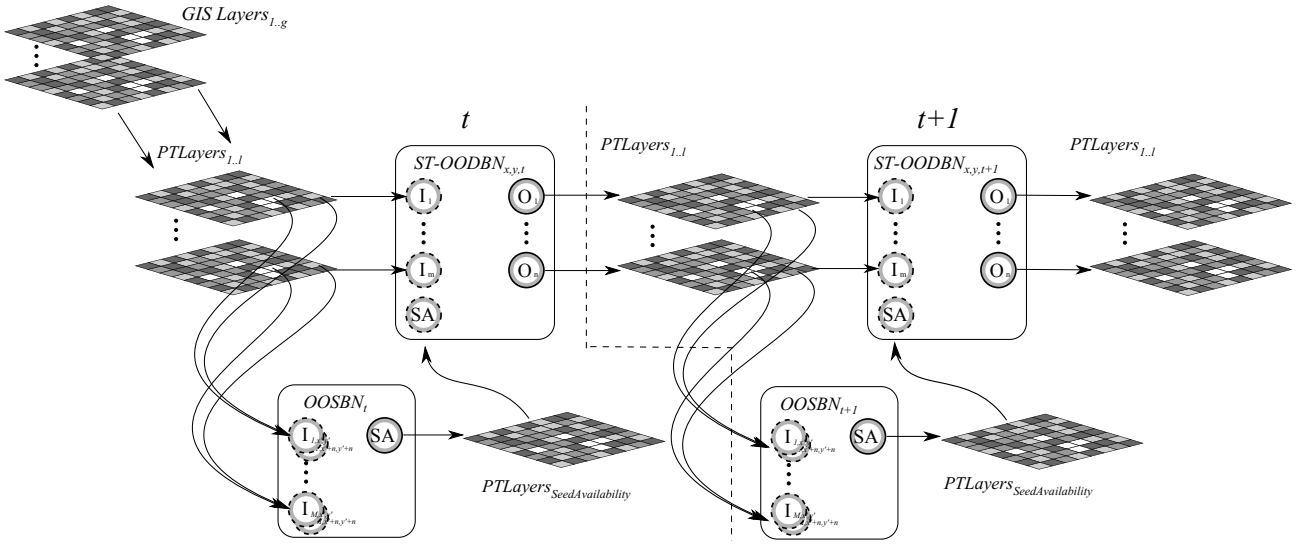


Figure 6: An abstract representation of the management tool architecture.

butions from the output nodes for each OO network, which are then transferred back to the corresponding *PTLayer*. This is equivalent to the standard "roll-out" followed by "roll-up" steps done in prediction with two time slice DBNs (Boyen and Koller, 1998) to avoid the computational complexity of rolling out a DBN over a large number of time steps. But here, in addition, the *PTLayers* are being used as an intermediate storage *across* the spatial grid, with the inputs for the next time $t+1$ coming not only from both the network for the same cell, but from outputs from networks for *other* cells. This is done via the seed dispersal OOSBN, which uses the *Seed Availability PTLayer* to accumulate the seed availability arising from seed production in other cells within the dispersal mask. In effect, the *PTLayer* replaces both the temporal arcs if the network was rolled-out over many time steps, and the spatial arcs between the networks for different cells, which are essentially the cross-network arcs from seed production in one place to seed availability in another. Note that this method is limited to prediction only –we cannot use the model for diagnosis, or to identify the starting states and management actions to achieve a preferred end-state.

More formally, using the notation from Figure 6, at time t and at each cell location (x, y), $PTLayers_{1...l}$ are used to initialise the priors of nodes $I_{1...m}$ and $I_{1...M}$ of $ST\text{-}OODBN_{x,y,t}$ and $OOSBN_t$ respectively. After propagation within $OOSBN_t$, beliefs from the output node SA are stored in $PTLayer_{SeedAvailabilty}$ and then used to update the priors of input node SA of $ST\text{-}OODBN_{x,y,t}$. Then belief updating is done for the $ST\text{-}OODBN_{x,y,t}$ for each cell(x, y) and the beliefs from output nodes $O_{1...n}$ propagated back to $PTLayers_{1..l}$ at time $t+1$. Although the mapping between the set of $PTLayers_{1...l}$ and input nodes $I_{1...m}$ is one-to-one, there may be cases where there are no GIS layers available for an input, in which case a prior distribution is used. Finally, with respect to OOSBN propagation, the range of locations $x', y' \ldots x'+n, y'+n$ (i.e. neighbourhood cells) that are included is defined by the dispersal mask described in the previous section.

Algorithm 1 An algorithm for propagating a GIS coupled ST-OODBN with spatial OOSBN subnetworks

```
1:  function PROPAGATE(ST− OODBN, OOSBN, ptlayers, t)
2:      I ← I(ST-OODBN)
3:      O ← O(ST-OODBN)
4:      SA ← getLayer(ptlayers, SeedAvailability)
5:      for t := 0 to t do
6:          PROPAGATE(OOSBN,ptlayers,dispMask)
7:          // computes all Seed Availabilities
8:          for all (x, y) ∈ Area do
9:              p(SeedAvailability) ← SA_{x,y}
10:             for all I_i ∈ I do
11:                 L_j ← getLayer(ptlayers, I_i)
12:                 p(I_i) ← L_j(x, y)
13:             end for
14:             update beliefs in ST-OODBN
15:             for all O_i ∈ O do
16:                 L_j ← getLayer(ptlayers, O_i)
17:                 L_j(x, y) ← Bel(O_i)
18:             end for
19:         end for
20:     end for
21: end function
```

The process is detailed in Algorithm 1, as a function $PROPAGATE$ which takes the ST-OODBN (shown in Fig. 4), and OOSBN (shown in Fig. 5) networks, a list of *ptlayers* (previously initialised from the GIS layers) whose cells correspond to the area under consideration, and the number of time steps T over which to propagate the network. In the algorithm, we use

$I(OOBN)$ (respectively $O(OOBN)$) to denote a function that returns the input (resp. output) nodes of the interface of the OOBN, and *getLayer(ptlayers,V)* to denote a function that returns the *PTLayer* corresponding to a node V.

Algorithm 2 An algorithm for dispersing seeds by wind using an OOSBN class

```
1: function PROPAGATE(OOSBN, ptlayers, dispMask)
2:     SA ← getLayer(ptlayers, SeedAvailability)
3:     for all (x, y) ∈ Area do
4:         P(SA_{x,y} = none) ← 1
5:     end for
6:     for all (x, y) ∈ Area do
7:         for all I_i ∈ I_{x,y}(OOSBN) do
8:             L_j ← getLayer(ptlayers, I_i)
9:             p(I_i) ← L_j(x, y)
10:        end for
11:        for all (x', y') ∈ dispMask do
12:            p(CumSeedAvail) ← SA_{x,y}
13:            d ← sqrt((x − x')^2 + (y − y')^2)
14:            p(Distance = d) ← 1
15:            update beliefs in OOSBN
16:            SA_{x,y} ← Bel(SeedAvailability)
17:        end for
18:    end for
19: end function
```

First, seed dispersal is done with the OOSBN using Algorithm 2. Then for each cell (x, y) in the area, for each input node I_i, the distribution for that cell from its corresponding *ptlayer* is set as the prior of the input node. Belief updating of the ST-OODBN is done, propagating the new priors through to updated posterior distributions for the output nodes. Finally the beliefs for each output node ($Bel(O_i)$) are copied back to the distribution at the (x, y) cell for the corresponding *ptlayer* (i.e. into $SA_{x,y}$).

Algorithm 2 details the propagation process using the *SeedDispersal OOSBN* class illustrated in Figure 5. The algorithm starts by taking the OOSBN, a list of *PTLayers* whose cells correspond to the area under consideration, and a dispersal mask (*dispMask*). Before starting the dispersal process, the provided *Seed Availability PTLayer* is initialised with no seeds available at all (x, y) co-ordinates. It then loops through each cell (x, y) in the area, setting the distribution for each input node I_i (i.e., *Cover*, *Rooted Basal Stem Diameter* and *Cumulative Seed Availability*) from its corresponding *PTlayer* cell. The algorithm then enters a second loop for every cell that is a possible seed source based on the dispersal mask. The *Distance* node is set using the Euclidean distance between the current cell and the target co-ordinates. Finally, belief updating is done within the OOSBN (Figure 5) and the beliefs (i.e. the posterior probability distribution) from the *Seed Availability* node at each point are transferred into the *Cumulative Seed Availability* for the subsequent cell at each iteration. After all the cells (x', y') within the dispersal mask have been visited, the *Seed Availability PTLayer*, $SA_{x,y}$ contains the overall seed availability in that cell, which is used for modelling germination in the ST-OODBN in Algorithm 1.

6 PRELIMINARY RESULTS

To implement the software architecture described we chose to use Hugin Researcher 7.7(2013) to develop the ST-OODBN and dispersal model OOSBN, the Hugin Researcher Java API 7.7 (2013) to provide programmatic access to the developed networks, the ImageIO-ext (2013) java library to provide access to GIS raster layer formats, and the Java programming language to implement the algorithms tying the components together. Hugin was chosen as the OOBN development platform as it currently has one of the most complete OOBN implementations. Java was chosen as the implementing language as it is platform independent and provides for a well established and understood OO development environment. We implemented the tool as a standalone program allowing preprocessing of GIS data to be performed in whatever program the end user was most familiar with. In our case we used a combination of ArcGIS (2013), Quantum GIS (2013)and SAGA GIS(2013).

To demonstrate our working implementation, we ran the model for the Blue Cypress Marsh Conservation Area (138 x 205 cells) within the Upper St. Johns River basin. We used a simplified (and unrealistic) management rule set that says if a cell is next to a canal, mechanical clearing is carried out, otherwise for landlocked cells, burning is prescribed (with a probability of 0.1). Maps of willow cover and seed production were generated at yearly intervals for a 25 year prediction window. This took about 8 hours of computation on a 64bit machine with a 2.8GHz processor. Figure 7 shows seed availability across the study area using output from the *SeedAvailability* nodes in the OOSBN at 5 yearly intervals. To produce the maps, the seed availability interval with the highest posterior probability distribution is used to produce a grayscale value where *zero* seeds is black, and 10^{12} is white. In the run shown, seed availability decreases over time as the level of willow cover is reduced by the management regime.

7 DISCUSSION AND FUTURE WORK

For coherent, coordinated and effective landscape-scale decision support, managers need the capability to predict willow state changes across *space* and *time*. We have tackled the challenges of this real-world problem by synthesising ideas and techniques from object-oriented knowledge engineering, dynamic BNs, GIS-

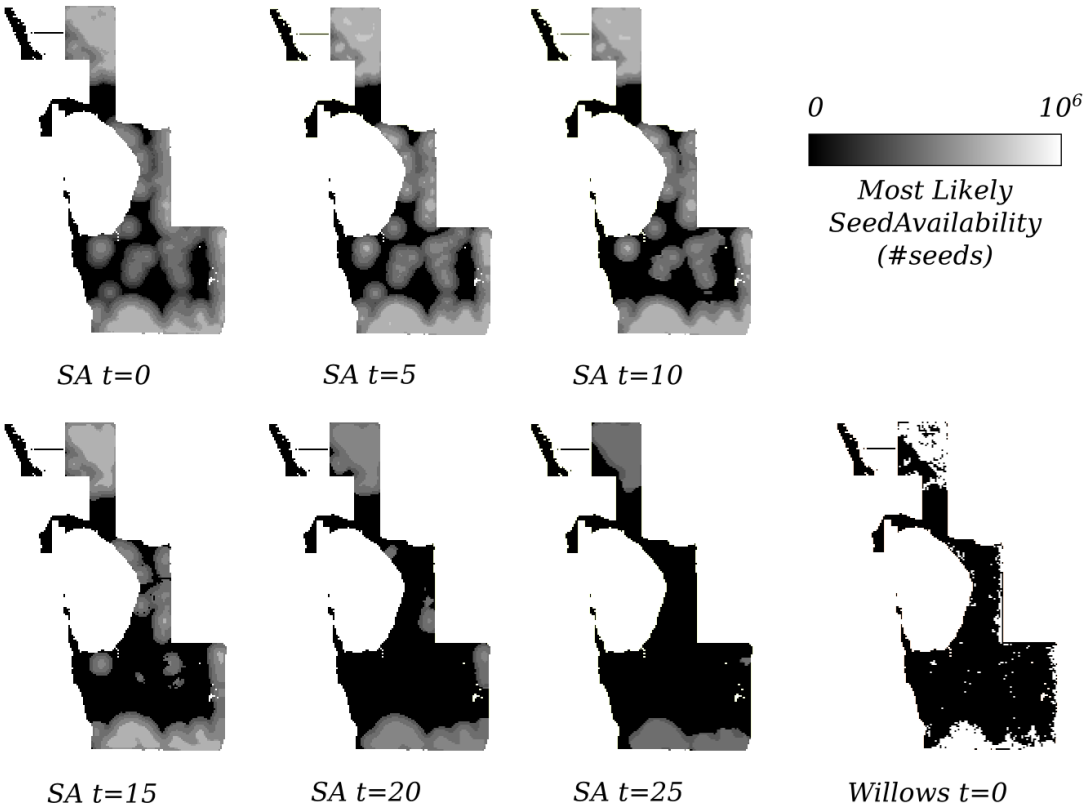


Figure 7: Seed availability predicted by the Willows ST-OODBN at $t = 0, 5, 10, 15, 20, 25$ years, across the Blue Cypress Marsh Conservation Area (138 x 208 cells). Adult willow occupancy at $t = 0$ is shown in the bottom right panel; black indicates absence, grey presence.

coupled BNs and dispersal modelling. To our knowledge, this is the first environmental management application in which OOBNs are used to model spatially-explicit process interactions.

Further work in the development of this management tool includes developing a water dispersal model and updating the parametrisation of the *ST-OODBN* and *OOSBN* using judgements from a larger pool of domain experts, together with specific empirical data where available. In addition, we will work with managers and domain experts to identify: i) realistic management scenarios, ii) useful summary descriptors for the various model outputs and iii) desirable features for a tool interface.

Throughout the research and integration process we encountered challenges with the development, management and use of OOBNs. While there have been advances in OOBN software, they still lack a lot of the useful features available in other development tools. For instance, modern software engineering IDEs provide easy to use re-factoring, documentation and integration with version control tools. The tools we used to design and implement the underlying OOBNs for our tool still lack powerful refactoring, making the management of object interface changes a time consuming and error prone task. Integrated source control is non-existent and documentation tools rudimentary. Improvements in these areas would make working with OOBNs far more accessible to the type of user that wishes to make use of OOBNs for natural resource management.

With respect to spatialising the ST-OOBN with the use of OOSBNs, there is currently no graphical tool up to the task of facilitating the integration of the required components. This means that anyone wanting to replicate our work would need to make use of the available APIs and this constitutes a barrier to usage by people with no or little programming background.

Acknowledgements

This work was supported by ARC Linkage Project LP110100304 and the Australian Centre of Excellence for Risk Analysis. John Fauth (UCF), and Dianne Hall, Kimberli Ponzio and Ken Snyder (SJWRMD) provided critical information about the St Johns River ecosystem, and knowledge about willow invasion.

References

Bashari, H., Smith, C., and Bosch, O. (2009). Developing decision support tools for rangeland management by combining state and transition models and Bayesian belief networks. *Agricultural Systems*, 99(1):23–34.

Bestelmeyer, B. T., Brown, J. R., Havstad, K. M., Alexander, R., Chavez, G., and Herrick, J. E. (2003). Development and use of state-and-transition models for rangelands. *Journal of Range Management*, (2):114–126.

Boyen, X. and Koller, D. (1998). Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 33–42, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Carmona, G., Molina, J., and Bromley, J. (2011). Object-Oriented Bayesian networks for participatory water management: two case studies in Spain. *Journal of Water Resources Planning and Management*, 137(4):366–376.

Clark, J. D., Silman, M., Kern, R., Macklin, E., and Hille Ris Lambers, J. (1999). Seed dispersal near and far: patterns across temperate and tropical forests. *Ecology*, 80(5):1475–1494.

Dean, T. and Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150.

Esri (2013). ArcGIS (Version 10.1) [Software]. `http://www.esri.com/software/arcgis`.

Fox, J. C., Buckley, Y. M., Panetta, F. D., Bourgoin, J., and Pullar, D. (2009). Surveillance protocols for management of invasive plants: modelling Chilean needle grass (*Nassella neesiana*) in Australia. *Diversity and Distributions*, 15(4):577–589.

GeoSolutions (2013). Image-IO-ext Java Library (Version 1.1.7) [Software]. `http://github.com/geosolutions-it/`.

Hugin Expert A/S (2013). Hugin Researcher (Version 7.7) [Software]. `http://www.hugin.com`.

Johnson, S. and Mengersen, K. (2012). Integrated Bayesian network framework for modeling complex ecological issues. *Integrated Environmental Assessment and Management*, 8(3):480–90.

Kinser, P., Lee, M. A., Dambek, G., Williams, M., Ponzio, K., and Adamus, C. (1997). Expansion of Willow in the Blue Cypress Marsh Conservation Area, Upper St. Johns River Basin. Technical report, St. Johns River Water Management District, Palatka, Florida.

Kjærulff, U. B. and Madsen, A. (2008). *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*. Springer Verlag, New York.

Koller, D. and Pfeffer, A. (1997). Object-oriented Bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI'97, pages 302–313, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Korb, K. B. and Nicholson, A. (2010). *Bayesian artificial intelligence*. Chapman&Hall/CRC, Boca Raton, FL, 2nd edition.

Molina, J., Bromley, J., García-Aróstegui, J., Sullivan, C., and Benavente, J. (2010). Integrated water resources management of overexploited hydrogeological systems using Object-Oriented Bayesian Networks. *Environmental Modelling & Software*, 25(4):383–397.

Neil, M., Fenton, N., and Nielson, L. (2000). Building large-scale Bayesian networks. *The Knowledge Engineering Review*, 15(3):1–33.

Nicholson, A., Chee, Y., and Quintana-Ascencio, P. (2012). A state-transition DBN for management of willows in an american heritage river catchment. In Nicholson, A., Agosta, J. M., and Flores, J., editors, *Ninth Bayesian Modeling Applications Workshop at the Conference of Uncertainty in Artificial Intelligence*, Catalina Island, CA, USA, http://ceur-ws.org/Vol-962/paper07.pdf.

Nicholson, A. E. (1992). *Monitoring Discrete Environments using Dynamic Belief Networks*. PhD thesis, Department of Engineering Sciences, Oxford.

Nicholson, A. E. and Flores, M. J. (2011). Combining state and transition models with dynamic Bayesian networks. *Ecological Modelling*, 222(3):555–566.

Quantum GIS Development Team (2013). Quantum GIS (Version 1.8.0) [Software]. `http://www.qgis.org/`.

Quintana-Ascencio, P. F., Fauth, J. E., Castro Morales, L. M., Ponzio, K. J., Hall, D., and Snyder, K. (2013). Taming the beast: managing hydrology to control Carolina Willow (*Salix caroliniana*) seedlings and cuttings. *Restoration Ecology*.

Rumpff, L., Duncan, D., Vesk, P., Keith, D., and Wintle, B. (2011). State-and-transition modelling for adaptive management of native woodlands. *Biological Conservation*, 144(4):1224–1236.

SAGA Development Team (2013). SAGA GIS (Version 2.0.8) [Software]. `http://www.saga-gis.org/`.

Shihab, K. (2008). Dynamic modeling of groundwater pollutants with Bayesian networks. *Applied Artificial Intelligence*, 22(4):352–376.

Product Trees for Gaussian Process Covariance in Sublinear Time

David A. Moore
Computer Science Division
University of California, Berkeley
Berkeley, CA 94709
dmoore@cs.berkeley.edu

Stuart Russell
Computer Science Division
University of California, Berkeley
Berkeley, CA 94709
russell@cs.berkeley.edu

Abstract

Gaussian process (GP) regression is a powerful technique for nonparametric regression; unfortunately, calculating the predictive variance in a standard GP model requires time $O(n^2)$ in the size of the training set. This is cost prohibitive when GP likelihood calculations must be done in the inner loop of the inference procedure for a larger model (e.g., MCMC). Previous work by Shen et al. (2006) used a k-d tree structure to approximate the predictive mean in certain GP models. We extend this approach to achieve efficient approximation of the predictive covariance using a tree clustering on pairs of training points. We show empirically that this significantly increases performance at minimal cost in accuracy. Additionally, we apply our method to "primal/dual" models having both parametric and nonparametric components and show that this enables efficient computations even while modeling longer-scale variation.

1 Introduction

Complex Bayesian models often tie together many smaller components, each of which must provide its output in terms of probabilities rather than discrete predictions. As a natively probabilistic technique, Gaussian process (GP) regression (Rasmussen and Williams, 2006) is a natural fit for such systems, but its applications in large-scale Bayesian models have been limited by computational concerns: training a GP model on n points requires $O(n^3)$ time, while computing the predictive distribution at a test point requires $O(n)$ and $O(n^2)$ operations for the mean and variance respectively.

This work focuses specifically on the fast evaluation of GP likelihoods, motivated by the desire for efficient inference in models that include a GP regression component. In particular, we focus on the predictive covariance, since this computation time generally dominates that of the predictive mean. In our setting, training time is a secondary concern: the model can always be trained offline, but the likelihood evaluation occurs in the inner loop of an ongoing inference procedure, and must be efficient if inference is to be feasible.

One approach to speeding up GP regression, common especially to spatial applications, is the use of covariance kernels with short lengthscales to induce sparsity or near-sparsity in the kernel matrix. This can be exploited directly using sparse linear algebra packages (Vanhatalo and Vehtari, 2008) or by more structured techniques such as space-partitioning trees (Shen et al., 2006; Gray, 2004); the latter approaches create a query-dependent clustering to avoid considering regions of the data not relevant to a particular query. However, previous work has focused on efficient calculation of the predictive mean, rather than the variance, and the restriction to short lengthscales also inhibits application to data that contain longer-scale variations.

In this paper, we develop a tree-based method to efficiently compute the predictive covariance in GP models. Our work extends the weighted sum algorithm of Shen et al. (2006), which computes the predictive mean. Instead of clustering points with similar weights, we cluster *pairs* of points having similar weights, where the weights are given by a kernel-dependent distance metric defined on the *product space* consisting of all pairs of training points. We show how to efficiently build and compute using a *product tree* constructed in this space, yielding an adaptive covariance computation that exploits the geometric structure of the training data to avoid the need to explicitly consider each pair of training points. This enables us to present what is to our knowledge the first account of GP regression in which the major test-time operations

(predictive mean, covariance, and likelihood) run in time sublinear in the training set size, given a suitably sparse kernel matrix. As an extension, we show how our approach can be applied to GP models that combine both parametric and nonparametric components, and argue that such models present a promising option for modeling global-scale structure while maintaining the efficiency of short-lengthscale GPs. Finally, we present empirical results that demonstrate significant speedups on synthetic data as well as a real-world seismic dataset.

2 Background

2.1 GP Regression Model

We assume as training input a set of labeled points $\{(\mathbf{x}_i, y_i)|i = 1, \ldots, n\}$, where we suppose that

$$y_i = f(\mathbf{x}_i) + \epsilon_i$$

for some unknown function $f(\cdot)$ and i.i.d. Gaussian observation noise $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$. Treating the estimation of $f(\cdot)$ as a Bayesian inference problem, we consider a Gaussian process prior distribution $f(\cdot) \sim GP(0, k)$, parameterized by a positive-definite *covariance* or *kernel* function $k(x, x')$. Given a set X^* containing m test points, we derive a Gaussian posterior distribution $f(X^*) \sim \mathcal{N}(\mu_*, \Sigma_*)$, where

$$\mu_* = K^{*T} K_y^{-1} \mathbf{y} \quad (1)$$

$$\Sigma_* = K^{**} - K^{*T} K_y^{-1} K^* \quad (2)$$

and $K_y = K(X, X) + \sigma_n^2 I$ is the covariance matrix of training set observations, $K^* = k(X, X^*)$ denotes the $n \times m$ matrix containing the kernel evaluated at each pair of training and test points, and similarly $K^{**} = k(X^*, X^*)$ gives the kernel evaluations at each pair of test points. Details of the derivations, along with general background on GP regression, can be found in Rasmussen and Williams (2006).

In this work, we make the additional assumption that the input points $\mathbf{x}_i$ and test points $\mathbf{x}_p^*$ lie in some metric space $(\mathcal{M}, d)$, and that the kernel is a monotonically decreasing function of the distance metric. Many common kernels fit into this framework, including the exponential, squared-exponential, rational quadratic, and Matérn kernel families; anisotropic kernels can be represented through choice of an appropriate metric.

2.2 k-d and Metric Trees

Tree structures such as k-d trees (Friedman et al., 1977) form a hierarchical, multiresolution partitioning of a dataset, and are commonly used in machine learning for efficient nearest-neighbor queries. They have also been adapted to speed up nonparametric regression (Moore et al., 1997; Shen et al., 2006); the general approach is to view the regression computation of interest as a sum over some quantity associated with each training point, weighted by the kernel evaluation against a test point. If there are sets of training points having similar weight – for example, if the kernel is very wide, if the points are very close to each other, or if the points are all far enough from the query to have effectively zero weight – then the weighted sum over the set of points can be approximated by an unweighted sum (which does not depend on the query and may be precomputed) times an estimate of the typical weight for the group, saving the effort of examining each point individually. This is implemented as a recursion over a tree structure augmented at each node with the unweighted sum over all descendants, so that recursion can be cut off with an approximation whenever the weight function is shown to be suitably uniform over the current region.

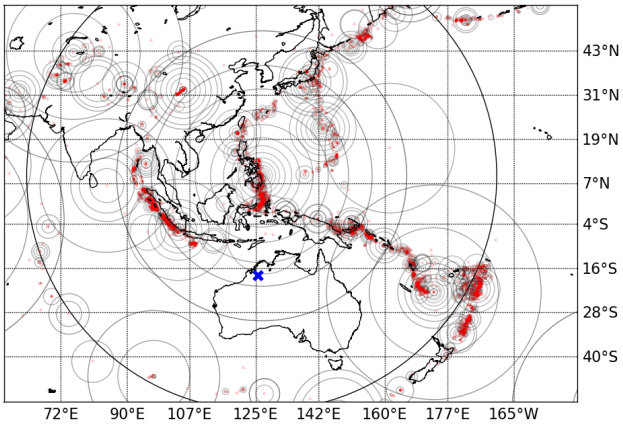


Figure 1: Cover tree decomposition of seismic event locations recorded at Fitzroy Crossing, Australia (with X marking the station location).

Major drawbacks of k-d trees include poor performance in high dimensions and a limitation to Euclidean spaces. By contrast, we are interested in non-Euclidean metrics both as a matter of practical application (e.g., in a geophysical setting we might consider points on the surface of the earth) and because some choices of kernel function require our algorithm to operate under a non-Euclidean metric even if the underlying space is Euclidean (see section 3.2). We therefore consider instead the class of trees having the following properties: (a) each node $\mathbf{n}$ is associated with some point $x_\mathbf{n} \in \mathcal{M}$, such that all descendants of $\mathbf{n}$ are contained within a ball of radius $r_\mathbf{n}$ centered at $x_\mathbf{n}$, and (b) for each leaf $\mathbf{L}$ we have $x_\mathbf{L} \in X$, with exactly one leaf node for each training point $\mathbf{x}_i \in X$. We call any tree satisfying these properties a *metric tree*.

function WEIGHTEDMETRICSUM(node $\mathbf{n}$, query points $(\mathbf{x}_i^*, \mathbf{x}_j^*)$, accumulated sum $\hat{S}$, tolerances ϵ_{rel}, ϵ_{abs})
  $\delta_{\mathbf{n}} \leftarrow \delta((\mathbf{x}_i^*, \mathbf{x}_j^*), (\mathbf{n}_1, \mathbf{n}_2))$
  if $\mathbf{n}$ is a leaf **then**
    $\hat{S} \leftarrow \hat{S} + (K_y^{-1})_{\mathbf{n}} \cdot \left(k(d(\mathbf{x}_i^*, \mathbf{n}_1)) \cdot k(d(\mathbf{x}_j^*, \mathbf{n}_2))\right)$
  else
    $w_{\min} \leftarrow k_{\text{lower}}^{\text{prod}}(\delta_{\mathbf{n}} + r_{\mathbf{n}})$
    $w_{\max} \leftarrow k_{\text{upper}}^{\text{prod}}(\max(\delta_{\mathbf{n}} - r_{\mathbf{n}}, 0))$
    if $w_{\max} \cdot S_{\mathbf{n}}^{\text{Abs}} \leq \left(\epsilon_{\text{rel}} \left|\hat{S} + w_{\min} \cdot S_{\mathbf{n}}^{\text{UW}}\right| + \epsilon_{\text{abs}}\right)$ **then**
      $\hat{S} \leftarrow \hat{S} + \frac{1}{2}(w_{\max} + w_{\min}) \cdot S_{\mathbf{n}}^{\text{UW}}$
    else
      for each child $\mathbf{c}$ of $\mathbf{n}$
       sorted by ascending $\delta((\mathbf{x}_i^*, \mathbf{x}_j^*), (\mathbf{c}_1, \mathbf{c}_2))$ **do**
        $\hat{S} \leftarrow \hat{S} +$ WEIGHTEDMETRICSUM$(\mathbf{c}, (\mathbf{x}_i^*, \mathbf{x}_j^*), \hat{S}, \epsilon_{\text{rel}}, \epsilon_{\text{abs}})$
      end for
    end if
  end if
  return $\hat{S}$
end function

Figure 2: Recursive algorithm to computing GP covariance entries using a product tree. Abusing notation, we use $\mathbf{n}$ to represent both a tree node and the pair of points $\mathbf{n} = (\mathbf{n}_1, \mathbf{n}_2)$ associated with that node.

Examples of metric trees include many structures designed specifically for nearest-neighbor queries, such as ball trees (Uhlmann, 1991) and cover trees (Beygelzimer et al., 2006), but in principle any hierarchical clustering of the dataset, e.g., an agglomerative clustering, might be augmented with radius information to create a metric tree. Although our algorithms can operate on any metric tree structure, we use cover trees in our implementation and experiments. A cover tree on n points can be constructed in $O(n \log n)$ time, and the construction and query times scale only with the *intrinsic* dimensionality of the data, allowing for efficient nearest-neighbor queries in higher-dimensional spaces (Beygelzimer et al., 2006). Figure 1 shows a cover-tree decomposition of one of our test datasets.

3 Efficient Covariance using Product Trees

We consider efficient calculation of the GP covariance (2). The primary challenge is the multiplication $K^{*T} K_y^{-1} K^*$. For simplicity of exposition, we will focus on computing the (i, j)th entry of the resulting matrix, i.e., on the multiplication $\mathbf{k_i^*}^T K_y^{-1} \mathbf{k_j^*}$ where $\mathbf{k_i^*}$ denotes the vector of kernel evaluations between the training set and the ith test point, or equivalently the ith column of K^*. Note that a naïve implementation of this multiplication requires $O(n^2)$ time.

We might be tempted to apply the vector multiplication primitive of Shen et al. (2006) separately for each row of K_y^{-1} to compute $K_y^{-1} \mathbf{k_j^*}$, and then once more to multiply the resulting vector by $\mathbf{k_i^*}$. Unfortunately, this requires n vector multiplications and thus scales (at least) linearly in the size of the training set. Instead, we note that we can rewrite $\mathbf{k_i^*}^T K_y^{-1} \mathbf{k_j^*}$ as a weighted sum of the entries of K_y^{-1}, where the weight of the (p, q)th entry is given by $k(\mathbf{x_i^*}, \mathbf{x_p}) k(\mathbf{x_j^*}, \mathbf{x_q})$:

$$\mathbf{k_i^*}^T K_y^{-1} \mathbf{k_j^*} = \sum_{p=1}^{N} \sum_{q=1}^{N} (K_y^{-1})_{pq} k(\mathbf{x_i^*}, \mathbf{x_p}) k(\mathbf{x_j^*}, \mathbf{x_q}). \quad (3)$$

Our goal is to compute this weighted sum efficiently using a tree structure, similar to Shen et al. (2006), except that instead of clustering points with similar weights, we now want to cluster *pairs* of points having similar weights.

To do this, we consider the *product space* $\mathcal{M} \times \mathcal{M}$ consisting of all pairs of points from $\mathcal{M}$, and define a *product metric* δ on this space. The details of the product metric will depend on the choice of kernel function, as discussed in section 3.2 below. For the moment, we will assume a SE kernel, of the form $k_{SE}(d) = \exp(-d^2)$, for which a natural choice is the 2-product metric:

$$\delta((\mathbf{x}_a, \mathbf{x}_b), (\mathbf{x}_c, \mathbf{x}_d)) = \sqrt{d(\mathbf{x}_a, \mathbf{x}_c)^2 + d(\mathbf{x}_b, \mathbf{x}_d)^2}.$$

Note that this metric, taken together with the SE kernel, has the fortunate property

$$k_{SE}(d(\mathbf{x}_a, \mathbf{x}_b)) k_{SE}(d(\mathbf{x}_c, \mathbf{x}_d)) = k_{SE}(\delta((\mathbf{x}_a, \mathbf{x}_b), (\mathbf{x}_c, \mathbf{x}_d))),$$

i.e., the property that evaluating the kernel in the product space (rhs) gives us the correct weight for our weighted sum (3) (lhs).

Now we can run any metric tree construction algorithm (e.g., a cover tree) using the product metric to build a *product tree* on all *pairs* of training points. In principle, this tree contains n^2 leaves, one for each pair of training points. In practice it can often be made much smaller; see section 3.1 for details. At each leaf node $\mathbf{L}$, representing a pair of training points, we store the element $(K_y^{-1})_{\mathbf{L}}$ corresponding to those two training points, and at each higher-level node $\mathbf{n}$ we cache the unweighted sum $S_{\mathbf{n}}^{UW}$ of these entries over all of its descendant leaf nodes, as well as the sum of absolute values $S_{\mathbf{n}}^{Abs}$ (these cached sums will be used to determine when to cut off recursive calculations):

$$S_{\mathbf{n}}^{\text{UW}} = \sum_{\mathbf{L} \in \text{leaves}(\mathbf{n})} (K_y^{-1})_{\mathbf{L}} \quad (4)$$

$$S_{\mathbf{n}}^{\text{Abs}} = \sum_{\mathbf{L} \in \text{leaves}(\mathbf{n})} \left|(K_y^{-1})_{\mathbf{L}}\right|. \quad (5)$$

Given a product tree augmented in this way, the weighted-sum calculation (3) is performed by the WEIGHTEDMETRICSUM algorithm of Figure 2. This algorithm is similar to the WEIGHTEDSUM and WEIGHTEDXTXBELOW algorithms of Shen et al. (2006) and Moore et al. (1997) respectively, but

adapted to the non-Euclidean and non-binary tree setting, and further adapted to make use of bounds on the product kernel (see section 3.2). It proceeds by a recursive descent down the tree, where at each non-leaf node it computes upper and lower bounds on the weight of any descendant, and applies a cutoff rule to determine whether to continue the descent. Many cutoff rules are possible; for predictive mean calculation, Moore et al. (1997) and Shen et al. (2006) maintain an accumulated lower bound on the total overall weight, and cut off whenever the difference between the upper and lower weight bounds at the current node is a small fraction of the lower bound on the overall weight. However, our setting differs from theirs: since we are computing a weighted sum over entries of K_y^{-1}, which we expect to be approximately sparse, we expect that some entries will contribute much more than others. Thus we want our cutoff rule to account for the weights of the sum *and* the entries of K_y^{-1} that are being summed over. We do this by defining a rule in terms of the current running weighted sum,

$$w_{\max} \cdot S_{\mathbf{n}}^{\mathrm{Abs}} \leq \left(\epsilon_{\mathrm{rel}} \left|\hat{S} + w_{\min} \cdot S_{\mathbf{n}}^{\mathrm{UW}}\right| + \epsilon_{\mathrm{abs}}\right), \quad (6)$$

which we have found to significantly improve performance in covariance calculations compared to the weight-based rule of Moore et al. (1997) and Shen et al. (2006). Here $\hat{S}$ is the weighted sum accumulated thus far, and ϵ_{abs} and ϵ_{rel} are tunable approximation parameters. We interpret the left-hand side of (6) as computing an upper bound on the contribution of node $\mathbf{n}$'s descendents to the final sum, while the absolute value on the right-hand side gives an estimated lower bound on the magnitude of the final sum (note that this is not a true bound, since the sum may contain both positive and negative terms, but it appears effective in practice). If the leaves below the current node $\mathbf{n}$ appear to contribute a negligible fraction of the total sum, we approximate the contribution from $\mathbf{n}$ by $\frac{1}{2}(w_{\max} + w_{\min}) \cdot S_{\mathbf{n}}^{\mathrm{UW}}$, i.e., by the average weight times the unweighted sum. Otherwise, the computation continues recursively over $\mathbf{n}$'s children. Following Shen et al. (2006), we recurse to child nodes in order of increasing distance from the query point, so as to accumulate large sums early on and increase the chance of cutting off later recursions.

3.1 Implementation

A naïve product tree on n points will have n^2 leaves, but we can reduce this and achieve substantial speedups by exploiting the structure of K_y^{-1} and of the product space $\mathcal{M} \times \mathcal{M}$:

Sparsity. If K_y is sparse, or can be well-approximated by a sparse matrix, then K_y^{-1} is often also sparse (or well-approximated as sparse) in practice. This occurs in the case of compactly supported kernel functions (Gneiting, 2002; Rasmussen and Williams, 2006), but also even when using standard kernels with short lengthscales. Note that although there is no guarantee that the inverse of a sparse matrix must itself be sparse (with the exception of specific structures, e.g., block diagonal matrices), it is often the case that when K_y is sparse many entries of K_y^{-1} will be very near to zero, since points with negligible covariance generally also have negligibly small correlations in the precision matrix, so K_y^{-1} can often be well-approximated as sparse. When this is the case, our product tree need include only those pairs $(\mathbf{x}_p, \mathbf{x}_q)$ for which $(K_y^{-1})_{pq}$ is non-negligible. This is often a substantial advantage.

Symmetry. Since K_y^{-1} is a symmetric matrix, it is redundant to include leaves for both $(\mathbf{x}_p, \mathbf{x}_q)$ and $(\mathbf{x}_q, \mathbf{x}_p)$ in our tree. We can decompose $K_y^{-1} = U + D + U^T$, where $D = \mathrm{diag}(K_y^{-1})$ is a diagonal matrix and $U = \mathrm{triu}(K_y^{-1})$ is a strictly upper triangular (zero diagonal) matrix. This allows us to rewrite

$$\mathbf{k_i^*}^T K_y^{-1} \mathbf{k_j^*} = \mathbf{k_i^*}^T U \mathbf{k_j^*} + \mathbf{k_i^*}^T D \mathbf{k_j^*} + \mathbf{k_i^*}^T U^T \mathbf{k_j^*},$$

in which the first and third terms can be implemented as calls to WeightedMetricSum on a product tree built from U; note that this tree will be half the size of a tree built for K_y^{-1} since we omit zero entries. The second (diagonal) term can be computed using a separate (very small) product tree built from the nonzero entries of D. The accumulated sum $\hat{S}$ can be carried over between these three computations, so we can speed up the later computations by accumulating large weights in the earlier computations.

Factorization of product distances. In general, computing the product distance δ will usually involve two calls to the underlying distance metric d; these can often be reused. For example, when calculating both $\delta((\mathbf{x}_a, \mathbf{x}_b), (\mathbf{x}_c, \mathbf{x}_d))$ and $\delta((\mathbf{x}_a, \mathbf{x}_e), (\mathbf{x}_c, \mathbf{x}_d))$, we can reuse the value of $d(\mathbf{x}_a, \mathbf{x}_c)$ for both computations. This reduces the total number of calls to the distance function during tree construction from a worst-case n^4 (for all pairs of pairs of training points) to a maximum of n^2, and in general much fewer if other optimizations such as sparsity are implemented as well. This can dramatically speed up tree construction when the distance metric is slow to evaluate. It can also speed up test-time evaluation, if distances to the same point must be computed at multiple levels of the tree.

3.2 Other Kernel Functions

As noted above, the SE kernel has the lucky property that, if we choose product metric $\delta = \sqrt{d_1^2 + d_2^2}$, then the product of two SE kernels is equal to the kernel of

Kernel	$k(d)$	$k(d_1)k(d_2)$	$\delta(d_1,d_2)$	$k^{\text{prod}}_{\text{lower}}(\delta)$	$k^{\text{prod}}_{\text{upper}}(\delta)$
SE	$\exp\left(-d^2\right)$	$\exp\left(-d_1^2-d_2^2\right)$	$\sqrt{d_1^2+d_2^2}$	$\exp\left(-(\delta)^2\right)$	$\exp\left(-(\delta)^2\right)$
γ-exponential	$\exp\left(-d^\gamma\right)$	$\exp\left(-d_1^\gamma-d_2^\gamma\right)$	$\left(d_1^\gamma+d_2^\gamma\right)^{1/\gamma}$	$\exp\left(-(\delta)^\gamma\right)$	$\exp\left(-(\delta)^\gamma\right)$
Rational Quadratic	$\left(1+\frac{d^2}{2\alpha}\right)^{-\alpha}$	$\left(1+\frac{d_1^2+d_2^2}{2\alpha}+\frac{d_1^2 d_2^2}{4\alpha^2}\right)^{-\alpha}$	$\sqrt{d_1^2+d_2^2}$	$\left(1+\frac{(\delta)^2}{2\alpha}+\frac{(\delta)^4}{16\alpha^2}\right)^{-\alpha}$	$\left(1+\frac{(\delta)^2}{2\alpha}\right)^{-\alpha}$
Matérn ($\nu=3/2$)	$\left(1+\sqrt{3}d\right)$ $\cdot\exp\left(-\sqrt{3}d\right]\Big)$	$\left(1+\sqrt{3}\left(d_1+d_2\right)+3d_1d_2\right)$ $\cdot\exp\left(-\sqrt{3}(d_1+d_2)\right)\Big)$	d_1+d_2	$\left(1+\sqrt{3}\delta\right)$ $\cdot\exp\left(-\sqrt{3}\delta\right)$	$\left(1+\sqrt{3}\delta+3(\delta/2)^2\right)$ $\cdot\exp(-\sqrt{3}\delta)$
Piecewise polynomial (compact support), $q=1$, dimension D, $j=\left\lfloor\frac{D}{2}\right\rfloor+2$	$(1-d)_+^{j+1}$ $\cdot((j+1)d+1)$	$\left((1-d_1)_+(1-d_2)_+\right)^{j+1}$ $\cdot\Big((j+1)^2d_1d_2$ $+(j+1)(d_1+d_2)+1\Big)$	d_1+d_2	$(1-\delta)_+^{j+1}$ $\cdot((j+1)\delta+1)$	$\left(1-\delta+\frac{(\delta)^2}{4}\right)_+^{j+1}$ $\cdot\left((j+1)^2\left(\frac{\delta}{2}\right)^2+(j+1)\delta+1\right)$

Table 1: Bounds for products of common kernel functions. All kernel functions are from Rasmussen and Williams (2006).

the product metric δ:

$$k_{SE}(d_1)k_{SE}(d_2)=\exp\left(-d_1^2-d_2^2\right)=k_{SE}(\delta).$$

In general, however, we are not so lucky: it is not the case that every kernel we might wish to use has a corresponding product metric such that a product of kernels can be expressed in terms of the product metric. In such cases, we may resort to upper and lower bounds in place of computing the exact kernel value. Note that such bounds are all we require to evaluate the cutoff rule (6), and that when we reach a leaf node representing a specific pair of points we can always evaluate the exact product of kernels directly at that node. As an example, consider the Matérn kernel

$$k_{\text{M}}(d)=(1+\sqrt{3}d)\exp(-\sqrt{3}d)$$

(where we have taken $\nu=3/2$); this kernel is popular in geophysics because its sample paths are once-differentiable, as opposed to infinitely smooth as with the SE kernel. Considering the product of two Matérn kernels,

$$k_{\text{M}}(d_1)k_{\text{M}}(d_2)= (1+\sqrt{3}(d_1+d_2)+3d_1d_2)\exp(-\sqrt{3}(d_1+d_2))$$

we notice that this is almost equivalent to $k_{\text{M}}(\delta)$ for the choice of $\delta=d_1+d_2$, but with an additional pairwise term of $3d_1d_2$. We bound this term by noting that it is maximized when $d_1=d_2=\delta/2$ and minimized whenever either $d_1=0$ or $d_2=0$, so we have $3(\delta/2)^2\geq 3d_1d_2\geq 0$. This yields the bounds $k^{\text{prod}}_{\text{lower}}$ and $k^{\text{prod}}_{\text{upper}}$ as shown in Table 1. Bounds for other common kernels are obtained analogously in Table 1.

4 Primal / Dual and Mixed GP Representations

In this section, we extend the product tree approach to models combining a long-scale parametric component with a short-scale nonparametric component. We introduce these models, which we refer to as *mixed primal/dual* GPs, and demonstrate how they can mediate between the desire to model long-scale structure and the need to maintain a short lengthscale for efficiency. (Although this class of models is well known, we have not seen this particular use case described in the literature). We then show that the necessary computations in these models can be done efficiently using the techniques described above.

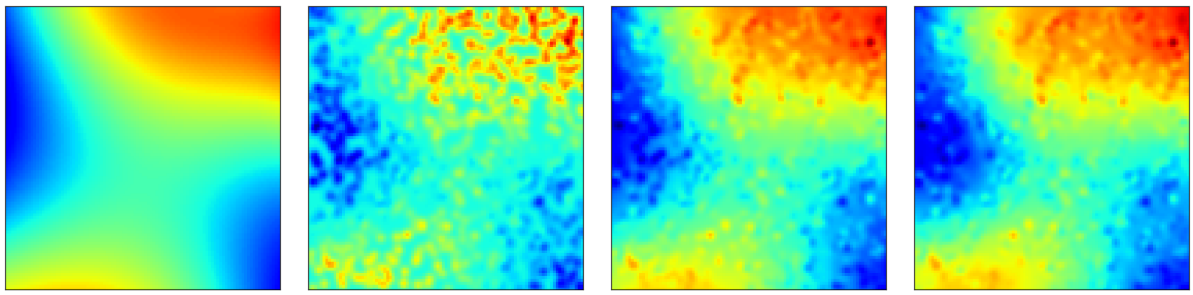
4.1 Mixed Primal/Dual GP Models

Although GP regression is commonly thought of as nonparametric, it is possible to implement parametric models within the GP framework. For example, a Bayesian linear regression model with Gaussian prior,

$$y=\mathbf{x}^T\beta+\epsilon,\ \ \beta\sim\ \mathcal{N}(\mathbf{0},I),\ \ \epsilon\sim\mathcal{N}(0,\sigma_n^2),$$

is equivalent to GP regression with a linear kernel $k(\mathbf{x},\mathbf{x}')=\langle\mathbf{x},\mathbf{x}'\rangle$, in the sense that both models yield the same (Gaussian) predictive distributions (Rasmussen and Williams, 2006). However, the two representations have very different *computational* properties: the primal (parametric) representation allows computation of the predictive mean and variance in $O(D)$ and $O(D^2)$ time respectively, where D is the input dimensionality, while the dual (nonparametric) representation requires time $O(n)$ and $O(n^2)$ respectively for the same calculations. When learning simple models on large, low-dimensional (e.g., spatial) data sets, the primal representation is obviously more attractive, since we can store and compute with model parameters directly, in constant time relative to n.

Of course, simple parametric models by themselves cannot capture the complex local structure that often appears in real-world datasets. Fortunately it is possible to combine a parametric model with a nonparametric GP model in a way that retains the advantages of both approaches. To define a combined model, we replace the standard zero-mean GP assumption with a



(a) *MAD*= 1.11
polynomial(5)

(b) *MAD*= 1.77
GP w/ $\ell = 0.02$

(c) *MAD*= 0.73
Mixed:
polynomial(5) +
GP w/ $\ell = 0.02$

(d) *MAD*= 0.65
True GP w/
$\ell_1 = 0.5$
$\ell_2 = 0.02$

Figure 3: A primal/dual mixture approximating a longer-scale GP.

parametric mean function $\mathbf{h}(\mathbf{x})^T\beta$, yielding the model

$$y = f(\mathbf{x}) + \mathbf{h}(\mathbf{x})^T\beta + \epsilon$$

where $\mathbf{h}(\mathbf{x})$ is a vector of feature values $[h_1(\mathbf{x}), \ldots, h_D(\mathbf{x})]$. The GP model is then learned jointly along with a posterior distribution on the coefficients β. Assuming a Gaussian prior $\beta \sim \mathcal{N}(b, B)$ on the coefficients, the predictive distribution $g(X^*) \sim \mathcal{N}(\mu'_*, \Sigma'_*)$ can be derived (Rasmussen and Williams, 2006) as

$$\mu'_* = H^{*T}\bar{\beta} + K^{*T}K_y^{-1}(\mathbf{y} - H^{*T}\bar{\beta}) \quad (7)$$

$$\Sigma'_* = K^{**} - K^{*T}K_y^{-1}K^* + R^T(B^{-1} + HK_y^{-1}H^T)R \quad (8)$$

where we define $H_{ij} = h_j(\mathbf{x}_i)$ for each training point x_i, similarly H^* for the test points, and we have $\bar{\beta} = (B^{-1} + HK_y^{-1}H^T)^{-1}(HK_y^{-1}\mathbf{y} + B^{-1}\mathbf{b})$ and $R = H^* - HK_y^{-1}K^*$. Section 2.7 of Rasmussen and Williams (2006) gives further details.

Note that linear regression in this framework corresponds to a choice of basis functions $h_1(x) = 1$ and $h_2(x) = x$; it is straightforward to extend this to polynomial regression and other models that are linear in their parameters. In general, any kernel which maps to a finite-dimensional feature space can be represented parametrically in that feature space, so this framework can efficiently handle kernels of the form $k(\mathbf{x}, \mathbf{x}') = \sum_i k_i(\mathbf{x}, \mathbf{x}') + k_S(\mathbf{x}, \mathbf{x}')$, where k_S is a short-lengthscale or compactly supported kernel, monotonically decreasing w.r.t. some distance metric as assumed above, and each k_i either has an exact finite-dimensional feature map or can be approximated using finite-dimensional features Rahimi and Recht (2007); Vedaldi and Zisserman (2010).

As an example, Figure 3 compares several approaches for inferring a function from a GP with long and short-lengthscale components. We drew training data from a GP with a mixture of two SE kernels at lengthscales $\ell_1 = 0.5$ and $\ell_2 = 0.02$, sampled at 1000 random points in the unit square. Figure 3 displays the posterior means of four models on a 100 by 100 point grid, reporting the mean absolute deviation (MAD) of the model predictions relative to the "true" values (drawn from the same GP) at 500 random test points. Note that although the short-scale GP (3b) cannot by itself represent the variation from the longer-scale kernel, when combined with a parametric polynomial component (3a) the resulting mixed model (3c) achieves accuracy approaching that of the true model (3d).

4.2 Efficient Operations in Primal/Dual Models

Likelihood calculation in primal/dual models is a straightforward extension of the standard case. The predictive mean (7) can be accommodated within the framework of Shen et al. (2006) using a tree representation of the vector $K_y^{-1}\left(\mathbf{y} - H^{*T}\bar{\beta}\right)$, then adding in the easily evaluated parametric component $H^{*T}\bar{\beta}$. In the covariance (8) we can use a product tree to approximate $K^{*T}K_y^{-1}K^*$ as described above; of the remaining terms, $\bar{\beta}$ and $B^{-1} + HK_y^{-1}H^T$ can be precomputed at training time, and H^* and K^{**} don't depend on the training set. This leaves $HK_y^{-1}K^*$ as the one remaining challenge; we note that this quantity can be computed efficiently using mD applications of the vector multiplication primitive from Shen et al. (2006), re-using the same tree structure to multiply each column of K^* by each row of HK_y^{-1}. Thus, all of the the operations required for likelihood computation can be implemented efficiently with no explicit dependence on n (i.e., with no direct access to the training set except through space-partitioning tree structures).

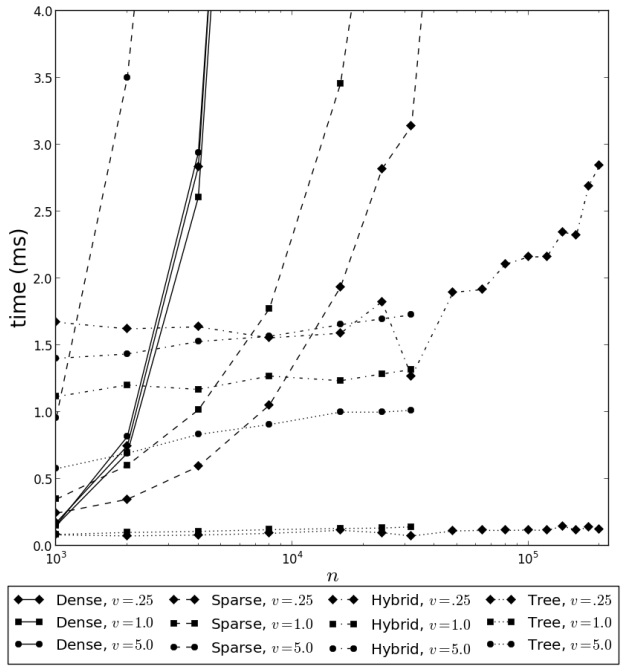


Figure 4: Mean runtimes for dense, sparse, hybrid, and product tree calculation of GP variance on a 2D synthetic dataset.

5 Evaluation

We compare calculation of the predictive variance using a product tree to several other approaches: a naïve implementation using dense matrices, a direct calculation using a sparse representation of K_y^{-1} and dense representation of $\mathbf{k_i^*}$, and a hybrid tree implementation that attempts to also construct a sparse $\mathbf{k_i^*}$ by querying a cover tree for all training points within distance r of the query point $\mathbf{x_i^*}$, where r is chosen such that $k(r')$ is negligible for $r' > r$, and then filling in only those entries of $\mathbf{k_i^*}$ determined to be non-negligible.

Our product tree implementation is a Python extension written in C++, based on the cover tree implementation of Beygelzimer et al. (2006) and implementing the optimizations from section 3.1. The approximation parameters ϵ_{rel} and ϵ_{abs} were set appropriately for each experiment so as to ensure that the mean approximation error is less than 0.1% of the exact variance. All sparse matrix multiplications are in CSR format using SciPy's sparse routines; we impose a sparsity threshold of 10^{-8} such that any entry less than the threshold is set to zero.

Figure 4 compares performance of these approaches on a simple two-dimensional synthetic data set, consisting of points sampled uniformly at random from the unit square. We train a GP on n such points and then measure the average time per point to compute the predictive variance at 1000 random test points. The GP uses an SE kernel with observation noise $\sigma_n^2 = 0.1$ and lengthscale $\ell = \sqrt{v\pi/n}$, where v is a parameter indicating the average number of training points within a one-lengthscale ball of a random query point (thus, on average there will be $4v$ points within two lengthscales, $9v$ within three lengthscales, etc.).

The results of Figure 4 show a significant advantage for the tree-based approaches, which are able to take advantage of the geometric sparsity structure in the training data. The dense implementation is relatively fast on small data sets but quickly blows up, while the sparse calculation holds on longer (except in the relatively dense $v = 5.0$ setting) but soon succumbs to linear growth, since it must evaluate the kernel between the test point and each training point. The hybrid approach has higher overhead but scales very efficiently until about $n = 48000$, where the sparse matrix multiplication's $\Omega(n)$ runtime (Bank and Douglas, 1993) begins to dominate. Conversely, the product tree remains efficient even for very large, sparse datasets, with $v = 0.25$ runtimes growing from 0.08ms at $n = 1000$ to just 0.13ms at $n = 200000$. Due to memory limitations we were unable to evaluate $v = 1.0$ and $v = 5.0$ for values of n greater than 32000.

Our second experiment uses amplitude data from 3105 seismic events (earthquakes) detected by a station in Fitzroy Crossing, Australia; the event locations are shown in Figure 1. The amplitudes are normalized for event magnitude, and the task is to predict the recorded amplitude of a new event given that event's latitude, longitude, and depth. Here our distance metric is the great-circle distance, and we expect our data to contain both global trends and local structure, since events further away from the detecting station will generally have lower amplitudes, but this may vary locally as signals from a given source region generally travel along the same paths through the earth and are dampened or amplified in the same ways as they travel to the detecting station.

Table 2 considers several models for this data. A simple parametric model, the fifth-order polynomial in event-to-station distance shown in Figure 5, is not very accurate but does allow for very fast variance evaluations. The GP models are more accurate, but the most accurate GP model uses a relatively long lengthscale of 50km, with correspondingly slow variance calculations. Depending on application requirements, the most appealing tradeoff might be given by the mixed model combining a fifth-degree polynomial with a 10km SE GP: this model achieves accuracy close to that of the 50km models, but with significantly faster variance calculations due to the shorter lengthscale, especially when using a product tree.

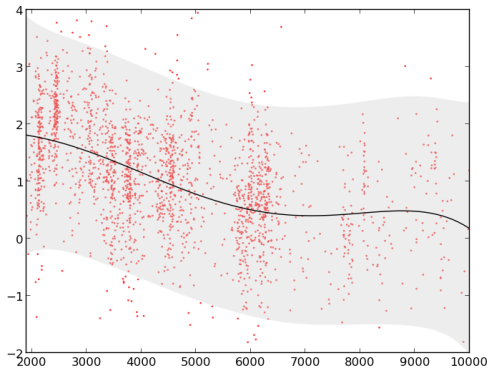


Figure 5: Normalized amplitude as a function of event-station distance, with a fifth-degree polynomial fit shading ±2std.

Model	Error	Sparse (ms)	Tree (ms)
Polynomial in distance (deg 5)	0.78	0.050	n/a
GP, SE, $\ell = 10$km	0.67	0.722 ± 0.032	0.216 ± 0.224
poly/GP, deg 5,SE, 10km	0.62	0.795 ± 0.033	0.413 ± 0.307
GP, Matérn, $\ell = 10$km	0.65	1.256 ± 0.592	0.337 ± 0.365
poly/GP, deg 5, Matérn, 10km	0.62	1.327 ± 0.602	0.654 ± 0.499
GP, SE, $\ell = 50$km	0.61	1.399 ± 0.661	1.168 ± 1.242
poly/GP, deg 5, SE, 50km	0.60	1.490 ± .677	1.551 ± 1.409

Table 2: Models for Fitzroy Crossing amplitude data, with mean absolute prediction error from five-fold cross validation and (mean ± std) time to compute the predictive variance via a direct sparse calculation versus a product tree.

6 Related Work

Previous approximations for GP mean prediction (Moore et al., 1997; Shen et al., 2006; Gray, 2004), which inspired this work, use tree structures to implement an efficient matrix-vector multiplication (MVM); the Improved Fast Gauss Transform (Morariu et al., 2008) also implements fast MVM for the special case of the SE kernel. It is possible to accelerate GP training by combining MVM methods with a conjugate gradient solver, but models thus trained do not allow for the computation of predictive variances. One argument against MVM techniques (and, by extension, our product tree approach) is that their efficiency requires shorter lengthscales than are common in machine learning applications (Murray, 2009); however, we have found them quite effective on datasets which do have genuinely sparse covariance structure (e.g., geospatial data), or in which the longer-scale variation can be represented by a parametric component.

Another set of approaches to speeding up GP regression, sparse approximations (Csató and Opper, 2002; Seeger et al., 2003; Snelson and Ghahramani, 2006; Quiñonero-Candela and Rasmussen, 2005), attempt to represent n training points using a smaller set of m points, allowing training in $O(nm^2)$ time and predictive covariance (thus likelihood) computation in $O(m^2)$ time. This is philosophically a different approach from that of this paper, where we generally want to retain all of our training points in order to represent local structure. However, there is no formal incompatibility: many sparse approaches, including all of those discussed by Quiñonero-Candela and Rasmussen (2005), yield predictive covariances of the form $\mathbf{k}_\mathbf{i}^{*T} Q \mathbf{k}_\mathbf{j}^*$ for some matrix Q (or a sum of terms of this form), where this product could be computed straightforwardly using a product tree. Several non-sparse approximations, e.g., the Nyström approximation (Williams and Seeger, 2001), also yield predictive covariances of this form.

More closely related to our setting are local approximations, in which different GPs are trained in different regions of the input space. There is some evidence that these can provide accurate predictions which are very fast to evaluate (Chalupka et al., 2013); however, they face boundary discontinuities and inaccurate uncertainty estimates if the data do not naturally form independent clusters. Since training multiple local GPs is equivalent to training a single global GP with a block diagonal covariance matrix, it should be possible to enhance local GPs with global parametric components as in section 4, similarly to the combined local/global approximation of Snelson and Ghahramani (2007).

7 Conclusion and Future Work

We introduce the *product tree* structure for efficient adaptive calculation of GP covariances using a multiresolution clustering of pairs of training points. Specific contributions of this paper include product metrics and bounds for common kernels, the adaptation to metric trees, a novel cutoff rule incorporating both the weights and the quantity being summed over, and covariance-specific performance optimizations. Additionally, we describe efficient calculation in GP models incorporating both primal and dual components, and show how such models can model global-scale variation while maintaining the efficiency of short-lengthscale GPs.

A limitation of our approach is the need to explicitly invert the kernel matrix during training; this can be quite difficult for large problems. One avenue for future work could be an iterative factorization of K_y

analogous to the CG training performed by MVM methods (Shen et al., 2006; Gray, 2004; Morariu et al., 2008). Another topic would be a better understanding of cutoff rules for the weighted sum recursion, e.g., an empirical investigation of different rules or a theoretical analysis bounding the error and/or runtime of the overall computation.

Finally, although our work has been focused primarily on low-dimensional applications, the use of cover trees instead of k-d trees ought to enable an extension to higher dimensions. We are not aware of previous work applying tree-based regression algorithms to high-dimensional data, but as high-dimensional covariance matrices are often sparse, this may be a natural fit. For high-dimensional data that do not lie on a low-dimensional manifold, other nearest-neighbor techniques such as locality-sensitive hashing (Andoni and Indyk, 2008) may have superior properties to tree structures; the adaptation of such techniques to GP regression is an interesting open problem.

References

Andoni, A. and Indyk, P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122.

Bank, R. E. and Douglas, C. C. (1993). Sparse matrix multiplication package (SMMP). *Advances in Computational Mathematics*, 1(1):127–137.

Beygelzimer, A., Kakade, S., and Langford, J. (2006). Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 97–104.

Chalupka, K., Williams, C. K., and Murray, I. (2013). A framework for evaluating approximation methods for Gaussian process regression. *Journal of Machine Learning Research*, 14:333–350.

Csató, L. and Opper, M. (2002). Sparse online Gaussian processes. *Neural Computation*, 14(3):641–668.

Friedman, J. H., Bentley, J. L., and Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226.

Gneiting, T. (2002). Compactly supported correlation functions. *Journal of Multivariate Analysis*, 83(2):493–508.

Gray, A. (2004). Fast kernel matrix-vector multiplication with application to Gaussian process learning. Technical Report CMU-CS-04-110, School of Computer Science, Carnegie Mellon University.

Moore, A. W., Schneider, J., and Deng, K. (1997). Efficient locally weighted polynomial regression predictions. In *Proceedings of the 14th International Conference on Machine Learning (ICML)*.

Morariu, V., Srinivasan, B. V., Raykar, V. C., Duraiswami, R., and Davis, L. (2008). Automatic online tuning for fast Gaussian summation. *Advances in Neural Information Processing Systems (NIPS)*, 21:1113–1120.

Murray, I. (2009). Gaussian processes and fast matrix-vector multiplies. In *Numerical Mathematics in Machine Learning workshop at the 26th International Conference on Machine Learning (ICML 2009)*.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *The Journal of Machine Learning Research*, 6:1939–1959.

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems (NIPS)*, 20:1177–1184.

Rasmussen, C. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Seeger, M., Williams, C. K., and Lawrence, N. D. (2003). Fast forward selection to speed up sparse Gaussian process regression. In *Artificial Intelligence and Statistics (AISTATS)*, volume 9.

Shen, Y., Ng, A., and Seeger, M. (2006). Fast Gaussian process regression using kd-trees. In *Advances in Neural Information Processing Systems (NIPS)*, volume 18, page 1225.

Snelson, E. and Ghahramani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems (NIPS)*.

Snelson, E. and Ghahramani, Z. (2007). Local and global sparse Gaussian process approximations. In *Artificial Intelligence and Statistics (AISTATS)*, volume 11.

Uhlmann, J. K. (1991). Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175 – 179.

Vanhatalo, J. and Vehtari, A. (2008). Modelling local and global phenomena with sparse Gaussian processes. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*.

Vedaldi, A. and Zisserman, A. (2010). Efficient additive kernels via explicit feature maps. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3539–3546. IEEE.

Williams, C. and Seeger, M. (2001). Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*. Citeseer.

Latent Topic Analysis for Predicting Group Purchasing Behavior on the Social Web

Feng-Tso Sun, Martin Griss, and Ole Mengshoel
Electrical and Computer Engineering Department
Carnegie Mellon University

Yi-Ting Yeh
Computer Science Department
Stanford University

Abstract

Group-deal websites, where customers purchase products or services in groups, are an interesting phenomenon on the Web. Each purchase is kicked off by a group initiator, and other customers can join in. Customers form communities with people with similar interests and preferences (as in a social network), and this drives bulk purchasing (similar to online stores, but in larger quantities per order, thus customers get a better deal). In this work, we aim to better understand what factors influence customers' purchasing behavior for such social group-deal websites. We propose two probabilistic graphical models, i.e., a product-centric inference model (PCIM) and a group-initiator-centric inference model (GICIM), based on Latent Dirichlet Allocation (LDA). Instead of merely using customers' own purchase history to predict purchasing decisions, these two models include other social factors. Using a lift curve analysis, we show that by including social factors in the inference models, PCIM achieves 35% of the target customers within 5% of the total number of customers while GICIM is able to reach 85% of the target customers. Both PCIM and GICIM outperform random guessing and models that do not take social factors into account.

1 Introduction

Group purchasing is a business model that offers various deals-of-the-day and an extra discount depending on the size of the purchasing group. After group-deal websites, such as Groupon and LivingSocial, have gained attention, similar websites, such as **ihergo**[1] and **Taobao**,[2] have introduced social networks as a feature for their users. These group-deal websites provide an interesting hybrid of social networks (e.g., Facebook.com and LinkedIn.com) and online stores (e.g., Amazon.com and Buy.com). Customers form communities with people with similar interests and preferences (as in a social network), and this drives bulk purchasing (similar to online stores, but in larger quantities per order, thus customers get a better deal). As we see more and more social interactions among customers in group-deal websites, it is critical to understand the interplay between social factors and purchasing preferences.

In this paper, we analyze a transactional dataset from the largest social group-deal website in Taiwan, ihergo.com. Figure 1 shows a screenshot from the group-deal page of ihergo.com. Each group-purchasing event on ihergo.com consists of three major components: (1) a group initiator, (2) a number of group members, and (3) a group-deal product. A group initiator starts a group-purchasing event for a specific group-deal product. While this event will be posted publicly, the group initiator's friends will also be notified. A user can choose to join the purchasing event to become a group member.

Group initiators play important roles on this kind of group-deal websites. Usually, the merchants would offer incentives for the group initiators to initiate group-purchasing events by giving them products for free if the size of the group exceeds some threshold. In addition, to save shipping costs, the group can choose to have the whole group-deal order shipped to the initiator. In this case, the initiator would need to distribute the products to group members in person. Hence, the group members usually reside or work in the proximity of the group initiator. Sometimes, they are friends or co-workers of the initiator.

Understanding customers' purchasing behavior in this

[1] http://www.ihergo.com

[2] http://www.taobao.com

kind of social group-purchasing scenario could help group-deal websites strategically design their offerings. Traditionally, customers search for or browse products of their interests on websites like Amazon.com. However, on social group-deal websites, customers can perform not only product search, but they can also browse group deals and search for initiators by ratings and locations. Therefore, a good recommender system [1] for social group-deal websites should take this into account. If the website can predict which customers are more likely to join a group-purchasing event started by a specific initiator, it can maximize group sizes and merchants' profits in a shorter period of time by delivering targeted advertising. For example, instead of spamming everyone, the website can send out notifications or coupons to the users who are most likely to join the group-purchasing events.

In this work, we aim to predict potential customers who are most likely to join a group-purchasing event. We apply Latent Dirichlet Allocation (LDA) [2] to capture customers' purchasing preferences, and evaluate our proposed predictive models based on a one-year group-purchasing dataset from ihergo.com.

Our contributions in understanding the importance of social factors for group-deal customers' decisions are the following:

- **A new type of group-purchasing dataset.** We introduce and analyze a new type of group-purchasing dataset, which consists of 5,602 users, 26,619 products and 13,609 group-purchasing events.
- **Predictive models for group-deal customers.** Based on topic models, we propose two predictive models that include social factor. They achieve higher prediction accuracy compared to the baseline models.

In the next section, we describe related work in the area of group purchasing behavior, social recommendations, and topic models for customer preferences. Section 3 introduces and analyzes the characteristics of our real-world group-purchasing dataset. In Section 4, we first review LDA, then present two proposed predictive models for group-deal customer prediction. Experimental results are given in Section 5. Finally, conclusion and future research direction are presented in Section 6.

2 Related Work

In this section, we review related work in three areas: (1) group purchasing behavior, (2) social recommendations, and (3) topic models for customer preferences.

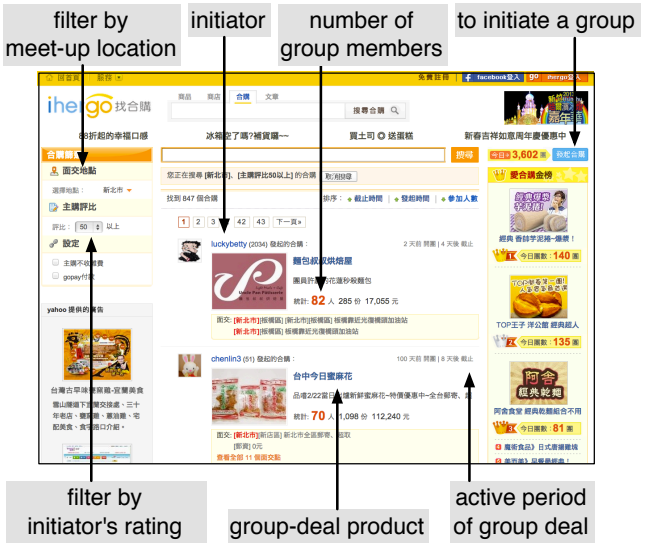


Figure 1: Screenshot of the group-deal page from ihergo.com.

Group Purchasing Behavior. Since group-deal websites such as Groupon and LivingSocial gained attention, several studies have been conducted to understand factors influencing group purchasing behavior. Byers et al. analyzed purchase histories of Groupon and LivingSocial [3]. They showed that Groupon optimizes deal offers strategically by giving "soft" incentives, such as deal scheduling and duration, to encourage purchases. Byers et al. also compared Groupon and LivingSocial sales with additional datasets from Yelp's reviews and Facebook's like counts [4]. They showed that group-deal sites benefit significantly from word-of-mouth effects on users' reviews during sales events. Edelman et al. studied the benefits and drawbacks of using Groupon from the point of view of the merchants [6]. Their work modeled whether advertising and price discrimination effects can make discounts profitable. Ye et al. introduced a predictive dynamic model for group purchasing behavior. This model incorporates social propagation effects to predict the popularity of group deals as a function of time [19]. In this work, we focus on potential customer prediction, as opposed to modeling the overall deal purchasing sales over time.

Social Recommendations. In real life, a customer's purchasing decision is influenced by his or her social ties. Guo et al. analyzed the dataset from the largest Chinese e-commerce website, Taobao, to study the relationship between information passed among buyers and purchasing decision [7]. Leskovec et al. used a stochastic model to explain the propagation of recommendations and cascade sizes [11]. They showed

that social factors have a different level of impact on user purchasing decision for different products. Moreover, previous work also tried to incorporate social information into existing recommendation techniques, such as collaborative filtering [13, 14, 20, 12]. Recently, many recommendation systems have been implemented, taking advantage of social network information in addition to users' preferences to improve recommendation accuracy. For example, Yang et al. proposed a Bayesian-inference based movie recommendation system for online social networks [18]. Our work considers the relationship between the group initiator and the group members as a social tie to augment customer prediction for group-purchasing events.

Topic Models for Customer Preference. Topic models such as LDA have been widely and successfully used in many applications including language modeling [2], text mining [17], human behavior modeling [9], social network analysis [5], and collaborative filtering [8]. Researchers have also proposed new topic models for purchasing behavior modeling. For example, topic models have been extended with price information to analyze purchase data [10]. By estimating the mean and the variance of the price for each product, the proposed model can cluster related items by taking their price ranges into account. Iwata and Watanabe proposed a topic model for tracking time-varying consumer purchase behavior, in which consumer interests and item trends change over time [9]. In this paper, we use LDA to learn topic proportions from purchase history to represent customers' purchasing preferences.

3 Group-Purchasing Dataset

The dataset for our data analysis comes from users' transactional data of a group-deal website, ihergo. It is the largest social group-deal website in Taiwan. We collected longitudinal data between October 1st 2011 and October 1st 2012. From the users' geographical profile, we are able to group them based on their living area. For this study, we include all 5,602 users living in Taipei, the capital of Taiwan. In total, our dataset contains 26,619 products and 13,609 group-purchasing events.

On ihergo, users can purchase a product by joining a group-purchasing event. There are two roles among the users: 1) the **group initiator** and 2) the **group member**. A group initiator initiates a purchase group which other users can join to become group members. Once the group size exceeds some threshold, the group members can get a discount on the product while the initiator can get the product for free. Sometimes the group initiator and the group members already know each other before they join the same group-purchasing event. Sometimes they become friends after the event. Moreover, each user can become a follower of a group initiator. When a new group-purchasing event is initiated by an initiator, the system will notify his or her followers.

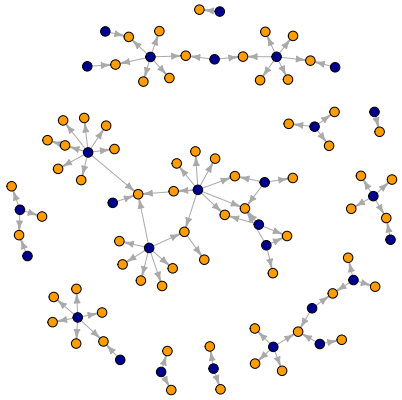


Figure 2: Part of group deal graph for ihergo dataset: illustration of the member-centric relationships between group members and initiators from a subset of randomly sampled joined group members. A directed edge is from a joined customer (dark blue) to an initiator (light orange).

Each group-purchasing deal in our dataset is composed of a set of attributes: the product description (e.g., discounted price, limited quantity, and product category), the group size, the group initiator, the group members, and the time period in which the deal is active. Group-purchasing deals are defined by a time series $\boldsymbol{D} = (D_1, D_2, ..., D_n)$, where D_i is a tuple $(t, p, o, \mathbf{m})$ denoting that a group-purchasing deal for product p is initiated by an organizer (initiator) o with joined group members $\mathbf{m} = \{m_1, ..., m_k\}$ at time t.

We represent group-purchasing events as a directed graph. Each user is a vertex in the graph. For every group-purchasing deal, we build directed edges from each group member to the initiator. There are 5,602 vertices and 16,749 edges in our ihergo dataset. The directed edges are defined by $E = \cup_{i\in[1,n]} \cup_{j\in[1,d(i)]} (m_{i,j}, o_i)$, where $d(i)$ is the number of joined customers for group deal i. The vertices in the graph are defined by $V = M \cup O$, where M denotes all group members $M = \mathbf{m_1} \cup \ldots \cup \mathbf{m_n}$ and O denotes total group-purchasing organizers (initiators) $O = \{o_1\} \cup \ldots \cup \{o_n\}$.

Figure 2 illustrates the joined customer centric graph structure by showing the relationships among a subset of randomly sampled joined customers. Light orange and dark blue vertices represent the group initiators and group members, respectively. According to this dataset, each user has joined 84 group-purchasing

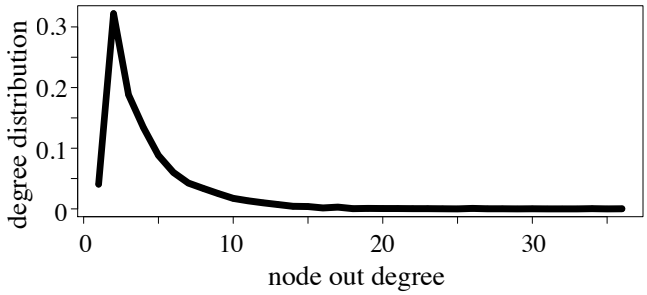


Figure 3: Node out-degree distribution for the group-purchasing graph of ihergo. 80% of the users follow five or fewer initiators.

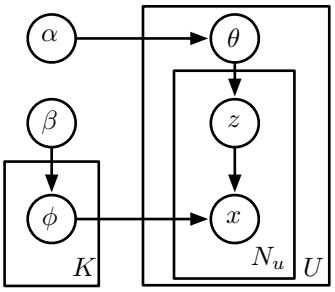


Figure 4: Graphical model representation of the latent Dirichlet allocation model.

events on average. However, one interesting observation from the graph is that the number of outgoing edges from dark blue vertices is far less than 84. This property can be even clearly seen from the out-degree distribution for the overall group-purchasing graph shown in Figure 3. We see that 80% of the users only join group-purchasing events initiated by 5 or fewer different initiators. Group members have a tendency to repeatedly join group-purchasing initiated by a relatively small number of initiators they co-bought with before.

Therefore, we hypothesize that customers' purchasing decisions are not only influenced by their own purchasing preferences but also strongly influenced by who the group initiator is. In the next section, we propose two new models to predict which customers are most likely to join a particular group-purchasing event.

4 Methodology

In this section, we first describe in Section 4.1 how we apply topic modeling to learn user purchasing preferences under the group-purchasing scenario. During the training phase, we compute for each user a mixture topic proportion by combining topic proportions of this user and the initiators with whom this user has co-bought products.

Given a new group-purchasing event, we would like to predict which customers are more likely to join. We propose two predictive models in Section 4.2. One model, which we denote as the product-centric inference model (PCIM), computes the posterior probability that a user would purchase this product given his or her mixture topic proportion. The other model, which we denote the group initiator centric inference model (GICIM), computes the posterior probability that a user would join the group-purchasing event initiated by this initiator given user's or initiator's mixture topic proportion.

4.1 Topic Model for User Purchasing Preference

We use topic modeling to characterize a user's purchasing preference. In particular, we apply LDA to our group-purchasing dataset. In a typical LDA model for text mining [2], a document is a mixture of a number of hidden topics which can be represented by a multinomial distribution, i.e. the topic proportion. A word can belong to one or more hidden topics with different probabilities. Figure 4 shows the graphical model for LDA. LDA is a generative model where each word in a document is generated by two steps: 1) sample a topic from its topic distribution and 2) draw a word from that topic. One can also use Bayesian inference to learn the particular topic proportion of each document.

In our model, we treat a user's purchase history as a document. Each purchased product can be seen as a word in a document. We make an analogy between text documents and purchasing patterns as shown in Table 1. We replace words with each purchased product and a document is one user's purchasing history. Assume that there are U users in our training data. Let $\mathbf{U}$ denote the set of users. Each user $u \in \mathbf{U}$ has a vector of purchased products $\boldsymbol{x}_u = \{x_{un}\}_{n=1}^{N_u}$ where N_u is the number of products that user u purchased.

The generative process of the LDA model for learning a user's purchasing preferences is described as following. Each user u has his or her own topic proportion (i.e., purchasing preference) $\theta_{\mathbf{u}}$ that is sampled from a Dirichlet distribution. Next, for each product x_{un} purchased by user u, a topic z_{un} is firstly chosen from the user's topic proportion θ_u. Then, a product x_{un} is drawn from the multinomial distribution $\phi_{z_{un}}$. To estimate θ_u and $\phi_{z_{un}}$, we use the collapsed Gibbs sampling method [15].

Symbol	Description for Group Purchase History	Description for Text Documents
U	Number of users	Number of documents
K	Number of latent topics	Number of latent topics
N_u	Number of purchased products of user u	Number of words of document u
z_{un}	Latent co-purchasing category of nth product	Latent topic of nth word of document u
x_{un}	nth purchased product of user u	nth word of document u
θ_u	Latent co-purchasing category proportion for user u	Topic proportin for document u
ϕ_k	Multinomial distribution over products for topic k	Mult. distribution over words for topic k
α	Dirichlet prior parameters for all θ_u	Dirichlet prior parameters for all θ_u
β	Dirichlet prior parameters for all ϕ_k	Dirichlet prior parameters for all ϕ_k

Table 1: Latent Dirichlet allocation plate model notation

4.2 Proposed Models for Predicting Group-Deal Customers

A group-purchasing event contains two kinds of critical information: who the group initiator is and what the group-deal product is. Our goal is to predict which customers are more likely to join a specific group-purchasing event. Intuitively, one may think that whether a customer would join a group-purchasing event solely depends on what the group-deal product is. However, from our observations in the dataset, we hypothesize a correlation between a customer's purchasing decision and who the group initiator is. Therefore, we would like to study how these two kinds of group-purchasing information affect the prediction accuracy by asking two questions:

1. What is the likelihood that a customer would join the event given what the *group-deal product* is?
2. What is the likelihood that a customer would join the event given who the *group-initiator* is?

This leads to our two proposed predictive models, the product centric inference model (*PCIM*) and the group initiator centric inference model (*GICIM*).

4.2.1 Product Centric Inference Model (PCIM)

Figure 5(a) shows the graphical structure of PCIM. For each user, we train a PCIM. PCIM computes the posterior probability that a user would purchase a product given his or her mixture topic proportion. Let C denote the *user's own topic proportion*, which we learned from LDA. Suppose that this user has joined group-purchasing events initiated by n group initiators, we use $\{I_i\}_{i=1}^n$ to denote the *learned topic proportions of these initiators*. Our model computes the weighted topic proportions of initiators W by linearly combining $\{I_i\}_{i=1}^n$ with the frequency distribution that the user co-bought products with them.

Intuitively, if a user joins a group-purchasing event initiated by a group initiator, they might share similar interests. Therefore, our model characterizes the user's purchasing preferences by a weighting scheme that combines C and W with a weighting parameter w. We use M to denote such a *mixture topic proportion* which encodes the overall purchasing preferences of the user.

Let P denote the *product random variable*. $\Omega(\mathrm{P}) = \{p_1, ..., p_m\}$, where p_i is the product. From each data record $D_i \in \boldsymbol{D}$, we have a tuple (t_i, p_i, o_i, $\mathbf{m_i}$) and know what group-deal product p_i corresponds to a particular group-purchasing event e_i. Our goal is to compute $Pr(P = p_i)$, the probability that the user would join a group-purchasing event e_i to buy a product p_i.

Given the topic proportion C and $\{I_i\}_{i=1}^n$ corresponding to the user and the weighting parameter w, we are able to compute

$$Pr(P) = \sum_{\mathbf{Y}=\mathbf{X}_p \setminus \{P\}} Pr(P, \mathbf{Y}) \tag{1}$$

where $\mathbf{X}_p = \{P, M, C, W, I_1, ..., I_n\}$; P a is product random variable; M is a mixture topic proportion.

To predict which users are more likely to join a group-purchasing event, we rank $\{\mathcal{P}_{p_i}^{(u_1)}, \ldots, \mathcal{P}_{p_i}^{(u_U)}\}$ in descending order where $\{u_1, \ldots, u_U\}$ denotes the set of users in our dataset and $\mathcal{P}_{p_i}^{(u_j)}$ denotes $Pr(P = p_i)$ of user u_j.

4.2.2 Group Initiator Centric Inference Model (GICIM)

The graphical illustration of GICIM is shown in Figure 5(b). GICIM computes the posterior probability that a user would join a group-purchasing event initiated by a particular initiator given user's or initiator's mixture topic proportion. GICIM and PCIM only differ in their leaf nodes. While PCIM considers only what the group-deal product is, GICIM models our observation that the decision of whether or not a user joins a

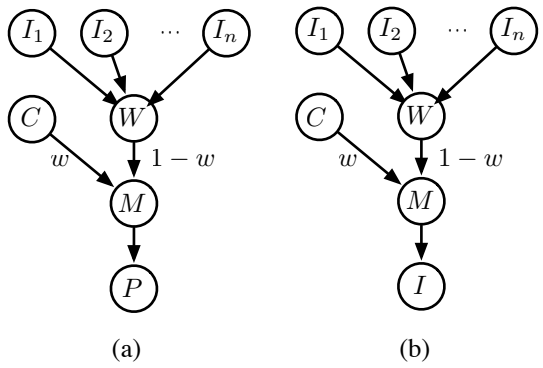


Figure 5: Our proposed models for predicting potential customers given a group-purchasing event. (a) Product centric inference model (PCIM). (b) Group initiator centric inference model (GICIM).

group-purchasing event is strongly influenced by who the group initiator of that event is.

Let I denote the initiator random variable and i_i denote the initiator of the group-purchasing event e_i. Again, from each data record $D_i \in \boldsymbol{D}$, we know who the group initiator is. Instead of evaluating $Pr(P = p_i)$ as in PCIM, we use GICIM to compute

$$Pr(I) = \sum_{\mathbf{Y}=\mathbf{X}_p \setminus \{I\}} Pr(I, \mathbf{Y}) \quad (2)$$

where $\mathbf{X}_p = \{I, M, C, W, I_1, ..., I_n\}$; I is an initiator random variable; M is a mixture topic proportion.

To predict which users are more likely to join a group-purchasing event e_i, we rank $\{\mathcal{P}_{o_i}^{(u_1)}, \ldots, \mathcal{P}_{o_i}^{(u_U)}\}$ in descending order where $\{u_1, \ldots, u_U\}$ denotes the set of users in our dataset and $\mathcal{P}_{o_i}^{(u_j)}$ denotes $Pr(I = o_i)$ of user u_j.

5 Experimental Evaluation

5.1 Data Pre-processing

We evaluate the proposed PCIM and GICIM models with the ihergo group-purchasing dataset. In order to capture meaningful user purchasing preferences, we remove users who purchased fewer than 10 products during the pre-processing step. We use ten-fold cross-validation to generate our training and testing datasets.

5.2 LDA Topic Modeling on Group Purchasing Dataset

To measure the performance of LDA for different number of topics (20, 40, 60, 80, 100) in our group-purchasing dataset, we compute the perplexity. It measures how well the model generalizes and predicts new documents [2]. Figure 6 shows that the perplexity decreases as the number of iteration increases and converges within 200 iterations. In addition, as we increase the number of topics, the perplexity decreases. Unless mentioned specifically, all topic proportions used in our experiments are learned with LDA using 100 topics.

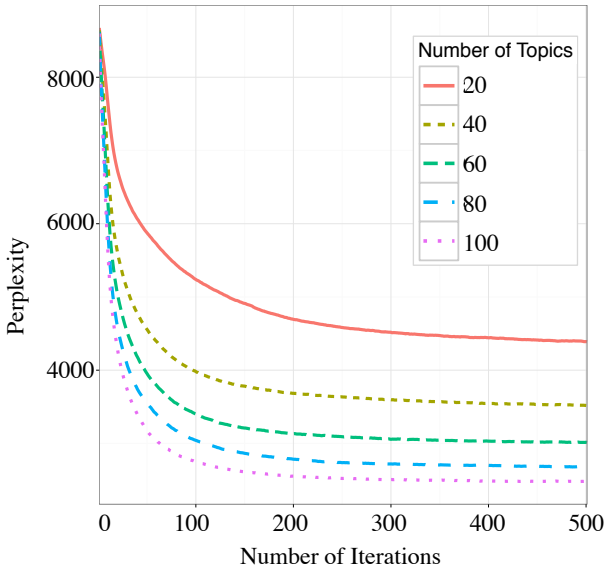
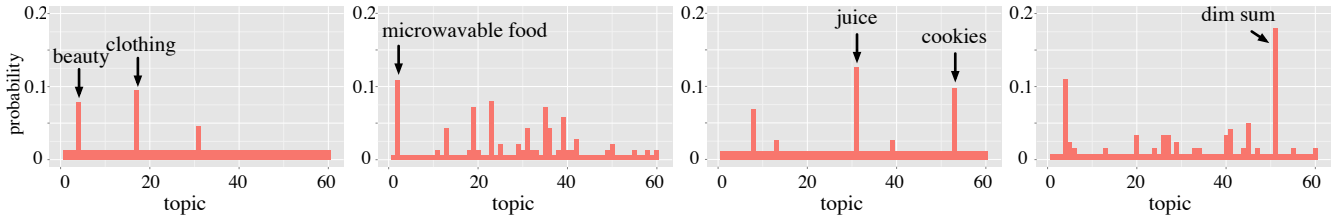


Figure 6: Perplexity as function of collapsed Gibbs sampling iterations for different number of topics used in LDA.

Figure 7 shows three example product topics learned by LDA using 100 topics. Each table shows the ten products that are most likely to be bought in that topic. Columns in the table represent the product name, the probability of the product being purchased in that topic, and the ground-truth category of the product, respectively. We see that Topic 1 is about "pasta." It contains a variety of cooked pasta and pasta sauce. Topic 18 and 53 are respectively about "bread and cakes" and "women accessories."

Figure 8 shows the topic proportions of four randomly selected users learned by LDA using 60 topics. We see that different users have distinguishable topic proportions, representing their purchasing preferences. For example, user #3617 purchased many products that are about "beauty" and "clothing" so her or his topic proportion has higher probabilities at topic 4 and topic 17. Similarly, user #39 tends to buy products in the "dim sum" category which can be represented in her or his topic proportion.



(a) User #3617 (b) User #5 (c) User #2618 (d) User #39

Figure 8: Examples illustrating learned topic proportions of four randomly selected users. For user #3617, the two most probable topics are "beauty" and "clothing".

Product	Prob.	Category
Chicken pasta (cream sauce)	0.0197	*Pasta*
Chicken pasta (pesto sauce)	0.0193	*Pasta*
Pork pasta (tomato sauce)	0.0167	*Pasta*
Pork steak	0.0155	*Meat*
Bacon pasta (cream sauce)	0.0153	*Pasta*
Spicy pasta (tomato sauce)	0.0149	*Pasta*
Clam garlic linguine	0.0146	*Pasta*
Tomato sauce pasta	0.0142	*Pasta*
German sausage sauce	0.0132	*Pasta*
Italian pasta (cooked)	0.0129	*Pasta*

(a) Topic 1, "pasta"

Product	Prob.	Category
Ham sandwich	0.0101	*Bread*
Cheese sandwich	0.0089	*Bread*
Milk bar cookie	0.0080	*Cookie*
Cherry chocolate tart	0.0078	*Cake*
Cheese roll	0.0077	*Cake*
Cheese almond tart	0.0074	*Cake*
Taro toast	0.0073	*Bread*
Creme Brulee	0.0073	*Cake*
Raisin toast	0.0071	*Bread*
Wheat ham sandwich	0.0070	*Bread*

(b) Topic 18, "bread and cakes"

Product	Prob.	Category
Knit Hat	0.0169	*Accessory*
Knit Scarf	0.0165	*Accessory*
Legging	0.0133	*Clothing*
Wool scarf	0.0120	*Accessory*
Long Pant	0.0111	*Clothing*
Cotton Socks	0.0099	*Accessory*
Wool Gloves	0.0097	*Accessory*
Facial Masks	0.0090	*Body Care*
Wool socks	0.0088	*Accessory*
Brown knit scarf	0.0081	*Accessory*

(c) Topic 53, "women accessories"

Figure 7: Illustration of product topics learned by LDA using 100 topics. Category is from ground truth.

5.3 Performance of PCIM and GICIM

We use lift charts to measure the effectiveness of PCIM and GICIM for predicting group-purchasing customers. In a lift chart, the x-axis represents the percentage of users sorted by our prediction score and the y-axis represents the cumulative percentage of the ground-truth customers we would predict. For all lift charts shown in this section, we also include two baseline models for comparison. One baseline model is to predict potential customers by *randomly sampling* from the set of users. Therefore, it is a straight line with slope 1.0 on the lift chart. Another baseline model, which we call category frequency, is to predict customers with the most frequent purchase history in a given product category. Specifically, to predict potential customers given a group-deal product category, we rank each customer in descending order of their normalized purchase frequency for the given product category.

Effect of w. We first measure the effect of the weighting parameter w in PCIM, which is shown in Figure 9. The particular w controls how much the user's own topic proportion is used in the mixture topic proportion. For example, $w = 1$ means that only the user's own topic proportion is used as the mixture topic proportion. We see that for all w values, PCIM performs much better than the baseline models between 0% and 25% of the customers predicted. For instance, PCIM is able to reach 50% of the targeted customers while the two baseline models only reach respectively 25% and 40% of the customers.

We also see that with $w = 1$, the curve first rises very fast, then flattens between 25% and 50%. It even performs worse than the baseline models starting at around 60% of the customers predicted; however this is the least interesting part of the curve. The intuition behind this behavior is that with $w = 1$, PCIM is good at predicting customers who have strong purchasing preferences that match the targeted group-deal product. On the other hand, for users without such strong purchasing preferences, the model is not able to per-

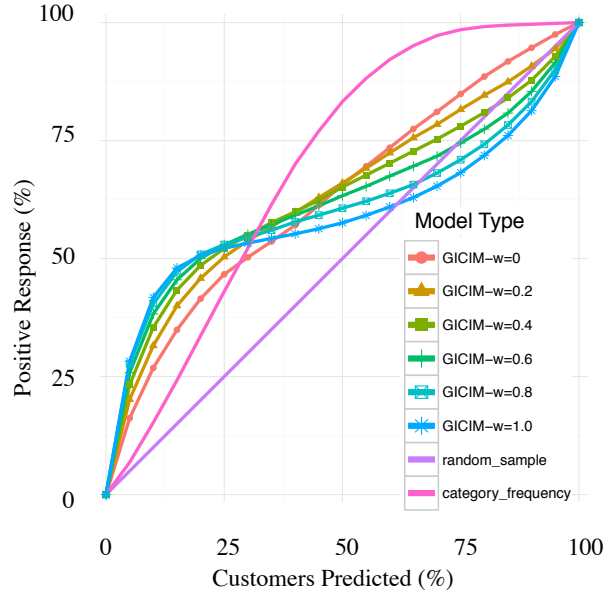


Figure 9: Lift chart of PCIM with different weighting parameter values. With $w = 1$, the model only includes the user's own topic proportion.

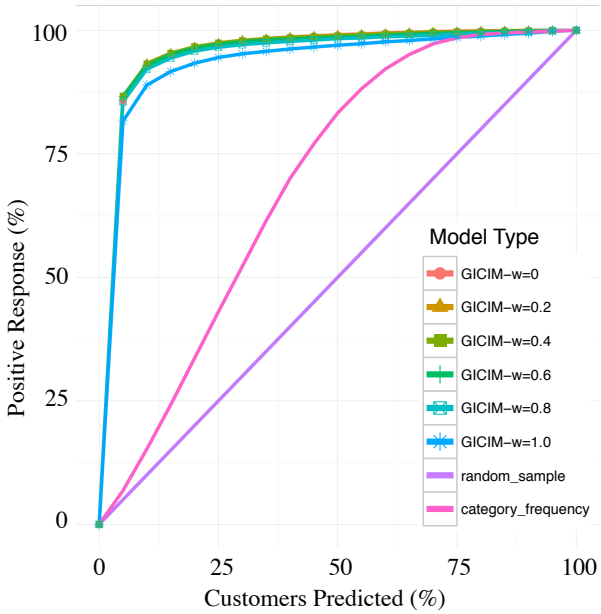


Figure 10: Lift chart of GICIM with different weighting parameter values. With $w = 1$, the model only includes the user's own topic proportion.

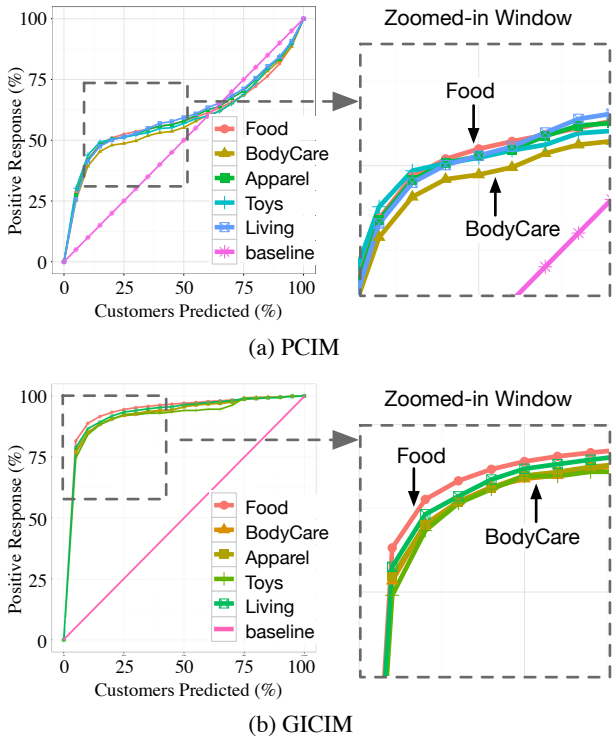
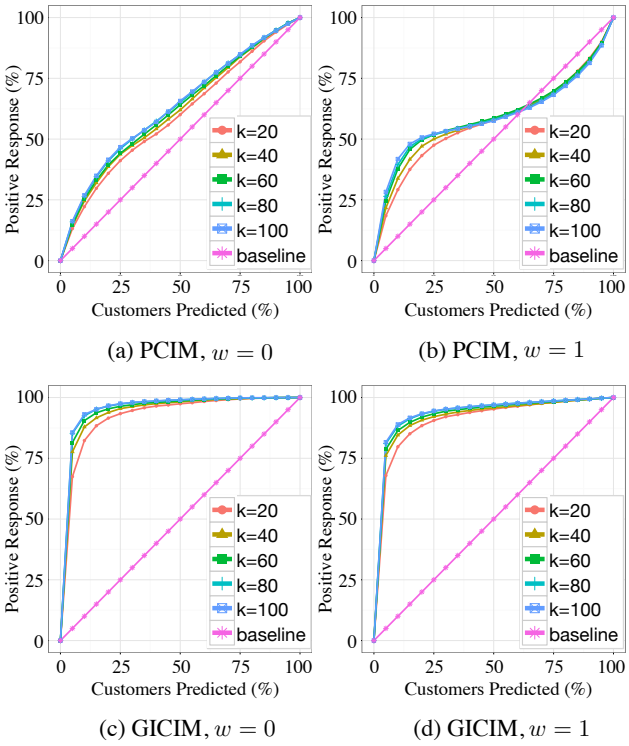


Figure 11: Lift charts of PCIM and GICIM over different product categories. The zoomed-in windows on the right show that performance is slightly better on frequently purchased items (*Food*) than on infrequently purchased items (*Body Care*).

form well. In general, by introducing the topic proportions of initiators with whom the user has co-bought products ($w < 1$), PCIM is able to reduce the flattening effect. With $w = 1$, we see that the lift curve is always above the baseline.

Figure 10 shows the effect of w in GICIM. We see that GICIM always performs better than PCIM and the baseline model even for the case where $w = 1$. In particular, for the cases where $w < 0.8$, GICIM achieves 90% positive response with only 10% of the predicted customers. The high prediction success of GICIM can be explained by the fact that whether a user chooses to join a group-purchasing event or not depends on who the group initiator is. We also note that the performance change due to different w values is not as significant as for PCIM.

Performance on different product categories. We next investigate whether frequently purchased items (e.g., drinks and food items) make PCIM and GICIM perform differently. We test on five different categories of group-deal products: *food*, *body care*, *apparel*, *toys*, and *living*. The ground-truth categories are from the dataset.



(a) PCIM, $w = 0$ (b) PCIM, $w = 1$

(c) GICIM, $w = 0$ (d) GICIM, $w = 1$

Figure 12: Lift charts of PCIM and GICIM over different number of topics.

Figure 11 shows the results. We see that, for all product categories tested, GICIM still performs better than PCIM. Moreover, from Figure 11, we see that both models are slightly better at predicting potential customers for the *food* category than for the *body care* category. We hypothesize that this may be related to the fact that purchases in the *food* category are more frequent and predictable compared to purchases in the *body care* category. A customers may buy one or more products in the *food* category repeatedly, while the same does not appear to be the case for all products in the *body care* category. For example, once someone has purchased a sunscreen spray (a product in *body care*), they are probably unlikely to buy it again, at least for the time span that our dataset covers. However, note that the differences between categories in Figure 11 are small, and developing a better understanding of them is an area of future research.

Effect of different number of topics. We ran PCIM and GICIM on different number of topics used in LDA. Results are given in Figure 12. We find that increasing the number of topics increases prediction accuracy for both models. This agrees with the above perplexity analysis that higher number of topics results in better performance.

6 Conclusion

In this paper, we study group-purchasing patterns with social information. We analyze a real-world group-purchasing dataset (5,602 users, 26,619 products, and 13,609 events) from ihergo.com. To the best of our knowledge, we are the first to analyze the group-purchasing scenario where each group-purchasing event is started by an initiator. Under this kind of social group-purchasing framework, each user builds up social ties with a set of group initiators. Our analysis of the dataset shows that a user usually joins group-purchasing events initiated by a certain and relatively small number of initiators. That is, if a user has co-bought a group-deal product with a group initiator, he or she is more likely to join a group-purchasing event started by that initiator again.

We develop two models to predict which users are most likely to join a group-purchasing event. Experimental results show that by including the weighted topic proportions of the initiators, we achieve higher prediction accuracy. We also find that whether a user decides to join a group-purchasing event is strongly influenced by who the group initiator of that event is.

Our model can be further improved in several ways. First, we can use Labeled LDA [16] by exploiting the ground-truth category of the products or user profile from the dataset. Second, we can incorporate other information such as the geographical and demographic information of users, and the seasonality of products in a more complex topic model. We are also interested in investigating the model to deal with *cold start*, where a new user or group-deal product is added to the system.

Acknowledgments

We want to thank Kuo-Chun Chang from ihergo.com for his cooperation and the use of ihergo dataset. This material is based, in part, upon work supported by NSF award CCF0937044 to Ole Mengshoel.

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6), June 2005.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet allocation. *J. Mach. Learn. Res.*, 3, Mar. 2003.

[3] J. W. Byers, M. Mitzenmacher, M. Potamias, and G. Zervas. A month in the life of Groupon. *CoRR*, 2011.

[4] J. W. Byers, M. Mitzenmacher, and G. Zervas. Daily deals: Prediction, social diffusion, and reputational ramifications. *CoRR*, 2011.

[5] Y. Cha and J. Cho. Social-network analysis using topic models. SIGIR '12, 2012.

[6] B. Edelman, S. Jaffe, and S. D. Kominers. To Groupon or not to Groupon: The profitability of deep discounts. Harvard business school working papers, Harvard Business School, Dec. 2010.

[7] S. Guo, M. Wang, and J. Leskovec. The role of social networks in online shopping: Information passing, price of trust, and consumer choice. *CoRR*, 2011.

[8] T. Hofmann. Collaborative filtering via Gaussian probabilistic latent semantic analysis. SIGIR '03, 2003.

[9] T. Iwata, S. Watanabe, T. Yamada, and N. Ueda. Topic tracking model for analyzing consumer purchase behavior. *IJCAI*, 2009.

[10] T. Iwata, T. Yamada, and N. Ueda. Modeling social annotation data with content relevance using a topic model. *NIPS*, 2009.

[11] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. *ACM Transactions on the Web*, 2007.

[12] T. Lu and C. E. Boutilier. Matching models for preference-sensitive group purchasing. EC '12. ACM, 2012.

[13] H. Ma, I. King, and M. R. Lyu. Learning to recommend with social trust ensemble. SIGIR '09. ACM, 2009.

[14] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. WSDM '11. ACM, 2011.

[15] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling. Fast collapsed Gibbs sampling for latent Dirichlet allocation. KDD '08. ACM, 2008.

[16] D. Ramage, D. Hall, R. Nallapati, and C. D. Manning. Labeled LDA: a supervised topic model for credit attribution in multi-labeled corpora. EMNLP '09. Association for Computational Linguistics, 2009.

[17] H. M. Wallach. Topic modeling: beyond bag-of-words. ICML '06. ACM, 2006.

[18] X. Yang, Y. Guo, and Y. Liu. Bayesian-inference based recommendation in online social networks. *IEEE Transactions on Parallel and Distributed Systems*, 99(PrePrints), 2012.

[19] M. Ye, C. Wang, C. Aperjis, B. A. Huberman, and T. Sandholm. Collective attention and the dynamics of group deals. *CoRR*, 2011.

[20] L. Yu, R. Pan, and Z. Li. Adaptive social similarities for recommender systems. RecSys '11. ACM, 2011.

A Lightweight Inference Method for Image Classification

John Mark Agosta
johnmark.agosta@gmail.com

Preeti J. Pillai
preetipillai04@gmail.com

Abstract

We demonstrate a two phase classification method, first of individual pixels, then of fixed regions of pixels for scene classification—the task of assigning posteriors that characterize an entire image. This can be realized with a probabilistic graphical model (PGM), without the characteristic segmentation and aggregation tasks characteristic of visual object recognition. Instead the spatial aspects of the reasoning task are determined separately by a segmented partition of the image that is fixed before feature extraction. The partition generates histograms of pixel classifications treated as virtual evidence to the PGM. We implement a sampling method to learn the PGM using virtual evidence. Tests on a provisional dataset show good (+70%) classification accuracy among most all classes.

1 Introduction

Scene recognition is a field of computer understanding for classification of scene types by analysis of a visual image. The techniques employed for scene recognition are well known, relying on methods for image analysis and automated inference. The fundamental process is to assign probabilities over a defined set of categories—the scene characteristics—based on analysis of the current visual state. This paper shows the practicability of a lightweight approach that avoids much of the complexity of object recognition methods, by reducing the problem to a sequence of empirical machine learning tasks.

The problem we have applied this to is classification of scene type by analysis of a video stream from a moving platform, specifically from a car. In this paper we address aspects of spatial reasoning—clearly there is also a temporal reasoning aspect, which is not considered here. In figurative terms the problem may be compared with Google's *Streetview*® application. *Streetview's* purpose is to tell you what your surroundings look like by knowing your location. The scene recognition problem is the opposite: to characterize your location from what your surroundings look like.

In this paper we consider a classification scheme for images where the image is subject to classification in multiple categories. We will consider outdoor roadway scenes, and these classification categories:

1. surroundings, zoning, development (urban, residential, commercial, mountainous, etc.)
2. visibility (e.g., illumination and weather),
3. roadway type,
4. traffic and other transient conditions,
5. roadway driving obstacles.

An image will be assigned one label from each of the set of five categories.

1.1 Uses of Scene Classification

There are numerous uses where the automated classification assigned to a scene can help. The purpose of scene classification is to capture the gist of the current view from its assigned category labels. For example, how would you describe a place from what you see? Certainly this is different from what you would know from just the knowledge of your lat-long coordinates. These are some envisioned uses:

- A scene classification provides context. For example in making a recommendation, the context could be to consider the practicality of the request: For instance, "Do you want to get a latte now? This is not the kind of neighborhood for that."

- Supplement search by the local surroundings. For example, "Find me a winery in a built-up area." "Find me a restaurant in a remote place." "Find a park in a less-travelled residential area."
- Coming up with a score for the current conditions. How is the view from this place? How shaded or sunny is the area? What fraction of the surroundings are natural versus artificial? Taking this one step further, given an individual driver's ratings of preferred locations, suggest other desirable routes to take, possibly out of the way from a "best" route.
- Distributed systems could crowd-source their findings about nearby locations to form a comprehensive picture of an area. For example, "How far does this swarm (road-race, parade) extend?"

1.2 Relevant previous work

One of the earliest formulations of image understanding as a PGM is found in Levitt, Agosta, and Binford (1989) and Agosta (1990). The approach assumed an inference hierarchy from object categories to low-level image features, and proposed aggregation operators that melded top-down (predictive) with bottom-up (diagnostic) reasoning over the hierarchy.

The uses of PGMs in computer vision have expanded into a vast range of applications. Just to mention a couple of examples, L. Fei-Fei, Fergus and P. Perona (2003) developed a Bayesian model for learning new object categories using a "constellation" model with terms for different object parts. In a paper that improved upon this, L. Fei-Fei and P. Perona (2005) proposed a Bayesian hierarchical theme model that automatically recognizes natural scene categories such as forest, mountains, highway, etc. based on a generalization of the original texton model by T. Leung and J. Malik (2001) and, L. Fei-Fei R. VanRullen, C. Koch, and P. Perona (2002). In another application of a Bayesian model, Sidenbladh, Black, and Fleet (2000) develop a generative model for the appearance of human figures. Both of these examples apply model selection methods to what are implicitly PGMs, if not explicitly labeled as such.

Computer vision approaches specifically to scene recognition recognize the need to analyze the image as a whole. Hoiem, Efros, and Hebert (2008) approach the problem by combining results from a set of *intrinsic images*, each a map of the entire image for one aspect of the scene. Oliva and Torralba (2006) develop a set of scene-centered global image features that capture the spatial layout properties of the image. Similar to our approach, their method does not require segmentation or grouping steps.

1.3 How Scene Classification differs from Object Recognition

Scene classification implies a holistic image-level inference task as opposed to the task of recovering the identity, presence, and pose of objects within an image. Central to object recognition is to distinguish the object from background of the rest of the image. Typically this is done by segmenting the image into regions of smoothly varying values separated by abrupt boundaries, using a bottoms-up process. Pixels may be grouped into "super-pixels" whose grouping is further refined into regions that are distinguished as part of the foreground or background. Object recognition then considers the features and relationships among foreground regions to associate them with parts to be assembled into the object, or directly with an entire object to be recovered.

Scene classification as we approach it does not necessarily depend on segmenting the image into regions, or identifying parts of the image. Rather it achieves a computational economy by treating the image as a whole; for example, to assign the image to the class of "indoor," "outdoor," "urban landscape," or "rural landscape," etc. from a set of pre-defined categories. We view classification as assigning a posterior to class labels, where the image may be assigned a value over multiple sets of labels; equivalently, the posterior may be a joint distribution over several scene variables.

Despite the lack of a bottoms-up segmentation step in our approach, our method distinguishes regions of the image by a partition that is prior to analyzing the image contents. This could be a fixed partition, which is appropriate for a camera in a fixed location such as a security camera, or it could depend on inferring the geometry of the location from sources distinct from the image contents, such as indicators of altitude and azimuth of the camera. In our case, the prior presumption is that the camera is on the vehicle, facing forward, looking at a road.

The rest of this paper is organized as follows. Section 2 describes the inference procedure cascade; the specific design and learning of the Bayes network PGM is the subject of Section 3, and the results of the learned model applied to classification of a set of images is presented in Section 4.

2 Lightweight inference with virtual evidence

In treating the image as a whole, our approach to inference for scene classification takes place by a sequence of two classification steps:

- First the image's individual pixels are classified, based on pixel level features. This classifier resolves the pixel into one of n discrete types, representing the kind of surface that generated it. In our examples $n = 8$: sky, foliage, building-structure, road-surface, lane, barrier-sidewalk, vehicle, and pedestrian.

- In the second step, the pre-defined partitions are applied to the image and in each partition the pixel types are histogrammed, to generate a likelihood vector for the partition. These likelihoods are interpreted as virtual evidence[1] for the second level image classifier, the scene classifier, implemented as a PGM. The classifier returns an joint distribution over the scene variables, inferred from the partitions' virtual evidence.

There is labeled data for both steps, to be able to learn a supervised classifier for each. Each training image is marked up into labeled regions using the open source *LabelMe* tool, (Russell, Torralba, K. Murphy and Freeman, 2007) and also labeled by one label from each category of scene characteristics. From the region labelings a dataset of pixels, with color and texture as features, and the region they belong to as labels can be created. In the second step we learn the structure and parameters of a Bayes network—a discrete valued PGM—from the set of training images that have been manually labeled with scene characteristics. Each image has one label assigned for each scene characteristic. The training images are reduced to a set of histograms of the predicted labels for the pixels, one for each partition. The supervised data for an image consists of the histogram distributions and the label set.

Scene recognition output is a summarization of a visual input as an admittedly modest amount of information from a input source orders of magnitude greater–even mores than for the object recognition task. From the order of 10^6 pixel values we infer a probability distribution over a small number of discrete scene classification variables. To obtain computational efficiency, we've devised an approach that summarizes the information content of the image in an early stage of the process that is adequate at later stages for the classification task.

2.1 Inference Cascade

The two phases in the inference cascade can be formalized as follows, starting from the pixel image and resulting in a probability distribution over scene characteristics. Consider an image of pixels p_{ij} over $i \times j$, each pixel described by a vector of features $\mathbf{f}_{ij}$. The features are derived by a set of filters, e.g. for color and texture, centered at coordinate (i, j). A pixel-level classifier is a function from the domain of $\mathbf{f}$ to one of a discrete set of n types, $C : \mathbf{f} \rightarrow \{c^{(1)}, \cdots c^{(n)}\}$. The result is an array of classified image pixels.

A pre-determined segmentation, G_m partitions the pixels in the image into M regions by assigning each pixel to one region, $r_m = \{p_{ij} \,|\, p_{ij} \in G_m\}, m = 1 \ldots M$, to form regions that are contiguous sets of pixels. Each region is described by a histogram of the pixel types it contains: $H_m = \left(|C(\mathbf{f}_{ij}) = c^{(1)}|, \cdots |C(\mathbf{f}_{ij}) = c^{(n)}|\right) s.t. \, \mathbf{f}_{ij} \in G_m$, for which we introduce the notation, $H_m = \left(|c_{ij}^{(1)}|_m, \cdots |c_{ij}^{(n)}|_m\right)$, where $|c^{(i)}|_m$ denotes the count of pixels of type $c^{(i)}$ in region m. The scene classifier is a PGM with virtual evidence nodes corresponding to the M regions of the image. See Figure 3. Each evidence node receives virtual evidence in the form of a lambda message, λ_m, with likelihoods in the ratios given by H_m. The PGM model has a subset of nodes $\mathbf{S} = \{S_1, \cdots S_v\}$, distinct from its evidence nodes, for scene characteristic variables, each with a discrete state space. Scene classification is completely described by $\mathsf{P}(\mathbf{S} \,|\, \lambda_1, \cdots \lambda_M)$, the joint of $\mathbf{S}$ when the λ_m are applied, or by a characterization of the joint by the MAP configuration over $\mathbf{S}$, or just the posterior marginals of $\mathbf{S}$.

2.2 Partitions of Pixel-level Data

As mentioned we avoid segmenting the image based on pixel values by using a fixed partition to group classified pixels. We introduce a significant simplification over conventional object recognition methods by using such a segmentation. This makes sense because we are not interested in identifying things that are in the image, but only in treating the image as a whole. For instance in the example we present here, the assumption is that the system is classifying an outdoor roadway scene, with sky above, road below, and surroundings characteristic of the scene to either side. The partitions approximate this division. The image is partitioned symmetrically into a set of twelve wedges, formed by rays emanating from the image center.

For greater efficiency the same method could be applied over a smoothed, or down-sampled image, so that every pixel need not be touched, only pixels on a regular grid. The result of the classification step is a *discrete class-valued image* array. See Figure 2. Despite the classifier ignoring local dependencies, neighboring pixels tend to be classed similarly, and the class-valued

[1] Sometimes called "soft evidence." We prefer the term virtual evidence, since soft evidence is also used to mean an application of Jeffrey's rule of conditioning that can change the CPTs in the network.

image resembles a cartoon version of the original.

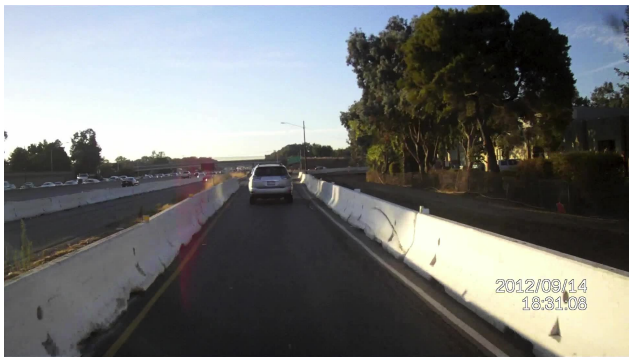


Figure 1: The original image. The barriers bordering the lane are a crucial feature that the system is trained to recognize.

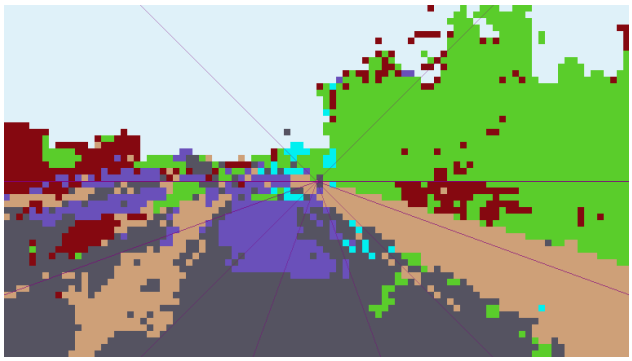


Figure 2: The image array of $C(\mathbf{f}_{ij})$, the pixel classifier, on an image down-sampled to 96 by 54. Rays emanating from the image center show the wedge-shaped regions. Colors are suggestive of the pixel class, e.g. green indicates foliage and beige indicates barriers.

2.2.1 Inferring partition geometry

The point chosen as the image center, where the vertices of the wedges converge approximates the vanishing point of the image. Objects in the roadway scene tend to conform (very) roughly to the wedge outlines so that their contents are more uniform, and hence, likelihoods are more informative. For example, the contents of the image along the horizon will fall within one wedge, and the road surface within another.

2.3 The image as a source of virtual evidence

For each wedge that partitions the image, the evidence applied to the Bayes network from the wedge m is: $\lambda_m \propto |c_{ij}^{(1)}|_m : |c_{ij}^{(2)}|_m : \cdots : |c_{ij}^{(n)}|_m$. One typically thinks of virtual evidence as a consequence of measurements coming from a sensor that garbles the precise value of the quantity of interest—where the actual observed evidence value is obscured by an inaccuracy in the sensor reading. Semantically, one should not think of the virtual image evidence as a garbled sensor variable. Rather it *is* the evidence that describes the region.

3 Bayes network design

Formally a Bayes network is a factorization of a joint probability distribution into local probability models, each corresponding to one node in the network, with directed arcs between the nodes showing the conditioning of one node's probability model on another's (Koller and Friedman, 2010). Inference—for example, classification—operates in the direction against the causal direction of the arc. In short, inference flows from lower level evidence in the network upward to the class nodes at the top of the network where it generates the posterior distributions over the class variables, in this case, the scene characteristics. We learn a fully observable Bayes network with virtual evidence for scene classification.

3.1 How the structure and parameters are defined

The design of the Bayes network model is fluid: It is easily re-learned under different partition inputs, output categories and structural constraints. The ability to easily modify the model to test different kinds of evidence as inputs, or differently defined nodes as outputs is an advantage of this approach. The structure of the model discovers dependencies among the model variables that reveal properties of the domain.

Learning the Bayes network is composed of two aspects; the first, learning the variables' structure, the second, learning the parameters of the variable conditional probability tables. The algorithm used is SMILE's *Bayesian Search* (Druzdzel et al., 1997), a conventional fully observable learning algorithm, with a Bayesian scoring rule used to select the preferred model. Learning structure and parameters occur simultaneously.

The model is structured into two levels, the top level of outputs and the lower level of inputs as shown in Figure 3. This is the canonical structure for classification with a Bayes network, in this case a multi-classifier with multiple output nodes. In the learning procedure this node ordering is imposed as a constraint on the structure, so that conditioning arcs cannot go from the lower level to the upper level.

Further constraints are used to limit in-degree and node ordering. The in-degree of evidence nodes is

limited to two. Node ordering of output nodes follows common sense causal reasoning: for instance, the "Surroundings" variable influences the "Driving Conditions" and not the other way around. The model consequently follows an approximately näive Bayes structure for each scene variable, but with additional arcs that are a consequence of the model selection performed during learning. The resulting network is relatively sparse and hence learning a network of this size, let alone running inference on it can be done interactively.

3.2 Bayes Network Learning Dataset

An interesting challenge in learning this model is that there is no conventional procedure for learning from virtual evidence, such as the histogram data.

3.2.1 Consideration of partition contents as virtual evidence

We considered three ways to approximate learning the Bayes network from samples that include virtual evidence.

1) Convert the dataset into an approximate equivalent observed evidence dataset by generating multiples of each evidence row, in proportion to the likelihood fraction for each state of the virtual evidence. If there are multiple virtual evidence nodes, then to capture dependencies among virtual evidence nodes this could result in a combinatorial explosion of row sets, one multiple for each combination of virtual evidence node states, with multiplicities in proportion to the likelihood of the state combination. This is equivalent in complexity to combining all virtual evidence nodes into one node for sampling.

Similarly one could sample from the combination of all virtual evidence nodes and generate a sample of rows based on the items in the sample. This is a bit like logic sampling the virtual states.

Both these methods make multiple copies of a row in the learning set as a way to emulate a training weight. Instead one could apply a weight to each row in the sampled training set, in proportion to its likelihood.

2) One could also consider a mixture, a "multi-net," of learned deterministic evidence models. The models would have the same structure, so the result would be a mixture of CPTs, weighted (in some way) by the likelihoods. It appears this would also suffer a combinatorial explosion of mixture components, and might be amenable to reducing the set by sampling.

3) Alternatively, one could consider the virtual evidence by a virtual node that gets added as a child to the evidence node, which is then instantiated to send the equivalent lambda msg to its parent. This is the method used in Refaat, Choi and Darwiche (2012). With many cases, there would be a set of virtual nodes added to the network for each case, again generating a possibly unmanageable method. Perhaps there is an incremental learning method that would apply: Build a network with one set of nodes, do one learning step, then replace the nodes with the next set, and repeat a learning step.

4 Results on a sample dataset

In this section we present the evaluation of the Bayes network as a classifier. We argue that the first-stage pixel-level classifier, whose accuracy approaches 90%, is a minor factor in the scene classification results, since the partition-level inputs to the Bayes network average over a large number of pixels, although this premise could be tested.

4.1 Learning from a sampled dataset

The sample dataset to learn the model was a further approximation on alternative 1), where each virtual evidence node was sampled independently to convert the problem into an equivalent one with sampled data. Each histogram was sampled according to its likelihood distribution, to generate a set of conventional evidence samples that approximated the histogram. The result was an expanded dataset that multiplied the number rows by the sample size for each row in the histogram dataset. The resulting dataset description is:

1. Original data set: 122 rows of 12 region histograms of images labeled by 5 scene labels.
2. Each region histogram is sampled 10 times, to generate 1220 rows
3. Final data set of 5 labels and 12 features by 1220 rows

4.2 Inference Results

As mentioned, the second-stage Bayes network classifier infers a joint probability distribution over the set of scene characteristic nodes—the nodes shown in orange in Figure 3. We will evaluate the scene classifier by the accuracy of the predicted marginals, comparing the highest posterior prediction for each scene variable with the true value.[2]

[2]The "dynamic environment" variable is not counted in the evaluation results, since most all labeled data was collected under overcast conditions, making the predicted results almost always correct, and uninteresting.

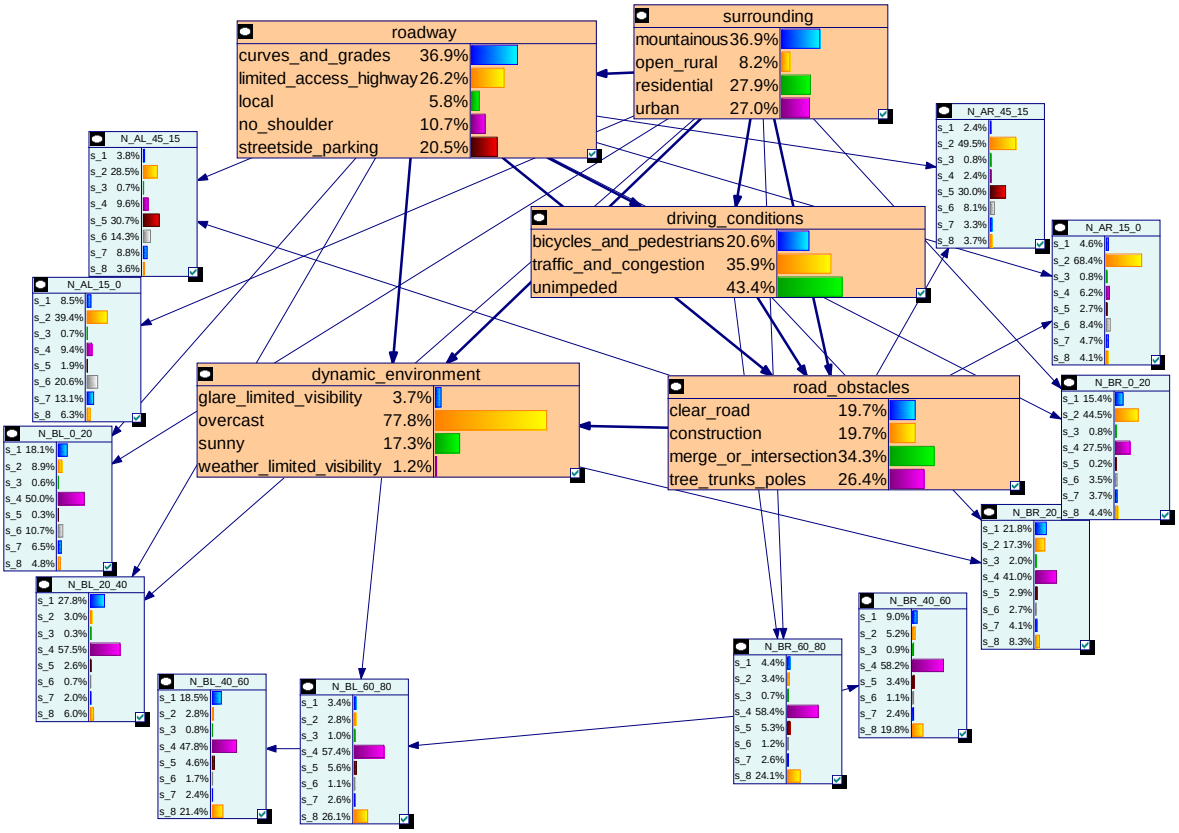


Figure 3: The entire Bayes network used for scene classification. Input nodes, corresponding to the wedges that partition the image are shown in light blue, and output nodes for the scene variables are in orange. The input nodes are arranged roughly in the positions of the corresponding wedges in the image. The input node histograms show the virtual evidence applied from that wedge. The labels used here for virtual evidence states, $s\_1, \ldots s\_8$ correspond to the classifier outputs $c^{(1)}, \ldots c^{(8)}$.

The matrix of counts of the true class by the predicted class is called a confusion matrix. The row sum of the confusion matrix for any class divided into the diagonal (true count) is the fraction of correct cases out of those possible, known as the recall or the coverage for that class. The column sum divided into the diagonal element is the fraction classified with that columns class label that truly belong to that class, which is called the precision. Tables $1-4$ show the recall and precision for each class, for each of the scene variables. As may be expected "Surroundings" that takes in the entire image performs better than "Road obstacles" that requires attention to detail in just the car's lane. This poor performance is even more true with "Bicycles and pedestrians," in Table 3 that appear in small areas of the image. In other classes either precision or recall approach 1.0, except for "Local" roads, where all cases were confused with "Curves and grades," again due to the limited variety in the training set.

Beyond evaluating the accuracy of marginal predictions, we can also make observations about the structure learned for the Bayes network. Arcs in the learned model show which wedge histograms are relevant to which scene variables. These arcs are relatively sparse, in part due to the afore-mentioned design constraint in-degree arc limit of two. The arcs chosen by the structure learning algorithm show a strong association between the location of the partitions, and different scene variables. We see this in the associations where the "Driving conditions" scene variable connects to partitions at the base of the image, and "Surroundings" connects to partitions on the image periphery. The relevance of the two wedges at the bottom of the diagram is limited, since their only incoming arcs are from other wedges, indicating that their evidence is supported entirely by neighboring wedges. We leave them in the model, since in the case of virtual evidence they will still have some information value for classification. Further along these lines, in terms of wedge dependencies, only one arc was learned between wedge histograms, indicating that the evidence contributed

to the scene is conditionally independent in all but this case. The sub-network of scene variables is more connected, indicating strong dependencies among the scene variables. Some of these are to be expected, for instance "Curves and grades" correlates strongly with "Mountainous" surroundings. Some are spurious, as a result of biased selection of the training sample images, (e.g. all divided highway images corresponded to overcast scenes) and have been corrected by adding more samples.

4.3 Discussion and Conclusion

We have demonstrated a novel scene classification algorithm that takes advantage of the presumed geometry of the scene to avoid computationally expensive image processing steps characteristic of object recognition methods, such as pixel segmentation, by a cascade of a pixel level and fixed partition level multi-classifier, for which we learn a Bayes network. As a consequence of the partition-level data we learn the Bayes network with virtual evidence.

The Bayes network classifies the scene in several dependent dimensions corresponding to a set of categories over which a joint posterior of scene characteristics is generated. Here we have only considered the marginals over categories, however it is a valid question whether a MAP interpretation—of the most likely combination of labels—is more appropriate.

The use of virtual evidence also raises questions about whether it is proper to consider the virtual evidence likelihood as a convex combination of "pure" image data. Another interpretation is that the histograms we are using are better"sliced and diced" to generate strong evidence from certain ratios of partition content. For instance a partition that includes a small fraction of evidence of roadway obstacles—think evidence of a small person—may be a larger concern than a partition obviously full of obstacles, and should not be considered a weaker version of the extreme partition contents. These subtleties could be considered as we expand the applicability of the system. In this early work it suffices that given the approximations, useful and accurate results can be achieved at modest computational cost.

Acknowledgements

This work would not have been possible without valuable discussions with Ken Oguchi and Joseph Dugash, and by Naiwala Chandrasiri, Saurabh Barv, Ganesh Yalla, and Yiwen Wan.

References

Agosta, J. M., 1990. The structure of Bayes networks for visual recognition, UAI (1988) North-Holland Publishing Co., pp. 397 - 406.

Druzdzel, M. et. al., 1997. SMILE (Structural Modeling, Inference, and Learning Engine) http://genie.sis.pitt.edu/.

Fei-Fei, L., R. Fergus, P. Perona, 2003. A Bayesian Approach to Unsupervised One-Shot Learning of Object Categories (ICCV 2003), pp. 1134-1141 vol.2.

Fei-Fei L. and P. Perona, 2005. A Bayesian Hierarchical Model for Learning Natural Scene Categories. (CVPR 2005), pp. 524-531.

Fei-Fei, L., R. VanRullen, C. Koch, and P. Perona, 2002. Natural scene categorization in the near absence of attention(PNAS 2002), 99(14):95969601.

Hoiem, D., A. A. Efros, and M. Hebert. 2008. Closing the Loop in Scene Interpretation.2008 IEEE Conference on Computer Vision and Pattern Recognition (June): 18.

Koller, D., and Friedman, N. 2010. Probabilistic Graphical Models: Principles and Techniques. Cambridge, Massachusetts: The MIT Press.

Leung, T. and J. Malik, 2001. Representing and recognizing the visual appearance of materials using three-dimensional textons(IJCV 2001), 43(1):2944.

Levitt, T., J. M. Agosta, T.O. Binford, 1989. Model-based influence diagrams for machine vision, (UAI 1989).

Oliva, A., and A. Torralba. 2006. Building the Gist of a Scene: The Role of Global Image Features in Recognition.Progress in Brain Research 155 (January): 2336.

Oliva, A., A. Torralba, 2001. Modeling the shape of the scene: a holistic representation of the spatial envelope. International Journal of Computer Vision, Vol. 42(3): 145-175.

Refaat, K. S. , A. Choi and A. Darwiche, 2012. New Advances and Theoretical Insights into EDML. (UAI 2012), pp. 705-714 .

Russell, B., A. Torralba, K. Murphy, W. T. Freeman, 2007. "LabelMe: a database and web-based tool for image annotation" International Journal of Computer Vision.

Sidenbladh, H., M. J. Black, and D. J. Fleet, 2000. Stochastic Tracking of 3D Human Figures Using 2D Image Motion. ECCV 2000: 702718.

	Mountainous	Open rural	Residential	Urban
Recall	1.0	0.9	0.794	0.45
Precision	0.642	1.0	0.964	1.0
Accuracy				0.784

Table 1: Surroundings

	Curves and grades	Limited access highway	Local	No shoulder	Streetside parking
Recall	1.0	0.75	0.0	0.85	0.56
Precision	0.61	1.0	NaN	1.0	1.0
Accuracy					0.770

Table 2: Roadways

	Bicycles and pedestrians	Traffic and congestion	Unimpeded
Recall	0.56	0.6	0.98
Precision	0.875	0.93	0.67
Accuracy			0.754

Table 3: Driving Conditions

	Clear road	Construction	Merge intersection	Tree trunks and poles
Recall	0.42	0.96	0.6	1.0
Precision	1.0	0.92	0.86	0.55
Accuracy				0.738

Table 4: Road Obstacles

Exploring Multiple Dimensions of Parallelism in Junction Tree Message Passing

Lu Zheng
Electrical and Computer Engineering
Carnegie Mellon University

Ole J. Mengshoel
Electrical and Computer Engineering
Carnegie Mellon University

Abstract

Belief propagation over junction trees is known to be computationally challenging in the general case. One way of addressing this computational challenge is to use node-level parallel computing, and parallelize the computation associated with each separator potential table cell. However, this approach is not efficient for junction trees that mainly contain small separators. In this paper, we analyze this problem, and address it by studying a new dimension of node-level parallelism, namely *arithmetic parallelism*. In addition, on the graph level, we use a clique merging technique to further adapt junction trees to parallel computing platforms. We apply our parallel approach to both marginal and most probable explanation (MPE) inference in junction trees. In experiments with a Graphics Processing Unit (GPU), we obtain for marginal inference an average speedup of 5.54x and a maximum speedup of 11.94x; speedups for MPE inference are similar.

1 INTRODUCTION

Bayesian networks (BN) are frequently used to represent and reason about uncertainty. The junction tree is a secondary data structure which can be compiled from a BN [2, 4, 5, 9, 10, 19]. Junction trees can be used for both marginal and most probable explanation (MPE) inference in BNs. Sum-product belief propagation on junction tree is perhaps the most popular exact marginal inference algorithm [8], and max-product belief propagation can be used to compute the most probable explanations [2, 15]. However, belief propagation is computationally hard and the computational difficulty increases dramatically with the density of the BN, the number of states of each network node, and the treewidth of BN, which is upper bounded by the generated junction tree [13]. This computational challenge may hinder the application of BNs in cases where real-time inference is required.

Parallelization of Bayesian network computation is a feasible way of addressing this computational challenge [1,6,7,9,11,12,14,19,20]. A data parallel implementation for junction tree inference has been developed for a cache-coherent shared-address-space machine with physically distributed main memory [9]. Parallelism in the basic sum-product computation has been investigated for Graphics Processing Units (GPUs) [19]. The efficiency in using disk memory for exact inference, using parallelism and other techniques, has been improved [7]. An algorithm for parallel BN inference using pointer jumping has been developed [14]. Both parallelization based on graph structure [12] as well as node level primitives for parallel computing based on a table extension idea have been introduced [20]; this idea was later implemented on a GPU [6]. Gonzalez et al. developed a parallel belief propagation algorithm based on parallel graph traversal to accelerate the computation [3].

A parallel message computation algorithm for junction tree belief propagation, based on the cluster-sepset mapping method [4], has been introduced [22]. Cluster-sepset based node level parallelism (denoted *element-wise parallelism* in this paper) can accelerate the junction tree algorithm [22]; unfortunately the performance varies substantially between different junction trees. In particular, for small separators in junction trees, element-wise parallelism [22] provides limited parallel opportunity as explained in this paper.

Our work aims at addressing the small separator issue. Specifically, this paper makes these contributions that further speed up computation and make performance more robust over different BNs from applications:

- We discuss another dimension of parallelism, namely *arithmetic parallelism* (Section 3.1). Inte-

grating arithmetic parallelism with *element-wise parallelism*, we develop an improved parallel sum-product propagation algorithm as discussed in Section 3.2.

- We also develop and test a parallel max-product (Section 3.3) propagation algorithm based on the two dimensions of parallelism.

- On the graph level, we use a clique merging technique (Section 4), which leverages the two dimensions of parallelism, to adapt the various Bayesian networks to the parallel computing platform.

In our GPU experiments, we test the novel two-dimensional parallel approach for both regular sum-propagation and max-propagation. Results show that our algorithms improve the performance of both kinds of belief propagation significantly.

Our paper is organized as follows: In Section 2, we review BNs, junction trees parallel computing using GPUs, and the small-separator problem. In Section 3 and Section 4, we describe our parallel approach to message computation for belief propagation in junction trees. Theoretical analysis of our approach is in Section 5. Experimental results are discussed in Section 6, while Section 7 concludes and outlines future research.

2 BACKGROUND

2.1 Belief Propagation in Junction Trees

A BN is a compact representation of a joint distribution over a set of random variables $\mathcal{X}$. A BN is structured as a directed acyclic graph (DAG) whose vertices are the random variables. The directed edges induce dependence and independence relationships among the random variables. The evidence in a Bayesian network consists of instantiated random variables.

The junction tree algorithm propagates beliefs (or posteriors) over a derived graph called a junction tree. A junction tree is generated from a BN by means of moralization and triangulation [10]. Each vertex C_i of the junction tree contains a subset of the random variables and forms a clique in the moralized and triangulated BN, denoted by $\mathcal{X}_i \subseteq \mathcal{X}$. Each vertex of the junction tree has a potential table $\phi_{\mathcal{X}_i}$. With the above notations, a junction tree can be defined as $J = (\mathbb{T}, \Phi)$, where $\mathbb{T}$ represents a tree and Φ represents all the potential tables associated with this tree. Assuming C_i and C_j are adjacent, a separator S_{ij} is induced on a connecting edge. The variables contained in S_{ij} are defined to be $\mathcal{X}_i \cap \mathcal{X}_j$.

The junction tree size, and hence also junction tree computation, can be lower bounded by *treewidth*, which is defined to be the minimal size of the largest junction tree clique minus one. Considering a junction tree with a treewidth t_w, the amount of computation is lower-bounded by $O(\exp(c*t_w))$ where c is a constant.

Belief propagation is invoked when we get new evidence $\boldsymbol{e}$ for a set of variables $\mathcal{E} \subseteq \mathcal{X}$. We need to update the potential tables Φ to reflect this new information. To do this, belief propagation over the junction tree is used. This is a two-phase procedure: evidence collection and evidence distribution. For the evidence collection phase, messages are collected from the leaf vertices all the way up to a designated root vertex. For the evidence distribution phase, messages are distributed from the root vertex to the leaf vertices.

2.2 Junction Trees and Parallelism

Current emerging many-core platforms, like the recent Graphical Processing Units (GPUs) from NVIDIA and Intel's Knights Ferry, are built around an array of processors running many threads of execution in parallel. These chips employ a Single Instruction Multiple Data (SIMD) architecture. Threads are grouped using a SIMD structure and each group shares a multithreaded instruction unit. The key to good performance on such platforms is finding enough parallel opportunities.

We now consider opportunities for parallel computing in junction trees. Associated with each junction tree vertex $\mathcal{C}_i$ and its variables $\mathcal{X}_i$, there is a potential table $\phi_{\mathcal{X}_i}$ containing non-negative real numbers that are proportional to the joint distribution of $\mathcal{X}_i$. If each variable contains s_j states, the minimal size of the potential table is $|\phi_{\mathcal{X}_i}| = \prod_{j=1}^{|\mathcal{X}_i|} s_j$, where $|\mathcal{X}_i|$ is the cardinality of $\mathcal{X}_i$.

Message passing from $\mathcal{C}_i$ to an adjacent vertex $\mathcal{C}_k$, with separator $\mathcal{S}_{ik}$, involves two steps:

1. **Reduction step**. In sum-propagation, the potential table $\phi_{\mathcal{S}_{ik}}$ of the separator is updated to $\phi^*_{\mathcal{S}_{ik}}$ by reducing the potential table $\phi_{\mathcal{X}_i}$:

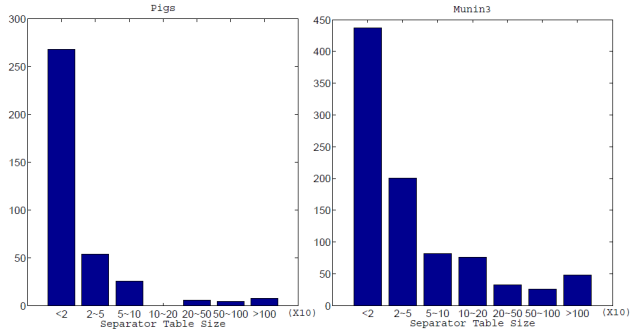
$$\phi^*_{\mathcal{S}_{ik}} = \sum_{\mathcal{X}_i / \mathcal{S}_{ik}} \phi_{\mathcal{X}_i}. \tag{1}$$

2. **Scattering step**. The potential table of $\mathcal{C}_k$ is updated using both the old and new table of $\mathcal{S}_{ik}$:

$$\phi^*_{\mathcal{X}_k} = \phi_{\mathcal{X}_k} \frac{\phi^*_{\mathcal{S}_{ik}}}{\phi_{\mathcal{S}_{ik}}}. \tag{2}$$

We define $\frac{0}{0} = 0$ in this case, that is, if the denominator in (2) is zero, then we simply set the corresponding $\phi^*_{\mathcal{X}_k}$ to zeros.

Figure 1: Histograms of the separator potential table sizes of junction trees *Pigs* and *Munin3*. For both junction trees, the great majority of the separator tables contain 20 or fewer elements.



Equation (1) and (2) reveal two dimensions of parallelism opportunity. The first dimension, which we return to in Section 3, is arithmetic parallelism. The second dimension is element-wise parallelism [22].

Element-wise parallelism in junction trees is based on the fact that the computation related to each separator potential table cell are independent, and takes advantage of an index mapping table, see Figure 2. In Figure 2, this independence is illustrated by the white and grey coloring of cells in the cliques, the separator, and the index mapping tables. More formally, an *index mapping table* $\mu_{\mathcal{X},\mathcal{S}}$ stores the index mappings from $\phi_{\mathcal{X}}$ to $\phi_{\mathcal{S}}$ [4]. We create $|\phi_{\mathcal{S}_{ik}}|$ such mapping tables. In each mapping table $\mu_{\mathcal{X}_i,\phi_{\mathcal{S}_{ik}}(j)}$ we store the indices of the elements of $\phi_{\mathcal{X}_i}$ mapping to the j-th separator table element. Mathematically,

$$\mu_{\mathcal{X}_i,\phi_{\mathcal{S}_{ik}}(j)} = \{r \in [0, |\phi_{\mathcal{X}_i}| - 1] \mid \phi_{\mathcal{X}_i}(r) \text{ is mapped to } \phi_{\mathcal{S}_{ik}}(j)\}.$$

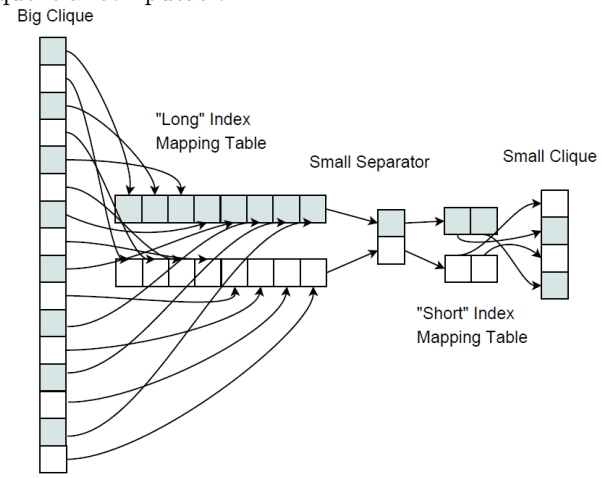
With the index mapping table, element-wise parallelism is obtained by assigning one thread per mapping table of a separator potential table as illustrated in Figure 2 and Figure 3. Consequently, belief propagation over junction trees can often be sped up by using a hybrid CPU/GPU approach [22].

2.3 Small Separator Problem

Figure 1 contains the histograms of two real-world BNs *Pigs* and *Munin3*.[1] We see that most separators in these two BNs are quite small and have a potential table size of less than 20.

In general, small separators can be found in these three scenarios: (i) due to two small neighboring cliques (we call it the *S-S-S* pattern); (ii) due to a small intersection set of two big neighboring cliques (the *B-S-B* pattern); and (iii) due to one small neighboring clique and one big neighboring clique (the *B-S-S* pattern).[2] Due to parallel computing issues, detailed next, these three patterns characterize what we call the *small-separator problem*.

Figure 2: Due to the small separator in the *B-S-S* pattern, a long index mapping table is produced. If only element-wise parallelism is used, there is just one thread per index mapping table, resulting in slow sequential computation.



Unfortunately, element-wise parallelism may not provide enough parallel opportunities when a separator is very small and the mapping tables to one or both cliques are very long. A small-scale example, reflecting the *B-S-S* pattern, is shown in Figure 2.[3] While the mapping tables may be processed in parallel, the long mapping tables result in a significant amount of sequential computation within each mapping table.

A state of the art GPU typically supports more than one thousand concurrent threads, thus message passing through small separators will leave most of the GPU resources idle. This is a major bottleneck for the performance, which we address next.

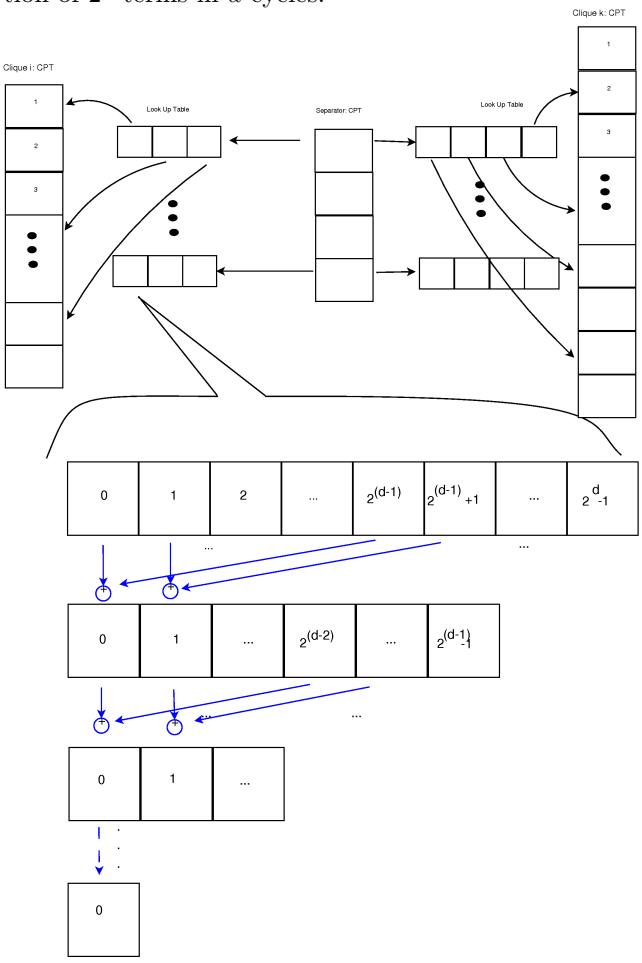
In this paper, to handle the small separator problem, we use clique merging to eliminate small separators (see Section 4) resulting from the *S-S-S* pattern and arithmetic parallelism (see Section 3) to attack the *B-*

[1] These BNs can be downloaded at `http://bndg.cs.aau.dk/html/bayesian_networks.html`

[2] These patterns, when read left-to-right, describe the size of a clique, the size of a neigboring separator, and the size of a neigboring clique (different from the first) as found in a junction tree. For example, the pattern *B-S-S* describes a big clique, a small separator, and a small clique.

[3] In fact, both the "long" and "short" index mapping tables have for presentation purposes been made short–in a realistic junction tree a "long" table can have more than 10,000 entries.

Figure 3: Example of data structure for two GPU cliques and their separator. Arithmetic parallelism (the reverse pyramid at the bottom) is integrated with element-wise parallelism (mapping tables at the top). The arithmetic parallelism achieves parallel summation of 2^d terms in d cycles.



S-S and B-S-B patterns.

3 PARALLEL MESSAGE PASSING IN JUNCTION TREES

In order to handle the small-separator problem, we discuss another dimension of parallelism in addition to *element-wise parallelism*, namely *arithmetic parallelism*. Arithmetic parallelim explores the parallel opportunity in the sum of (1) and in the multiplication of (2). By considering also arithmetic parallelism, we can better match the junction tree and the many-core GPU platform by optimizing the computing resources allocated to the two dimensions of parallelism.

Mathematically, this optimization can be modeled as a computation time minimization problem:

$$\begin{aligned} &\min_{p_e,p_a} \ T(p_e, p_a, \mathcal{C}_i, \mathcal{C}_j, \Phi), \\ &\text{subject to} : p_e + p_a \leq p_{tot} \end{aligned} \tag{3}$$

where $T(\cdot)$ is the time consumed for a message passing from clique $\mathcal{C}_i$ to clique $\mathcal{C}_j$; p_e and p_a are the number of parallel threads allocated to the element-wise and arithmetic dimensions respectively; p_{tot} is the total number of parallel threads available in the GPU; and Φ is a collection of GPU-related parameters, such as the cache size, etc. Equation (3) is a formulation of optimizing algorithm performance on a parallel computing platform. Unfortunately, traditional optimization techniques can typically not be applied to this optimization problem. This is because the analytical form of $T(\cdot)$ is usually not available, due to the complexity of the hardware platform. So in our work we choose p_e and p_a empirically for our implementation.

In the rest of this section, we will describe our algorithm design, seeking to explore both element-wise and arithmetic parallelism.

3.1 Arithmetic Parallelism

Arithmetic parallelism needs to be explored in different ways for reduction and scattering, and also integrated with element-wise parallelism, as we will discuss now.

For reduction, given a certain fixed element j, Equation (1) is essentially a summation over all the clique potential table $\phi_{\mathcal{X}_i}$ elements indicated by the corresponding mapping table $\mu_{\mathcal{X}_i,\phi_{\mathcal{S}_{ik}}(j)}$. The number of sums is $|\mu_{\mathcal{X}_i,\phi_{\mathcal{S}_{ik}}(j)}|$. We compute the summation in parallel by using the approach illustrated in Figure 3. This improves the handling of long index mapping tables induced, for example, by the B-S-B and B-S-S patterns. The summation is done in several iterations. In each iteration, the numbers are divided into two groups and the corresponding two numbers in each

group are added in parallel. At the end of this recursive process, the sum is obtained, as shown in Algorithm 1. The input parameter d is discussed in Section 3.2; the remaining inputs are an array of floats.

Algorithm 1 ParAdd$(d, op(0), \ldots, op(2^d - 1))$

Input: $d, op(0), \ldots, op(2^d - 1)$.
sum = 0
for $i = 1$ *to* d **do**
  for $j = 0$ *to* $2^{d-i} - 1$ **in parallel do**
    $op(j) = op(j) + op(j + 2^{d-i})$
  end for
end for
return $op(0)$

For scattering, Equation (2) updates the elements of $\phi_{\mathcal{X}_k}$ independently despite that $\phi_{\mathcal{S}_{ik}}$ and $\phi^*_{\mathcal{S}_{ik}}$ are re-used to update different elements. Therefore, we can compute each multiplication in (2) with a single thread. The parallel multiplication algorithm is given in Algorithm 2. The input parameter p is discussed in Section 3.2; the remaining inputs are an array of floats.

Algorithm 2 ParMul$(p, op1, op2(0), \ldots, op2(p-1))$

Input: $p, op1, op2(0), \ldots, op2(p-1)$.
sum = 0
for $j = 0$ *to* $p - 1$ **in parallel do**
  $op2(j) = op1 * op2(j)$
end for

3.2 Parallel Belief Propagation

Combining element-wise and arithmetic parallelism, we design the reduction and scattering operations as shown in Algorithm 3 and Algorithm 4. Both of these algorithms take advantage of arithmetic parallelism. In Algorithm 3, the parameter d is the number of cycles used to compute the reduction (summation), while in Algorithm 4, p is the number of threads operating in parallel. Both p and d are parameters that determine the degree of arithmetic parallelism. They can be viewed as a special form of p_a in (3). Based on Algorithm 3 and Algorithm 4, junction tree message passing can be written as shown in Algorithm 5.

Belief propagation can be done using both breadth-first and depth-first traversal over a junction tree. We use the Hugin algorithm [5], which adopts depth-first belief propagation. Given a junction tree J with root vertex C_{root}, we first initialize the junction tree by multiplying together the Bayesian network potential tables (CPTs). Then, two phase belief propagation is adopted [10]: collect evidence and then distribute evidence [22].

Algorithm 3 Reduction$(d, \phi_{\mathcal{X}_i}, \phi_{\mathcal{S}_{ik}}, \mu_{\mathcal{X}_i,\mathcal{S}_{ik}})$

Input: $d, \phi_{\mathcal{X}_i}, \phi_{\mathcal{S}_{ik}}, \mu_{\mathcal{X}_i,\mathcal{S}_{ik}}$.
for $n = 1$ *to* $|\phi_{\mathcal{S}_{ik}}|$ **in parallel do**
  for $j = 0$ *to* $\lceil |\mu_{\mathcal{X}_i,\mathcal{S}_{ik}(n)}|/2^d \rceil$ **do**
    sum = sum + ParAdd$(d, \phi_{\mathcal{X}_i}(\mu_{\mathcal{X}_i,\mathcal{S}_{ik}(n)}(j * 2^d)), \ldots, \phi_{\mathcal{X}_i}(\mu_{\mathcal{X}_i,\mathcal{S}_{ik}(n)}((j+1) * 2^d - 1)))$
  end for
end for

Algorithm 4 Scattering$(p, \phi_{\mathcal{X}_k}, \phi_{\mathcal{S}_{ik}}, \mu_{\mathcal{X}_k,\mathcal{S}_{ik}})$

Input: $p, \phi_{\mathcal{X}_k}, \phi_{\mathcal{S}_{ik}}, \mu_{\mathcal{X}_k,\mathcal{S}_{ik}}$.
for $n = 1$ *to* $|\phi_{\mathcal{S}_{ik}}|$ **in parallel do**
  for $j = 0$ *to* $\lceil |\phi_{\mathcal{S}_{ik}}|/p \rceil$ **do**
    sum = sum + ParMul$(p, \frac{\phi^*_{\mathcal{S}_{ik}}(n)}{\phi_{\mathcal{S}_{ik}}(n)}, \phi_{\mathcal{X}_k}(\mu_{\mathcal{X}_k,\mathcal{S}_{ik}(n)}(j * p)), \ldots, \phi_{\mathcal{X}_k}(\mu_{\mathcal{X}_k,\mathcal{S}_{ik}(n)}((j+1) * p - 1)))$
  end for
end for

3.3 Max-product Belief Propagation

In this paper, we also apply our parallel techniques to max-product propagation (or in short, max-propagation), which is also referred as the Viterbi algorithm. Max-propagation solves the problem of computing a most probable explanation. For max-propagation, the $\sum$ in (1) is replaced by max [2,15]. In this paper we use sum-product propagation to explain our parallel algorithms; the explanation can generally be changed to discuss max-propagation by replacing add with max.

4 CLIQUE MERGING FOR JUNCTION TREES

The performance of our parallel algorithm is to a large extent determined by the degree of parallelism available in message passing, which intuitively can be measured by the separator size $|\phi_{\mathcal{S}_{ik}}|$, which determines the element-wise parallelism, and the mapping table size $|\mu_{\mathcal{X}_i,\mathcal{S}_{ik}(n)}|$ which upper bounds the arithmetic parallelism. In other words, the larger $|\phi_{\mathcal{S}_{ik}}|$ and $|\mu_{\mathcal{X}_i,\mathcal{S}_{ik}(n)}|$ are, the greater is the parallelism opportunity. Therefore, message passing between small cliques (the *S-S-S* pattern), where $|\phi_{\mathcal{S}_{ik}}|$ and $|\mu_{\mathcal{X}_i,\mathcal{S}_{ik}(n)}|$ are small, is not expected to have good performance. There is not enough parallelism to make full use of

Algorithm 5 PassMessage$(p, d, \phi_{\mathcal{X}_i}, \phi_{\mathcal{X}_k}, \phi_{\mathcal{S}_{ik}}, \mu_{\mathcal{X}_i,\mathcal{S}_{ik}})$

Input: $p, d, \phi_{\mathcal{X}_i}, \phi_{\mathcal{X}_k}, \phi_{\mathcal{S}_{ik}}, \mu_{\mathcal{X}_i,\mathcal{S}_{ik}}$.
Reduction$(d, \phi_{\mathcal{X}_i}, \phi_{\mathcal{S}_{ik}}, \mu_{\mathcal{X}_i,\mathcal{S}_{ik}})$
Scattering$(p, \phi_{\mathcal{X}_k}, \phi_{\mathcal{S}_{ik}}, \mu_{\mathcal{X}_k,\mathcal{S}_{ik}})$

a GPU's computing resources.

In order to better use the GPU computing power, we propose to remove small separators (that follow the S-S-S pattern) by selectively merging neighboring cliques. This increases the length of mapping tables, however the arithmetic parallelism techniques introduced in Section 3 can handle this. Clique merging can be done offline according to this theorem [8].

Theorem 4.1 *Two neighboring cliques C_i and C_j in a junction tree J with the separator S_{ij} can be merged together into an equivalent new clique C_{ij} with the potential function*

$$\phi(x_{C_{ij}}) = \frac{\phi(x_{C_i})\phi(x_{C_j})}{\phi(x_{S_{ij}})}, \tag{4}$$

while keeping all the other part of the junction tree unchanged.

The result of merging cliques is three-fold: (i) it produces larger clique nodes and thus longer mapping tables; (ii) it eliminates small separators; and (iii) it reduces the number of cliques. Larger clique nodes will result in more computation and therefore longer processing time for each single thread, but getting rid of small separators will improve utilization of the GPU and reduce computation time. We have to manage these two conflicting objectives to improve the overall performance of our parallel junction tree algorithm.

Our algorithm for clique merging is shown in Algorithm 6. It uses two heuristic thresholds, the separator size threshold τ_s and the index mapping table size threshold τ_μ, to control the above-mentioned two effects. We only merge two neighboring cliques C_i and C_j into a new clique C_{ij} when $|\phi_{\mathcal{S}_{ij}}| < \tau_s$ and $|\mu_{\mathcal{X}_j,\phi_S}| < \tau_\mu$.

Algorithm 6 MergeCliques(J, τ_s, τ_μ)

```
merge_flag = 1
while merge_flag do
  merge_flag = 0
  for each adjacent clique pair (C_i, C_j) in J do
    if φ_S < τ_s and |μ_{X_j,φ_S}| < τ_μ then
      Merge (J, C_i, C_j)
      merge_flag = 1
    end if
  end for
end while
```

Given an S-S-S-S-S pattern, Algorithm 6 may merge two S cliques and produce an S-B-S pattern. Here, B is the merged clique. Note that the B-S sub-pattern creates a long index mapping table, which is exactly what arithmetic parallelism handles. There is in other words potential synergy between clique merging and arithmetic parallelism, as is further explored in Section 6.

5 ANALYSIS AND DISCUSSION

In this section, we analyze the theoretical speedup for our two-dimensional parallel junction tree inference algorithm under the idealized assumption that there is unlimited parallel threads available from the many-core computing platform.

The degree of parallelism opportunity is jointly determined by the size of the separators' potential table, $|\phi_{\mathcal{S}}|$, and the size of the index mapping table $|\mu_{\mathcal{X},\phi_S}|$. Consider a message passed from $\mathcal{C}_i$ to $\mathcal{C}_k$. Since we employ separator table element-wise parallelism in our algorithm, we only need to focus on the computation related to one particular separator table element. With the assumption of unlimited parallel threads, we can choose $d = \lceil \log |\mu_{\mathcal{X}_i,\phi_S}| \rceil$. The time complexity for the reduction is then $\lceil \log |\mu_{\mathcal{X}_i,\phi_S}| \rceil$, due to our use of summation parallelism.[4] Note since $|\mu_{\mathcal{X}_i,\phi_S}| = \frac{|\phi_{\mathcal{X}_i}|}{|\phi_S|}$, the time complexity can be written as $\lceil \log |\phi_{\mathcal{X}_i}| - \log |\phi_{\mathcal{S}}| \rceil$. For the scattering phase, we choose $p = |\mu_{\mathcal{X}_k,\phi_S}|$ and the time complexity is given by $|\mu_{\mathcal{X}_k,\phi_S}|/p + 1 = 2$ due to the multiplication parallelism. Thus the overall time complexity of the two-dimensional belief propagation algorithm is:

$$\lceil \log |\phi_{\mathcal{X}_i}| - \log |\phi_{\mathcal{S}}| \rceil + 2, \tag{5}$$

which is the theoretical optimal time complexity under the assumption of an infinite number of threads. Nevertheless, this value is hard to achieve in practice since the value of d and p are subject to the concurrency limit of the computing platform. For example, in the above-mentioned BN *Pigs*, some message passing requires $p = 1120$ while the GTX460 GPU supports at most 1024 threads per thread block.

Belief propagation is a sequence of messages passed in a certain order [10], for both CPU and GPU [22]. Let $Ne(\mathcal{C})$ denote the neighbors of $\mathcal{C}$ in the junction tree. The time complexity for belief propagation is

$$\sum_i \sum_{k \in Ne(\mathcal{C}_i)} \left(\lceil \log |\phi_{\mathcal{X}_i}| - \log |\phi_{\mathcal{S}}| \rceil + 2 \right),$$

Kernel invocation overhead, incurred each time Algorithm 5 is invoked, turns out to be an important performance factor. If we model the invocation overhead for each kernel call to be a constant τ, then the time

[4] We assume, for simplicity, sum-propagation. The analysis for max-propagation is similar.

complexity becomes

$$\sum_i d_i\tau + \sum_i \sum_{k\in Ne(\mathcal{C}_i)} (\lceil \log|\phi_{\mathcal{X}_i}| - \log|\phi_{\mathcal{S}}| \rceil + 2)\,,$$

where d_i is the degree of a node $\mathcal{C}_i$. In a tree structure, $\sum d_i = 2(n-1)$. Thus the GPU time complexity is

$$2(n-1)\tau + \sum_i \sum_{k\in Ne(\mathcal{C}_i)} (\lceil \log|\phi_{\mathcal{X}_i}| - \log|\phi_{\mathcal{S}}| \rceil + 2)\,.$$

From this equation, we can see that the junction tree topology impacts GPU performance in at least two ways: the total invocation overhead is proportional to the number of nodes in the junction tree, while the separator and clique table sizes determine the degree of parallelism.

The overall speedup of our parallel belief propagation approach is determined by the equation

$$Speedup = \frac{\sum_i \sum_{k\in Ne(\mathcal{C}_i)} (|\phi_{\mathcal{X}_i}| + |\phi_{\mathcal{X}_k}|)}{2(n-1)\tau + \sum_i \sum_{k\in Ne(\mathcal{C}_i)} \left(\left\lceil \log \frac{|\phi_{\mathcal{X}_i}|}{|\phi_{\mathcal{S}}|} \right\rceil + 2 \right)}.$$

Clearly, the speedup depends on the distribution of the sizes of the separators' and cliques' potential tables. That is the reason we propose the clique merging technique. Using clique merging, we change the number of nodes in the junction tree and distribution of the size of the separators' and cliques' potential table as well, adapting the junction tree for the specific parallel computing platform.

From the equations above, we can estimate the overall belief propagation speedup. However, taking into account that the CPU/GPU platform incurs invocation overhead and the long memory latency when loading data from slow device memory to fast shared memory, the theoretical speedup is hard to achieve in practice. We take an experimental approach to study how the structure of the junction trees affects the performance of our parallel technique on the CPU/GPU setting in Section 6.

6 EXPERIMENTAL RESULTS

In experiments, we study Bayesian networks compiled into junction trees. We not only want to compare the two-dimensional parallel junction tree algorithm to the sequential algorithm, but also study how effective the arithmetic parallelism and clique merging methods are. Consequently, we experiment with different combinations of element-wise parallelism (EP), arithmetic parallelism (AP), and clique merging (CM).

6.1 Computing Platform

We use the NVIDIA GeForce GTX460 as the platform for our implementation. This GPU consists of seven multiprocessors, and each multiprocessor consists of 48 cores and 48K on-chip shared memory per thread block. The peak thread level parallelism achieves 907GFlop/s. In addition to the fast shared memory, a much larger but slower off-chip global memory (785 MB) that is shared by all multiprocessors is provided. The bandwidth between the global and shared memories is about 90 Gbps. In the junction tree computations we are using single precision for the GPU and the thread block size is set to 256.

6.2 Methods and Data

For the purpose of comparison, we use the same set of BNs as used previously [22] (see `http://bndg.cs.aau.dk/html/bayesian_networks.html`). They are from different domains, with varying structures and state spaces. These differences lead to very different junction trees, see Table 1, resulting in varying opportunities for element-wise and arithmetic parallelism. Thus, we use clique merging to carefully control our two dimensions of parallelism to optimize performance. The Bayesian networks are compiled into junction trees and merged offline and then junction tree propagation is performed.

6.3 GPU Optimization: Arithmetic Parallelism

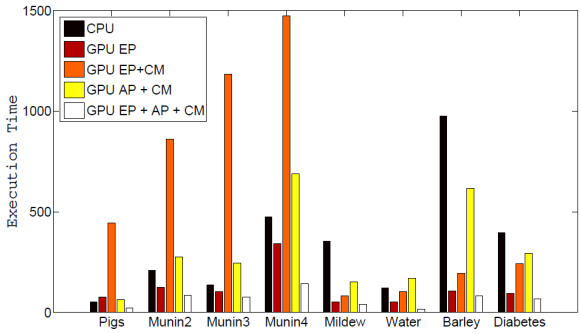
Arithmetic parallelism gives us more freedom to match the parallelism in message passing and the concurrency provided by a GPU: when there is not enough potential table element-wise parallelism available, we can increase the degree of arithmetic parallelism. The number of threads assigned to arithmetic parallelism affects the performance significantly. The parameter p in parallel scattering and the parameter 2^d in the parallel reduction should be chosen carefully (see Algorithm 1 and 2). Since the GPU can provide only limited concurrency, we need to balance the arithmetic parallelism and the element-wise parallelism for each message passing to get the best performance.

Consider message passing between big cliques, for example according to the *B-S-B* pattern. Intuitively, the values of the arithmetic parallelism parameters p and d should be set higher than for the message passing between smaller cliques. Thus, based on extensive experimentation, we currently employ a simple heuristic parameter selection scheme for the scattering parameter p

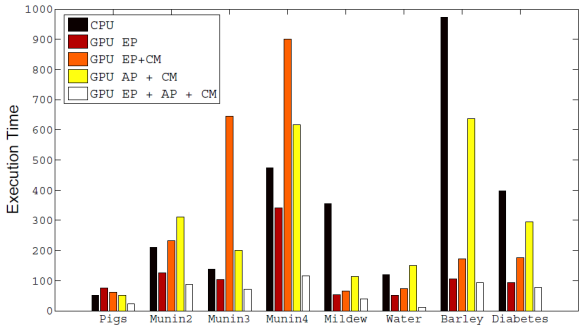
$$p = \begin{cases} 4 & \text{if} |\mu_{\mathcal{X}_i,\mathcal{S}_{ik}(n)}| \leq 100 \\ 128 & \text{if} |\mu_{\mathcal{X}_i,\mathcal{S}_{ik}(n)}| > 100 \end{cases} \quad (6)$$

Dataset	Pigs	Munin2	Munin3	Munin4	Mildew	Water	Barley	Diabetes
# of original JT nodes	368	860	904	872	28	20	36	337
# of JT nodes after merge	162	553	653	564	22	18	35	334
Avg. CPT size before merge	1,972	5,653	3,443	16,444	341,651	173,297	512,044	32,443
Avg. CPT size after merge	5,393	10,191	7,374	26,720	447,268	192,870	527,902	33,445
Avg. SPT size before merge	339	713	533	2,099	9,273	26,065	39,318	1,845
Avg. SPT size after merge	757	1,104	865	3,214	11,883	29,129	40,475	1,860
GPU time (sum-prop) [ms]	22.61	86.40	74.99	141.08	41.31	16.33	81.82	68.26
GPU time (max-prop) [ms]	22.8	86.8	72.6	114.9	38.6	12.1	94.3	78.3
CPU time (sum-prop) [ms]	51	210	137	473	355	120	974	397
CPU time (max-prop) [ms]	59	258	119	505	259	133	894	415
Speedup (sum-prop)	**2.26x**	**2.43x**	**1.82x**	**3.35x**	**8.59x**	**7.35x**	**11.94x**	**5.81x**
Speedup (max-prop)	**2.58x**	**2.97x**	**1.64x**	**4.39x**	**6.71x**	**10.99x**	**9.48x**	**5.30x**

Table 1: Junction tree (JT) statistics and belief propagation (BP) performance for various junction trees, with speedup for our GPU approach (GPU EP + AP + CM) compared to CPU-only in the two bottom rows. The row "CPU time (sum-prop)" gives previous results [22].



(a) Junction tree sum-propagation



(b) Junction tree max-propagation

Figure 4: Comparison of combinations of junction tree optimization techniques CM, AP, and EP for (a) sum- and (b) max-propagation. Best performance is achieved for GPU EP + AP + CM.

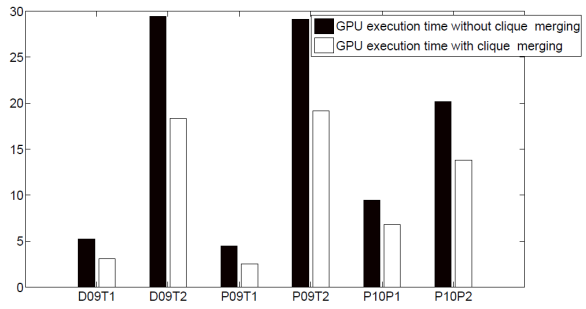


Figure 5: GPU execution times with CM (GPU EP + AP + CM) and without CM (GPU EP + AP) for junction trees compiled from sparse BNs representing electrical power systems.

and the reduction parameter d

$$d = \begin{cases} 2 & \text{if} |\mu_{\mathcal{X}_i, \mathcal{S}_{ik}(n)}| \leq 100 \\ 7 & \text{if} |\mu_{\mathcal{X}_i, \mathcal{S}_{ik}(n)}| > 100 \end{cases} \quad (7)$$

We compare the execution time when using element-wise parallelism alone and the case when element-wise parallelism is used in combination with arithmetic parallelism. Results, for both sum-propagation and max-propagation, are shown in Figure 4(a) and Figure 4(b). In all cases, the GPU EP + AP + CM outperforms all the other approaches.[5]

6.4 GPU Optimization: Clique Merging

Clique merging is based on the observation that many junction trees mostly consist of small cliques. This lack of parallelism opportunity will hinder the efficient

[5]The GPU EP + CM combination is included for completeness, but as expected it often performs very poorly. The reason for this is that CM, by merging cliques, creates larger mapping tables that EP is not equipped to handle.

use of the available computing resources, since a single message passing will not be able to occupy the GPU. Merging neighboring small cliques, found in the *S*-*S*-*S* pattern, can help to increase the average size of separators and cliques. Clique merging also reduces the total number of nodes in the junction tree, which in turn reduces the invocation overhead.

We use two parameters to determine which cliques should be merged–one is τ_s, the threshold for separators' potential table size and the other is τ_μ, the threshold for the size of the index mapping table. These parameters are set manually in this paper, however in a companion paper [21] this parameter optimization process is automated by means of machine learning.

In the experiments, we used both arithmetic parallelism and element-wise parallelism. This experiment presents how much extra speedup can be obtained by using clique merging and arithmetic parallelism. The experimental results can be found in Table 1, in the rows showing the GPU execution times for both sum-propagation and max-propagation. In junction trees *Pigs*, *Munin2*, *Munin3* and *Munin4*, a considerable fraction of cliques (and consequently, separators) are small, in other words the *S*-*S*-*S* pattern is common. By merging cliques, we can significantly increase the average separators' and cliques' potential size and thus provide more parallelism.

Comparing GPU EP + AP with GPU EP + AP + CM, the speedup for these junction trees ranged from 10% to 36% when clique merging is used. However, in junction trees *Mildew*, *Water*, *Barley* and *Diabetes*, clique merging does not help much since they mainly consist of large cliques to start with.

Another set of experiments were performed with junction trees that represent an electrical power system ADAPT [16–18]. These junction trees contain many small cliques, due to their underlying BNs being relatively sparsely connected.[6] The experimental results are shown in Figure 5. Using clique merging, the GPU execution times are shortened by 30%-50% for these BNs compared to not using clique merging.

6.5 Performance Comparison: CPU

As a baseline, we implemented a sequential program on an Intel Core 2 Quad CPU with 8MB cache and a 2.5GHz clock. The execution time of the program is comparable to that of GeNie/SMILE, a widely used C++ software package for BN inference.[7] We do not directly use GeNie/SMILE as the baseline here, because we do not know the implementation details of GeNie/SMILE.

In Table 1, the bottom six rows give the execution time comparison for our CPU/GPU hybrid versus a traditional CPU implementation. The CPU/GPU hybrid uses arithmetic parallelism, element-wise parallelism and clique merging. The obtained speedup for sum-propagation ranges from 1.82x to 11.94x, with an arithmetic average of 5.44x and a geometric average of 4.42.

The speedup for max-propagation is similar, but different from sum-propagation in non-trivial ways. The performance is an overall effect of many factors such as parallelism, memory latency, kernel invocation overhead, etc. Those factors, in turn, are closely correlated with the underlying structures of the junction trees. The speedup for max-propagation ranges from 1.64x to 10.99x, with an arithmetic average of 5.51x and a geometric average of 4.61x.

6.6 Performance Comparison: Previous GPU

We now compare the GPU EP + AP + CM technique introduced in this paper with our previous GPU EP approach [22]. From results in Table 1, compared with the GPU EP approach [22], the arithmetic average cross platform speedup increases from 3.38x (or 338%) to 5.44x (or 544%) for sum-propagation. For max-propagation[8] the speedup increases from 3.22x (or 322%) to 5.51x (or 551%).

7 CONCLUSION AND FUTURE WORK

In this paper, we identified small separators as bottlenecks for parallel computing in junction trees and developed a novel two-dimensional parallel approach for belief propagation over junction trees. We enhanced these two dimensions of parallelism by careful clique merging in order to make better use of the parallel computing resources of a given platform.

In experiments with a CUDA implementation on an NVIDIA GeForce GTX460 GPU, we explored how the performance of our approach varies with different junction trees from applications and how clique merging can improve the performance for junction trees that contains many small cliques. For sum-propagation, the average speedup is 5.44x and the maximum speedup is 11.94x. The average speedup for max-propagation is 5.51x while the maximum speedup is 10.99x.

In the future, we would like to see research on parameter optimization for both clique merging and message

[6] `http://works.bepress.com/ole_mengshoel/`

[7] `http://genie.sis.pitt.edu/`

[8] We implemented max-propagation based on the approach developed previously [22].

passing. It would be useful to automatically change the merging parameters for different junction trees based on the size distribution of the cliques and separators. In addition, we also want to automatically change the kernel running parameters for each single message passing according to the size of a message. In fact, we have already made progress along these lines, taking a machine learning approach [21].

Acknowledgments

This material is based, in part, upon work supported by NSF awards CCF0937044 and ECCS0931978.

References

[1] R. Bekkerman, M. Bilenko, and J. Langford, editors. *Scaling up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, 2011.

[2] A. P. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2:25–36, 1992.

[3] J. Gonzalez, Y. Low, C. Guestrin, and D. O'Hallaron. Distributed parallel inference on large factor graphs. In *Proc. of the 25th Conference in Uncertainty in Artificial Intelligence (UAI-09)*, 2009.

[4] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, (3):225–263, 1994.

[5] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic network by local computations. *Computational Statistics*, pages 269–282, 1990.

[6] H. Jeon, Y. Xia, and V. K. Prasanna. Parallel exact inference on a CPU-GPGPU heterogenous system. In *Proc. of the 39th International Conference on Parallel Processing*, pages 61–70, 2010.

[7] K. Kask, R. Dechter, and A. Gelfand. BEEM: bucket elimination with external memory. In *Proc. of the 26th Annual Conference on Uncertainty in Artificial Intelligence (UAI-10)*, pages 268–276, 2010.

[8] D. Koller and N. Friedman, editors. *Probablistic graphic model: principle and techniques*. The MIT Press, 2009.

[9] A. V. Kozlov and J. P. Singh. A parallel Lauritzen-Spiegelhalter algorithm for probabilistic inference. In *Proc. of the 1994 ACM/IEEE conference on Supercomputing*, pages 320–329, 1994.

[10] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50(2):157–224, 1988.

[11] M. D. Linderman, R. Bruggner, V. Athalye, T. H. Meng, N. B. Asadi, and G. P. Nolan. High-throughput Bayesian network learning using heterogeneous multicore computers. In *Proc. of the 24th ACM International Conference on Supercomputing*, pages 95–104, 2010.

[12] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. Hellerstein. GraphLab: A new framework for parallel machine learning. In *Proc. of the 26th Annual Conference on Uncertainty in Artificial Intelligence (UAI-10)*, pages 340–349, 2010.

[13] O. J. Mengshoel. Understanding the scalability of Bayesian network inference using clique tree growth curves. *Artificial Intelligence*, 174:984–1006, 2010.

[14] V. K. Namasivayam and V. K. Prasanna. Scalable parallel implementation of exact inference in Bayesian networks. In *Proc. of the 12th International Conference on Parallel and Distributed System*, pages 143–150, 2006.

[15] D. Nilsson. An efficient algorithm for finding the m most probable configurations in probablistic expert system. *Statistics and Computing*, pages 159–173, 1998.

[16] B. W. Ricks and O. J. Mengshoel. The diagnostic challenge competition: Probabilistic techniques for fault diagnosis in electrical power systems. In *Proc. of the 20th International Workshop on Principles of Diagnosis (DX-09)*, pages 415–422, Stockholm, Sweden, 2009.

[17] B. W. Ricks and O. J. Mengshoel. Methods for probabilistic fault diagnosis: An electrical power system case study. In *Proc. of Annual Conference of the PHM Society, 2009 (PHM-09)*, San Diego, CA, 2009.

[18] B. W. Ricks and O. J. Mengshoel. Diagnosing intermittent and persistent faults using static bayesian networks. In *Proc. of the 21st International Workshop on Principles of Diagnosis (DX-10)*, Portland, OR, 2010.

[19] M. Silberstein, A. Schuster, D. Geiger, A. Patney, and J. D. Owens. Efficient computation of sum-products on GPUs through software-managed cache. In *Proc. of the 22nd ACM International Conference on Supercomputing*, pages 309–318, 2008.

[20] Y. Xia and V. K. Prasanna. Node level primitives for parallel exact inference. In *Proc. of the 19th International Symposium on Computer Architechture and High Performance Computing*, pages 221–228, 2007.

[21] L. Zheng and O. J. Mengshoel. Optimizing parallel belief propagation in junction trees using regression. In *Proc. of 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD-13)*, Chicago, IL, August 2013.

[22] L. Zheng, O. J. Mengshoel, and J. Chong. Belief propagation by message passing in junction trees: Computing each message faster using GPU parallelization. In *Proc. of the 27th Conference in Uncertainty in Artificial Intelligence (UAI-11)*, pages 822–830, Barcelona, Spain, 2011.