图模型推理的层次消息传递算法

孙怿　欧智坚　孙甲松
清华大学电子工程系　北京　100084

摘 要：本文提出了用于图模型精确推理的层次消息传递（Hierarchical Message Passing，HMP）算法以及包含树（Containing Tree）算法，以解决传统连接树算法在存在约束包含和约束消除情况下无法充分利用图模型中的结构信息的问题。HMP 算法采用递归结构，逐级挖掘图模型具有的条件独立性以减小推理计算量；包含树算法利用势函数定义域之间的包含关系，有效的降低推理所需的乘法次数。理论分析和实验结果均表明，HMP 算法和包含树算法能够显著地降低在存在约束包含和约束消除情况下的推理计算量。

关键字: 图模型推理、层次消息传递算法、包含树算法

Hierarchical Message Passing Algorithm for Graphical Models

SUN Yi, OU Zhijian and SUN Jiasong
Department of Electronic Engineering, Tsinghua University
100084, Beijing, China

Abstract: *In this paper, we propose Hierarchical Message Passing (HMP) algorithm for efficient inference on Graphical Models, which exploits the variable-level structural information contained in the graph thoroughly in a recursive way. Specifically, each step of message passing in a reasoning problem (i.e. marginalizing a product function) is treated as a smaller reasoning problem, and we introduce containing tree as an efficient structure for marginalizing over a single cluster (the lowest-level problem). We demonstrate that HMP can be order-of-magnitude better than typical algorithms based on Shenoy-Shafer architecture, especially when operating on cluster-trees constructed under constraints. Experimental results with various random graphs show that HMP achieves significant performance improvements over basic Shenoy-Shafer algorithm and lazy propagation.*

Keywords: *graphical model inference, hierarchical message passing, containing tree*

1　引言

连接树算法是图模型推理中一类重要的算法[1]。这类算法首先在图模型上构造连接树[2]（Join Tree）或与之等价的树结构（Bucket Tree[3]、Junction Tree[4]等），而后在连接树上进行消息传递（Message Passing），进而求解连接树各节点上变量集合的边缘分布。

连接树算法的效率依赖于输入图模型自身的结构和连接树的构造方法。当输入的图模型规模较小时，现有的连接树构造算法可以构造出宽度[9]接近图模型树宽的连接树，使得连接树算法具有良好的性能。然而，在一些情况下，无法构造出宽度较小的连接树。首先，一大类图模型上的推理任务需要求解给定变量集合上的边缘分布。在这种情况下，连接树必须根据特定的推理任务构造，将给定变量集合置于同一个连接树节点中以求得它们的联合概率密度（我们称之为约束包含，Constrained Inclusion）。其次，对于一些特定的图模型，必须利用约束消除（Constrained Elimination）以满足特定的复杂度约束。特别的，在动态贝叶斯网络推理中，每帧中的界面变量集（Interface Set）的消除必须在该帧中其它变量之前被消除[5;6]。在这些情况下，用于构造连接树的图模型不再是输入的原始图模型，而是一个在原有图模型基础上加入大量虚边（Virtual Edge）的图模型。这些虚边的引入保证了构造出的连接树能够满足特定的推理任务要求。在引入虚边后的图模型中，一部分条件独立性（Conditional Independence）被引入的虚边破坏，因此引入虚边后的图模型的树宽往往比原有的图模型高得多，相应的，在引入虚边后的图模型上构造的连接树的宽度也大大高于原始图模型的树宽，严重的增加了推理的复杂度。

针对这一问题，我们提出**层次消息传递**（Hierarchical Message Passing, HMP）算法。HMP 算法的主要思想在于：在消息计算的过程中，发现和利用由于引入虚边而被破坏的条件独立

*基金项目：国家自然科学基金（60402029），863（2006AA01Z149）

性，以降低在约束包含和约束消除情况下的推理复杂度。通过递归的方法，HMP 算法能够逐级挖掘由于引入虚边而被破坏的条件独立性。在每次递归调用中，HMP 算法利用当前发现的条件独立性将输入的推理问题分解为若干个规模更小的推理问题。而在递归调用的最后一步，当推理问题无法利用条件独立性进行进一步分解时，则通过构造包含树（Containing Tree），利用变量集之间的包含关系进一步降低运算量。

2 层次消息传递算法

2.1 算法描述

在本文中，我们将离散随机变量简称为变量（Variable），将定义在一组变量上的非负实值函数称为势函数（Potential）。为了简化叙述，我们假定有 N 个可能取值的变量的值域为{0, 1,..., N－1}。在叙述中，我们使用粗体大写字母（非斜体）表示势函数和变量的集合（例如 **V**，**P**），用大写斜体字母表示单个的变量或势函数（例如 V，P）。变量集合的取值用小写粗体（非斜体）字母表示（如 **v**，**e**），单个变量的取值用小写斜体字母表示（如 v，e）。给定势函数 P 和一组变量取值 **e**，$P|\mathbf{e}$ 表示将 P 中出现的在 **e** 中已赋值的变量取定后得到的新的势函数，而 $\mathbf{P}|\mathbf{e}$ 表示将 **P** 中每个势函数 P 替换为 $P|\mathbf{v}$ 得到的势函数集合。

一个图模型$\mathbb{G}$为一个二元组$\langle \mathbf{V}, \mathbf{P}\rangle$，其中 **V** 为图模型的变量集合，**P** 为图模型的势函数集合。为了简化叙述，我们用$|\cdot|$表示任意集合中元素的个数（集合的势）。此外，我们用 $\mathbf{dom}(P)$, $\mathbf{dom}(\mathbf{P})$表示势函数 P 以及势函数集合 **P** 的定义域。

一个**推理问题**（Reasoning Problem）$\mathcal{P}$是一个二元组：$\langle \mathbf{\Phi}, \mathbf{Z}\rangle$，其中$\mathbf{\Phi}$是一个势函数集合，**Z** 是一个变量集合。设 $\mathbf{X}=\mathbf{dom}(\mathbf{\Phi})$为$\mathbf{\Phi}$的定义域，一个势函数集合$\mathbf{\Psi}$被称为推理问题$\mathcal{P}$的解，如果$\mathbf{\Psi}$定义在 $\mathbf{Z}\cap\mathbf{X}$ 上，且式(1)恒成立：

$$\prod_{\psi\in\mathbf{\Psi}}\psi=\sum_{\mathbf{X}\setminus\mathbf{Z}}\prod_{\varphi\in\mathbf{\Phi}}\varphi \tag{1}$$

我们分别称$\mathbf{\Phi}$和$\mathbf{\Psi}$为**输入**和**输出**势函数集，称$\mathbf{X}$、$\mathbf{X}\setminus\mathbf{Z}$、$\mathbf{X}\cap\mathbf{Z}$为**全变量集**、**消除变量集**、**目标变量集**。定义$\mathbb{G}\langle\mathbf{X},\mathbf{\Phi}\rangle$为$\mathbf{\Phi}$对应的图模型，记为$\mathbb{G}_{\mathbf{\Phi}}$。$\mathbb{G}_{\mathbf{\Phi}}$上的连接树$\mathbb{T}$被称为与$\mathcal{P}$**相容**的，如果$\mathbb{T}$的根节点包含了目标变量集$\mathbf{X}\cap\mathbf{Z}$。如果$\mathbb{T}$与$\mathcal{P}$相容，则可以通过 1）在$\mathbb{T}$上进行消息收集（Message Gathering），而后 2）从根节点中边缘化掉（Marginalize）不属于目标变量集的变量，求得目标变量集上的边缘分布。

需要指出的是，在这里定义的图模型的推理问题涵盖了一大类图模型的应用。给定贝叶斯网络$\mathbb{G}\langle\mathbf{V},\mathbf{P}\rangle$和**证据集合**（Evidence）**e**，$\mathcal{P}:\langle\mathbf{P}|\mathbf{e},\emptyset\rangle$计算了证据集合 **e** 对应的概率；如果目标变量集不为空，则$\mathcal{P}:\langle\mathbf{P}|\mathbf{e},\mathbf{Z}\rangle$计算了在观测到 **e** 的情况下，目标变量集 **Z** 的边缘分布。此外，在消息传递过程中，每个消息的产生过程实质上也是一个推理问题，其中的输入势函数集合为连接树节点上的势函数与其接收到的消息的并集，而输出势函数集合则为产生的消息。

HMP算法过程如算法 1 所示。给定推理问题$\mathcal{P}$: $\langle\mathbf{\Phi},\mathbf{Z}\rangle$，我们考虑以下两种情况。首先，考虑$\mathbb{G}_{\mathbf{\Phi}}$含有多个连通分量（Connected Component）的情况（行 2）。在这种情况下，$\mathbf{\Phi}$可以被分解为N个不相交的势函数集合$\mathbf{\Phi}_i$，$i=1,\ldots,N$，且任取其中两个集合，如$\mathbf{\Phi}_i$和$\mathbf{\Phi}_j$，有$\mathbf{dom}(\mathbf{\Phi}_i)\cap\mathbf{dom}(\mathbf{\Phi}_j)=\emptyset$。由此，给定的推理问题$\mathcal{P}$可以分解为若干个子问题$\mathcal{P}_i$: $\langle\mathbf{\Phi}_i, \mathbf{Z}\cap\mathbf{dom}(\mathbf{\Phi}_i)\rangle$，如式(2)。

$$\prod_{\psi\in\mathbf{\Psi}}\psi=\prod_{i=1}^{N}\left(\sum_{\mathbf{dom}(\mathbf{\Phi}_i)\setminus\mathbf{Z}}\prod_{\varphi_i\in\mathbf{\Phi}_i}\varphi_i\right) \tag{2}$$

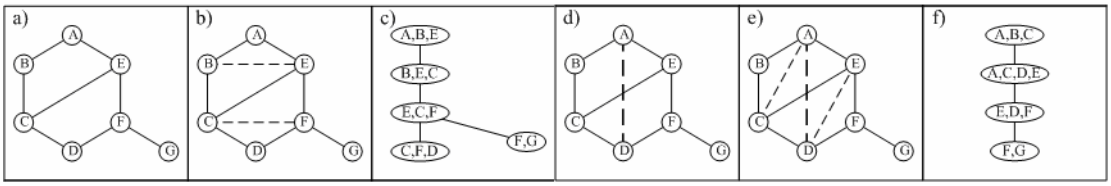

图 1 约束包含的例子。图中，原始的输入图模型为 a)，三角化后得到的图模型为 b)，其中虚线为三角化加入的边。如果需要计算变量 A，D 的边缘分布，则需要在原有的图模型中加入虚边(A,D)，这样得到的图模型三角化后的结果如图 e)。两种情况下的连接树分别如 c）和 f)，可以看到，在无约束包含的情况下，连接树的宽度为 3，而加入约束包含后，宽度增加为 4。

注意到，如果对某个子推理问题$\mathcal{P}_i$，有$\mathbf{Z}\cap\mathbf{dom}(\mathbf{\Phi}_i)=\varnothing$，则$\mathcal{P}_i$可以不经过计算直接删去（行 5），因为$\mathcal{P}_i$的解为一常数因子，不影响最终结果。设每个子问题$\mathcal{P}_i$的解为$\mathbf{\Psi}_i$，则原推理问题$\mathcal{P}$的解为$\cup\mathbf{\Psi}_i$，即将每个子问题的解直接取并即为原推理问题的解（行 6）。

其次，考虑$\mathbb{G}_\Phi$只含有一个连通分量的情况（行 8）。在这种情况下，HMP算法首先生成一棵与$\mathcal{P}$相容的连接树$\mathbb{T}$。如果$\mathbb{T}$只含有一个节点，说明没有更多的条件独立性可以加以利用。在这种情况下，必须将输入的势函数相乘，消除目标变量集以外的变量（行 11，Mar_cluster(r, **Z**)），并将结果作为推理问题的解输出。如果$\mathbb{T}$含有多于一个节点，那么说明输入势函数集合中依然有条件独立性可以加以利用。在这种情况下，在$\mathbb{T}$上进行消息收集过程（行 12 到 22）。如前所述，在连接树$\mathbb{T}$上进行的每一次消息传递本身又对应了一个推理问题，这些推理问题同样可以通过 HMP 算法进行解决（行 15，16）。因此，如果$\mathbb{T}$含有多于一个节点，原推理问题可以进一步分解为一系列规模更小的推理问题。

算法 1 HMP 算法

输入：推理问题$\mathcal{P}$: $\langle\mathbf{\Phi},\mathbf{Z}\rangle$
输出：推理问题的解$\mathbf{\Psi}$

```
HMP(P)
1   Ψ←∅
2   N←number of connected component in G_Φ
3   if N > 1:
4       for each sub problem P_i: ⟨Φ_i, Z∩dom(Φ_i)⟩:
5           if Z∩dom≠∅:
6               Ψ←Ψ∪HMP(P_i)
7       return Ψ
8   form a join tree T compatible with P
9   if T has only one node r:
10      return Mar_cluster(r, Z)
11  else:
12      recv←empty dictionary
13      for each message gathering n_a→n_b:
14          P_a←⟨ n_a∪recv(n_a), sep(n_a,n_b) ⟩
15          recv[n_b]←HMP(P_a)
16      r←root of T
17      if dom(r)⊋Z:
18          P_r←⟨ n_r∪recv(n_r), Z⟩
19          return HMP(P_r)
20      else:
21          return recv[r]
```

2.2 复杂度分析

易证HMP算法在有限次递归后结束，其推理的运算量可以如下分析。首先考虑$\mathbb{G}_\Phi$含有多个连通分量的情况。设$C=(|\boldsymbol{\Phi}|-1)\exp(|\mathbf{dom}(\boldsymbol{\Phi})|)$，$C_i=(|\boldsymbol{\Phi}_i|-1)\exp(|\mathbf{dom}(\boldsymbol{\Phi}_i)|)$。利用式(2)计算$\mathcal{P}$需要$\sum_i \exp(|\mathbf{dom}(\boldsymbol{\Phi}_i)|)$次加法和$\sum_i C_i$次乘法。如果不分解$\mathbb{G}_\Phi$，则需要$\exp(|\mathbf{dom}(\boldsymbol{\Phi})|)$次加法和$C$次乘法。考虑到$\boldsymbol{\Phi}_i \subsetneq \boldsymbol{\Phi}$，可知利用(2)计算$\mathcal{P}$比不分解$\mathbb{G}_\Phi$直接计算$\mathcal{P}$有指数量级的改进。

其次，考虑$\mathbb{G}_\Phi$含有单个连通分量并且$\mathbb{T}$含有多于一个节点的情况。根据图模型理论，在生成树$\mathbb{T}$上进行消息收集所需的复杂度为$O(\exp(w))$，其中w为$\mathbb{T}$的宽度。由于$\mathbb{T}$含有多于一个的节点，可知$w<|\mathbf{dom}(\boldsymbol{\Phi})|$[1]。由于直接计算$\mathcal{P}$的复杂度为$O(\exp(|\mathbf{dom}(\boldsymbol{\Phi})|)$，利用生成树$\mathbb{T}$进行计算比直接计算$\mathcal{P}$有指数量级的提高。

最后，如果$\mathbb{G}_\Phi$含有单个连通分量并且$\mathbb{T}$只含有一个节点，那么利用HMP算法计算$\mathcal{P}$和直接计算$\mathcal{P}$具有相同的计算量。

由此，我们可以得到如下结论：采用 HMP 算法计算推理问题$\mathcal{P}$需要的计算量不大于直接计算$\mathcal{P}$所需的计算量。

3 包含树（Containing Tree）算法

在HMP算法中，迭代算法的最后一步是调用Mar_cluster($\boldsymbol{\Phi}$,$\mathbf{Z}$)。在调用Mar_cluster时，$\mathbb{G}_\Phi$仅含有一个连通分量且其连接树只有单个节点。在这种情况下，依然可以进行计算量的约减。首先，我们将$\boldsymbol{\Phi}$划分为$\boldsymbol{\Lambda}$和$\boldsymbol{\Lambda}'$两个部分，满足$\boldsymbol{\Lambda}\cup\boldsymbol{\Lambda}'=\boldsymbol{\Phi}$，且$\mathbf{dom}(\boldsymbol{\Lambda})\subset\mathbf{Z}$，$\mathbf{dom}(\boldsymbol{\Lambda}')\backslash\mathbf{Z}\neq\emptyset$。注意到$\boldsymbol{\Lambda}$不参与求和，因此可以提到求和号之外（[1]中也提到了这种优化方法），而Mar_cluster($\boldsymbol{\Phi}$,$\mathbf{Z}$)可以依据式(3)进行计算。

$$\prod_{\psi\in\boldsymbol{\Psi}}\psi=\prod_{\varphi\in\boldsymbol{\Lambda}}\varphi\sum_{\mathbf{dom}(\boldsymbol{\Lambda}')\backslash\mathbf{Z}}\prod_{\varphi'\in\boldsymbol{\Lambda}'}\varphi' \tag{3}$$

根据$\boldsymbol{\Lambda}$的定义，我们可以直接将$\boldsymbol{\Lambda}$加入$\boldsymbol{\Psi}$中，因此只需要计算求和号右侧的乘积。进行这一分解造成的计算量变化可以如下估计：首先，如果$\mathbf{dom}(\boldsymbol{\Lambda}')\subsetneq\mathbf{dom}(\boldsymbol{\Phi})$，采用式(3)进行计算的复杂度以指数小于直接计算$\prod_{\varphi\in\boldsymbol{\Phi}}\varphi$；其次，如果$\mathbf{dom}(\boldsymbol{\Lambda}')=\mathbf{dom}(\boldsymbol{\Phi})$，且$|\boldsymbol{\Lambda}|>0$，由于参与求和号右侧连乘的势函数个数变少，计算量也将按比例减少。

式(3)右侧的乘积-求和计算还可以利用势函数定义域之间的包含关系进一步进行优化。首先看一个例子，计算式(4)

$$F(X)=\sum_Y G_1(Y)F_1(X,Y)G_2(Y) \tag{4}$$

设X，Y均有N个可能取值，如果直接计算势函数的乘积，需要$2N^2$次乘法。然而，如果首先将$G_1(Y)$ 和$G_2(Y)$ 相乘，得到$G(Y)$，而后将$G(Y)$ 与$F_1(X,Y)$ 相乘，则需要的乘法次数减为N^2+N次，当N较大时，所需计算量约为直接相乘的 1/2。从这个例子中我们可以看出，如果将参与相乘的势函数按照定义域的包含关系进行排序，让定义域较小并且相互接近的势函数首先相乘，而后再与定义域较大的势函数相乘，那么所需的乘法次数可以显著的降低。为此，我们提出一种新的数据结构：包含树（Containing Tree）。在包含树中，势函数定义域之间的包含关系被组织成树结构。利用包含树上的算法能够有效的减少求解势函数连乘积所需的乘法次数。

3.1 包含树的构造和运算

设$\boldsymbol{\Phi}$为一势函数集合，并设$\mathbf{V}=\mathbf{dom}(\boldsymbol{\Phi})$，$\mathcal{T}$被称为$\boldsymbol{\Phi}$上的包含树，如果满足：

1) $\mathcal{T}$中每个节点a为一个二元组$\langle\boldsymbol{\Gamma}_a,\mathbf{V}_a\rangle$，其中$\boldsymbol{\Gamma}_a$为势函数集合，$\mathbf{V}_a$为变量集合；
2) 设$\mathcal{T}$的根节点为r，$r=\langle\boldsymbol{\Gamma}_r,\mathbf{V}\rangle$，其中$\boldsymbol{\Gamma}_r=\{\varphi|\varphi\in\boldsymbol{\Phi},\ \mathbf{dom}(\varphi)=\mathbf{V}\}$；

[1] 这中间蕴含了一个假设，即$\mathbb{T}$中没有变量集完全相同的节点。

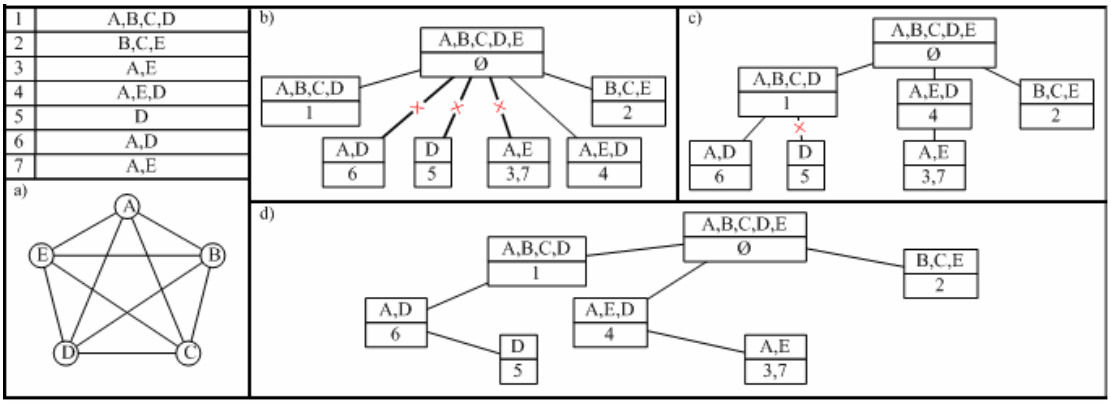

图 2 包含树的例子。图中左上角为输入的势函数及其编号，a）为对应的图模型。该图模型只含有 1 个簇（clique）。b）为初始的包含树，所有节点与根节点直接相连。c）为经过 1 次调整后的包含树，此时 b）中标明×的边均被删除。d）为最终输出的极小包含树，可直接利用算法 2 计算输入势函数的乘积。

3) 对$\mathcal{T}$的每个非根节点a，$\mathbf{\Gamma}_a$非空，$\mathbf{V}_a=\mathbf{dom}(\mathbf{\Gamma}_a)$；

4) 任取$\varphi_a, \varphi_b\in\mathbf{\Phi}$，$\mathbf{dom}(\varphi_a)=\mathbf{dom}(\varphi_b)$当且仅当$\varphi_a$和$\varphi_b$属于同一节点；

5) 如果a为b的父节点，有$\mathbf{V}_a \supsetneq \mathbf{V}_b$；

其中 2)，3)保证$\mathcal{T}$中除根节点外，节点变量集与节点势函数集对应；4)保证每个$\mathcal{T}$中节点的变量集唯一，即无重复的节点变量集；5)为包含树的核心特性，即子节点上的变量集为父节点变量集的真子集。

给定势函数集合$\mathbf{\Phi}$上的包含树$\mathcal{T}$，$\prod_{\varphi\in\mathbf{\Phi}}\varphi$可以使用算法 2 求出。在算法 2 中，势函数相乘的顺序为从叶子节点到根节点，即首先计算当前节点的所有子节点上缓存的乘积，而后将该乘积与当前节点上势函数相乘，置于当前节点的缓存中。

算法 2 利用包含树计算势函数乘积

输入：包含树$\mathcal{T}$

输出：$\mathcal{T}$中所有势函数的乘积

```
Calc_product(T)
1   for each node a from leave to root in T:
2   φ_a ← ∏_{□a} φ
3       C ← child set of a
4   if C ≠ Ø:
5   φ*_a ← ∏_{b,C} φ_b
6           φ_a ← φ_a · φ*_a
7       return φ_r
```

给定势函数集合$\mathbf{\Phi}$，有很多包含树与之对应。一种最简单的情况是，将定义在 $\mathbf{dom}(\mathbf{\Phi})$上的势函数置于包含树的根节点，令其它所有包含树的节点直接与根节点相连。容易证明，这样得到的包含树满足上面的性质 1)到 5)。显然，用这种方法定义的包含树不一定是最优的，因为一些包含关系并没有得到充分的利用。为此，我们定义包含树的极小性。一个包含树$\mathcal{T}$被称为是**极小**的，如果对$\mathcal{T}$中的任意节点 a，有：

1) a 没有子节点或仅有 1 个子节点，或者

2) 任取a的两个子节点u，v，有$\mathbf{V}_u\not\subset\mathbf{V}_v$，且$\mathbf{V}_v\not\subset\mathbf{V}_u$；

任意一个非极小的包含树都可以通过有限次变换变为极小的，其主要方法是：对包含树中的每个节点，检查其子节点，如果发现子节点的变量集间有包含关系，例如$\mathbf{V}_u\subset\mathbf{V}_v$，则将$u$的父节点重新设置为$v$。初始的包含树可以由前述的方法直接得到。算法 3 描述了这一

过程。

算法 3 构造极小的包含树

输入：包含树 $\mathcal{T}$

效果：$\mathcal{T}$ 被调整为极小的包含树

```
CT_construct(T)
1    if T contains only one node:
2    return
3    r←root of T
4    for each pair of children (u, v) of r:
5        if V_u⊂V_v:
6            reset parent of u to v
7    for each child u of r:
8        T_u←subtree rooted at u
9        CT_construct(T_u)
10   return
```

在算法 3 中，输入的包含树按照自根节点到叶节点的顺序，递归的调整。注意到 CT_construct 对包含树中的每个节点调用且只调用一次，因此算法的时间复杂度线性于包含树中的节点数目。

3.2 复杂度分析

对利用包含树计算势函数的连乘积，有如下两条结论。首先，利用算法 2 计算势函数乘积需要的乘法次数不多于直接相乘所需的乘法次数，且当包含树满足：

1) 存在某个非根节点，其势函数集含有多于 1 个势函数，或者
2) $\mathcal{T}$的深度大于等于 2，或者
3) 根节点含有多于 1 个子节点，且子节点的变量集的并$\mathbf{V}_{root}^{*}=\cup_{a\in\mathbf{ch}(root)}\mathbf{V}_a$为$\mathbf{V}_{root}$的真子集

则利用算法 2 计算乘积所需的乘法次数小于直接相乘所需的乘法次数。

其次，对算法 3，有如下结论：假定势函数乘积利用算法 2 计算，设经过算法 3 调整前的包含树为$\mathcal{T}$，调整后的包含树为$\mathcal{T}'$，则利用$\mathcal{T}'$计算乘积所需的计算量不大于利用$\mathcal{T}$进行计算所需的计算量，特别的，如果$\mathcal{T}'\neq\mathcal{T}$，利用$\mathcal{T}'$计算乘积所需的计算量小于利用$\mathcal{T}$进行计算所需的计算量。

4 实验

为了评估HMP算法和包含树算法的性能，我们利用随机生成的贝叶斯网络进行了测试。测试中使用的贝叶斯网络是由BNGenerator[7]生成的。实验分为三个部分：首先，我们测试了HMP算法和包含树算法用于无约束包含情况下贝叶斯网络推理的性能；其次，我们测试了算法在约束包含情况下性能；最后，我们将HMP算法和包含树算法用于动态贝叶斯网络。在实验中，我们测试了 4 种推理算法： Shenoy-Shafer算法（SS）、Lazy Propagation 算法[8]（LZ）、单独使用HMP算法（H0）和HMP算法结合包含树算法（H1）。在实验中，我们统计利用这些算法进行一次完整的消息传递（包括消息收集和消息分发）所需的浮点乘法和加法次数，作为评估性能的标准。其中，基本的Shenoy-Shafer算法被作为参考基线，我们将其它三种算法完成推理所需的乘法和加法次数与基线相除，计算相对改进量。此外，为了分析HMP算法和包含树算法的运行状况，我们列出了推理中HMP算法的最大迭代深度和包含树的最大深度作为参考。测试使用的软件工具为PyGM[2]。

在实验中，连接树的生成采用了一步预测算法（One Step Look Ahead），并综合利用了

[2] PyGM是笔者编写的用于图模型推理的工具包。PyGM采用Python语言编写，基于连接树算法进行图模型推理。

各种常用的启发量（heuristics），如状态空间，填入边（fill-in）等。对每个图模型，我们尝试 20 次连接树的构建，从中选取结果最好（宽度最小）的连接树作为推理中使用的连接树。为了比较的公平，一旦连接树选定，那么测试中使用的 4 种算法均使用这一选定的连接树进行推理（对 HMP 算法而言，这表明在顶层递归调用中，使用选定的连接树）。

4.1 无约束包含的贝叶斯网络推理

在这一实验中，我们利用 BNGenerator 产生了 150 个贝叶斯网络，其中每个贝叶斯网络含有 40 个隐变量和 20 个观测变量。变量的可能取值安排如下：20%的变量的可能取值均匀分布于 2－4 之间，10%的变量可能取值均匀分布于 35－40 之间，其余变量的可能取值均匀分布于 8－12 之间。贝叶斯网络中的势函数均为稠密的，其中的数值为随机生成的。根据节点的最大入度，全部 150 个贝叶斯网络被分为 5 组，每组 30 个模型，对应的节点最大入度分别为 3－7。实验结果如表 1。

表 1 实验结果：无约束包含的推理

In.	CT.	HMP.	Mul.Imp			Add.Imp		
Deg	Dep	Dep	LZ	H0	H1	LZ	H0	H1
3	3.5	1	1.2	1.2	2.9	1	1	1
4	3.6	1	1.1	1.1	4.5	1	1	1
5	3.7	1	0.83	0.83	5	1	1	1
6	3.8	1	1.6	1.6	7.1	1	1	1
7	3.9	1.1	0.86	0.86	6.6	1	1	1.03

注：In.Deg 为输入模型中变量的最大入度。CT.Dep 为包含树最大深度（即每组 30 个模型中最大深度的平均值，下同），HMP.Dep 为 HMP 算法迭代的最大深度。Mul.Imp 和 Add.Imp 为乘法次数和加法次数与基线 Shenoy-Shafer 算法的比值的倒数，即若 Mul.Imp 为 7，说明采用该算法所需的乘法数量为 Shenoy-Shafer 算法的 1/7。

从表 1 中，我们可以看出，在无约束包含条件下，单独使用 HMP 算法与 Lazy Propagation 算法有着相同的表现，而且两者与基线的 Shenoy-Shafer 算法差别不大。显然，这是由于在无约束包含的条件下，连接树是直接建立在输入图模型上的，从而比较充分的利用了模型中的条件独立性关系。这也可以从 HMP 算法中最大迭代深度均为 1（即对 HMP 的递归调用只进行 1 次）得到印证。另一方面，表 1 显示包含树算法对于削减乘法次数有着非常明显的效果，在最大入度较大时能够比较有效的削减运算量，这是由于在最大入度较大时，连接树中各节点的变量集也随之增大，从而使得由于利用包含关系带来的计算量削减更为明显。最后，在无约束的情况下，3 种算法都无法有效的削减加法次数。

4.2 约束包含的贝叶斯网络推理

在无约束包含的推理实验基础上，我们进行了有约束包含情况下贝叶斯网络推理的实验。在实验中，使用了最大入度为 3，4 的两组 60 个贝叶斯网络作为测试集。在测试中，我们在每个图模型中随机的选取两个约束包含集，每组含有 5 个变量，而后在这一约束下进行图模型的推理，结果列于表 2。

表 2 实验结果：有约束包含的推理

In.	CT.	HMP.	Mul.Imp			Add.Imp		
Deg	Dep	Dep	LZ	H0	H1	LZ	H0	H1
3	3.4	3.5	0.9	2.6	10.4	1.1	2.1	2.1
4	3.5	3.2	0.96	1.9	8.4	1.1	1.8	1.8

注：表中各栏的意义与表 1 相同。

从表 2 中我们可以看到，在加入约束包含后，HMP 算法的性能明显的好于基线的 Shenoy-Shafer 算法和 Lazy Propagation 算法。在存在约束包含的情况下，HMP 的迭代深度不再恒等于 1，表明初始的连接树不能充分的利用图模型中的条件独立性。其次，包含树算法对于削减乘法次数仍然起到了非常有效的作用，从表中可见，在最大入度为 3 时，采用 HMP 结合包含树算法进行推理所需的乘法次数仅为基线的 Shenoy-Shafer 算法的 1/10。最后，HMP 算法能够一定程度上降低推理所需的加法次数。

4.3 动态贝叶斯网络推理

在动态贝叶斯网络推理中，界面变量集（Interface Set）在每一帧的推理中必须首先被消除（边缘化），这是一个典型的约束消除问题。在实验中，我们使用 100 个随机生成的动态贝叶斯网络作为测试集。在这些动态贝叶斯网络中，我们按照界面变量集的势将测试模型分为 5 组，分别对应界面变量集含有 6、8、10、12、14 个变量的情况。测试用的动态贝叶斯网络的生成方式如下，我们首先利用BNGenerator生成帧的内部连接，而后随机选取指定数量的界面变量，对每个变量随机生成 1－3 条与下一帧连接的边。在动态贝叶斯网络推理的试验中，基线算法为在 1.5-slice连接树[6]上运行的Shenoy-Shafer算法。实验结果如表 3 所示。

表 3 实验结果：动态贝叶斯网络

Int. Num	CT. Dep	HMP. Dep	Mul.Imp			Add.Imp		
			LZ	H0	H1	LZ	H0	H1
6	3.3	2.9	1.2	1.4	1.9	1.1	1.1	1.1
8	3.4	3.5	2.8	3.9	7.8	2	3.7	3.7
10	3.7	3.6	1.1	5.1	8.9	1.2	3.8	3.8
12	3.9	3.8	4.1	8.1	12.2	3.1	5.4	5.4
14	3.9	3.6	3.1	8.5	9.6	3.7	4	4

注：表中除第一栏外，其余各栏的意义与表 1 相同，Int.Num 为界面变量集中变量的个数。

从表 3 中可以看到，随着界面节点集的增大，HMP 算法比基线的 1.5-slice Shenoy-Shafer 算法性能有明显的提高。在界面节点集为 12，14 时，采用 HMP 算法结合包含树带来接近 10 倍的计算量削减，说明 HMP 算法以及包含树算法能够更为有效的利用输入图模型本身的条件独立性关系，显著的改善由约束包含和约束消除带来的性能下降。

参考文献

[1] K.Kask, R.Dechter, J.Larrosa, and A.Dechter, "Unifying Cluster-Tree Decompositions for Reasoning in Graphical models," Artificial Intelligence, vol. 166, pp. 165-193, 2005.

[2] G.R.Shafer and P.P.Shenoy, "Axioms for Probability and Belief-Function Propagation," UAI-1988, pp. 169-198, 1988.

[3] K.Kask, R.Dechter, J.Larrosa, and F.Cozman, "Bucket-Tree Elimination for Automated Reasoning," Technical Report, 2001.

[4] C.Huang and A.Darwiche, "Inference in Belief Networks: A procedural guide," International Journal of Approximate Reasoning (IJAR), vol. 15, no. 3, pp. 225-263, 1996.

[5] Jeff Bilmes and Chris Bartels, "On Triangulating Dynamic Graphical Models," UAI-2003, pp. 47-56, 2003.

[6] K.Murphy, "Dynamic Bayesian Networks--Representation, Inference and Learning." Doctoral Thesis, U.C.Berkeley, 2002.

[7] "BNGnerator," http://www.pmr.poli.usp.br/ltd/Software/BNGenerator/, 2003.

[8] A.L.Madsen and F.V.Jenson, "Lazy Propagation in Junction Trees," UAI-1998), pp. 362-369, 1998.

[9] S.L.Lauritzen, Graphical Models, 1996.