Implementación paralela del algoritmo Belief Propagation

A. Mendiburu J. Miguel-Alonso J. A. Lozano*

Facultad de Informática, Universidad del País Vasco

20018 San Sebastián, Gipuzkoa

amendiburu@si.ehu.es j.miguel@ehu.es ja.lozano@ehu.es

Resumen

En el campo de la computación evolutiva existen diferentes propuestas que utilizan modelos probabilísticos para representar las (in)dependencias entre variables de un problema concreto. Entre estos métodos podemos citar los algoritmos belief propagation, que se aplican habitualmente en inferencia probabilística. En este trabajo presentamos una aproximación paralela de uno de estos algoritmos de inferencia, concretamente loopy belief propagation sobre factor graphs. Nuestra paralelización ha sido diseñada para poder ser ejecutada en multiprocesadores o clusters de ordenadores, reduciendo así el tiempo de ejecución. Además, se ha realizado un estudio del algoritmo, tratando de crear una herramienta flexible donde parámetros tales como planificación o condiciones de parada puedan ser fijados en función de las necesidades particulares de cada problema.

1. Introducción

La investigación realizada en el área de la computación evolutiva ha experimentado un desarrollo importante tras la integración de resultados provenientes de otras áreas de investigación. En algunos casos, estos resultados han contribuido a superar algunas limitaciones conocidas, ampliando el espectro de problemas sobre los que aplicar estas técnicas. Un claro ejemplo lo constituyen los algoritmos evolutivos que combinan modelos probabilísticos con técnicas de machine learning para guiar las búsquedas. Entre ellas, podemos destacar los algoritmos de estimación de distribuciones, Estimation of Distribution Algorithms (EDA) [9].

Los EDAs se caracterizan por el uso de modelos probabilísticos gráficos que se utilizan para representar las relaciones entre las variables de un problema concreto. Generalmente, se asume que es posible construir un modelo del espacio de búsqueda que puede ser usado para guiar la búsqueda hacia la solución óptima. Estos algoritmos han mostrado su eficiencia en un amplio rango de problemas de optimización [5]. A su vez, los modelos gráficos probabilísticos aplicados en los EDAs han resultado ser muy útiles para realizar un análisis teórico de los propios EDAs (por ejemplo, analizando la relación entre la estructura de la función y las distribuciones de búsqueda [8]).

Por otro lado, los algoritmos conocidos como belief propagation (BP) se utilizan habitualmente en modelos gráficos probabilísticos para inferencia, y pueden también ser aplicados a problemas de optimización [11, 14, 15]. Dado un modelo probabilístico gráfico, los algoritmos BP pueden obtener tanto la distribución de probabilidad marginal como el estado más probable de la distribución. Estos algoritmos utilizan operaciones de paso de mensajes hasta que se satisface una determinada condición de parada.

En comparación con los EDAs, no es necesario calcular la función de evaluación (sólo se realizan cálculos locales), ni existe la necesidad

*Este trabajo ha sido financiado por los proyectos SAIOTEK Autoinmune (II) 2006, y Etortek del Gobierno Vasco, y por los proyectos TIN 2005-03824 y TIN 2004-07440-C02-02 del Ministerio de Educación y Ciencia.

de utilizar poblaciones. En BP, las iteraciones de paso de mensajes sustituyen las generaciones utilizadas en los EDAs. En cambio, la limitación de los BPs, en comparación con los EDAs, radica en que es necesario disponer de información completa acerca de la estructura de la función a optimizar y, por lo tanto, no son aplicables a problemas en los que dicha estructura se desconoce. Una solución a este problema son los algoritmos mixtos, que incorporan características de ambas aproximaciones [7, 13]. Sin embargo, es fundamental para estos algoritmos mixtos disponer de implementaciones de algoritmos BP que sean eficientes y flexibles. La flexibilidad es necesaria para manipular los diferentes componentes del algoritmo, tales como métodos de inicialización, políticas de envío de mensajes, y condiciones de parada. Por otro lado, la eficiencia es importante cuando se quieren afrontar problemas de optimización complejos y se desean obtener resultados en un tiempo razonable.

Con estos objetivos, en este trabajo presentamos una aproximación paralela y flexible del algoritmo BP sobre factor graphs [4]. La paralelización parece ser una buena alternativa para obtener implementaciones rápidas si bien su escalabilidad necesita un estudio más detallado.

El resto del artículo está organizado de la siguiente manera: en la sección 2 se introducen el algoritmo BP y los factor graphs. La secciones 3 y 4 presentan el análisis del algoritmo BP y su diseño paralelo respectivamente. Los resultados obtenidos tras los experimentos se discuten en la sección 5. Finalmente, la sección 6 presenta las conclusiones y líneas futuras de investigación.

2. Factor graphs y belief propagation

En teoría de la probabilidad, un modelo gráfico es una estructura utilizada para representar las (in)dependencias entre un conjunto de variables. En dicha estructura (grafo), los nodos representan a las variables, y los arcos se utilizan para indicar (in)dependencias condicionales entre las variables.

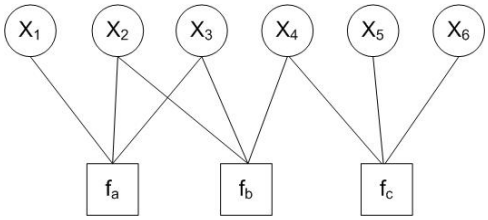

Figura 1: Ejemplo de un factor graph.

Gracias a las interesantes características que poseen los modelos gráficos, tales como representación visual de las dependencias y simplicidad para obtener las factorizaciones de la distribución de probabilidad, son usados en un amplio rango de problemas, tales como diagnóstico médico, procesamiento de habla e imagen, u optimización.

En el presente trabajo nos centramos en los factor graphs [4], que son grafos bipartitos con dos tipos de nodos: nodos variable y nodos factor. Cada nodo variable identifica a una variable X_i que toma valores de un dominio (habitualmente) discreto, mientras que los nodos factor f_j representan diferentes funciones cuyos argumentos son subconjuntos de variables. Se representan de manera gráfica mediante arcos que unen un determinado nodo función con sus nodos variable (argumentos). A modo de ejemplo, la Figura 1 muestra un factor graph con seis nodos variable $\{X_1, X_2, \ldots, X_6\}$ y tres nodos factor $\{f_a, f_b, f_c\}$.

Los factor graphs son adecuados para representar aquellos casos en los que la distribución de probabilidad conjunta puede ser expresada como una factorización de un conjunto de funciones locales:

$$g(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{j \epsilon J} f_j(\boldsymbol{x_j}) \qquad (1)$$

donde $Z = \sum_{\boldsymbol{x}} \prod_{j \epsilon J} f_j(\boldsymbol{x_j})$ es una constante de normalización, n es el número de nodos variable, J es un conjunto discreto de indices, $\boldsymbol{X_j}$ es un subconjunto de $\{X_1, \ldots, X_n\}$, y $f_j(\boldsymbol{x_j})$ es una función que contiene las variables de $\boldsymbol{X_j}$ como argumento.

Aplicando esta factorización al factor graph

de la Figura 1, la distribución de probabilidad conjunta resultaría:

$$g(x_1,\ldots,x_6) = \frac{1}{Z} f_a(x_1,x_2,x_3) \\ f_b(x_2,x_3,x_4) f_c(x_4,x_5,x_6) \tag{2}$$

Belief Propagation (BP) [11] es un método ampliamente utilizado para solucionar problemas de inferencia en modelos gráficos. Se aplica básicamente en dos situaciones: (1) cuando el objetivo es obtener distribuciones marginales para algunas de las variables presentes en el problema, y (2) cuando, dado un modelo, se busca el estado global más probable para un determinado problema. Estos dos casos se conocen habitualmente con el nombre de algoritmo *sum-product* y *max-product* respectivamente.

Se ha demostrado que el algoritmo BP es eficiente en estructuras con forma de árbol, y también se han presentado (de forma empírica) buenas aproximaciones para estructuras cíclicas. Este hecho ha sido ampliamente demostrado en diferentes aplicaciones, tales como low-density parity-check codes [12], turbo codes [6], procesamiento de imágenes [3], u optimización [2].

La característica principal del algoritmo BP radica en que la inferencia se calcula utilizando paso de mensajes entre los nodos. Cada nodo envía y recibe mensajes hasta que alcanza una situación estable. Los mensajes, calculados localmente por cada nodo, contienen información estadística de interés para los nodos vecinos.

Cuando el algoritmo BP se aplica a los factor graphs, es necesario utilizar dos tipos de mensajes: mensajes $n_{i\to a}(x_i)$ enviados de un nodo variable i a un nodo factor a, y mensajes $m_{a\to i}(x_i)$ enviados por un nodo factor a a un nodo variable i. Es importante indicar que se envía un mensaje para cada valor de cada variable X_i.

Estos mensajes se actualizan conforme a las siguientes reglas:

$$n_{i\to a}(x_i) := \prod_{c\epsilon N(i)\backslash a} m_{c\to i}(x_i) \tag{3}$$

$$m_{a\to i}(x_i) := \sum_{\boldsymbol{x_a}\backslash x_i} f_a(\boldsymbol{x_a}) \prod_{j\epsilon N(a)\backslash i} n_{j\to a}(x_j) \tag{4}$$

$$m_{a\to i}(x_i) := \arg \max_{\boldsymbol{x_a}\backslash x_i} \{f_a(\boldsymbol{x_a}) \prod_{j\epsilon N(a)\backslash i} n_{j\to a}(x_j)\} \tag{5}$$

donde $N(i)\backslash a$ representa todos los nodos factores vecinos del nodo i exceptuando al nodo a, y $\sum_{\boldsymbol{x_a}\backslash x_i}$ expresa que la suma se realiza teniendo en cuenta todos los posibles valores que todas las variables (excepto X_i) pueden tomar en $\boldsymbol{X}_a$ –mientras la variable X_i toma su valor x_i.

Las ecuaciones 3 y 4 se utilizan cuando se desean obtener probabilidades marginales (sum-product). Cuando el objetivo es encontrar la configuración más probable (max-product), se deben utilizar las ecuaciones 3 y 5.

Cuando el algoritmo converge (es decir, los mensajes no cambian), las funciones marginales (sum-product) o los máximos (max-product) se obtienen como el producto normalizado de todos los mensajes recibidos por X_i.

$$g_i(x_i) \propto \prod_{a\epsilon N(i)} m_{a\to i}(x_i) \tag{6}$$

3. Análisis del algoritmo BP

Tal y como se ha mencionado en la sección previa, BP es un algoritmo ampliamente estudiado y aplicado a diferentes problemas. Debido a ello, el algoritmo ha sufrido diferentes modificaciones desde su propuesta original. No obstante, la mayoría de estas propuestas han sido implementadas de manera secuencial, y tienen algunas limitaciones en lo que a número de nodos, vecinos, o políticas de planificación respecta. Por otro lado, y hasta donde nosotros sabemos, las implementaciones paralelas disponibles han sido desarrolladas para métodos de inferencia exacta [10].

Este hecho nos animó a diseñar una versión paralela del algoritmo BP sobre factor graphs.

De cara a ello, nos centramos en estudiar concienzudamente las características del algoritmo, con el propósito de diseñar una herramienta flexible que pueda ser parametrizada y aplicada a diferentes problemas. En algunos casos, el hecho de permitir que el usuario seleccione determinados valores para algunos parámetros puede hacer que el rendimiento del algoritmo mejore. En otros casos, permite ampliar el espectro de experimentación, realizando diferentes pruebas que permiten observar la importancia que pueden tener los parámetros seleccionados inicialmente. En lo que respecta al algoritmo BP, hemos detectado tres grupos de parámetros principales: (1) políticas de planificación –es decir, como y cuando se envían o reciben mensajes– (2) condiciones de parada –que indican cuando se debe detener el programa– y (3) valores iniciales para determinados parámetros –incluyendo por ejemplo, los valores iniciales de los mensajes.

3.1. Políticas de planificación

Un aspecto importante del algoritmo BP es la forma en la que los mensajes son propagados a través de los nodos que conforman el factor graph. En muchas implementaciones, la planificación se diseña siguiendo modelos síncronos, donde un reloj indica los envíos de mensajes. En nuestra implementación proponemos un modelo basado en reglas. De esta forma, el usuario puede definir las condiciones particulares que regulan el comportamiento de cada nodo, siendo posible definir reglas diferentes para cada uno de ellos. Se contemplan dos tipos de reglas:

- Número de mensajes: esta es la regla más simple. Se activa cuando un nodo recibe un número dado de mensajes. Entonces, calcula los nuevos mensajes y los envía hacia los nodos desde los que no ha recibido ningún mensaje.
- Conjuntos de mensajes: esta regla es más compleja, y permite definir pares ($CjtRcb$,$CjtEnv$). $CjtRcb$ y $CjtEnv$ representan subconjuntos de nodos. Cuando se han recibido mensajes de todos los nodos contenidos en $CjtRcb$, se calcularán nuevos mensajes, y estos serán enviados a los nodos definidos en $CjtEnv$.

3.2. Criterios de parada

Cuando la ejecución comienza, cada nodo actúa en base a las reglas y parámetros definidos en la configuración inicial, recibiendo, calculando, y enviando mensajes hacia otros nodos. Es sabido que el algoritmo BP converge cuando la estructura es acíclica. Es decir, se alcanza una situación estable donde los valores de los mensajes son fijos. Por otro lado, cuando BP se aplica sobre estructuras con ciclos, es posible que el algoritmo obtenga buenos resultados, pero no se puede garantizar que se alcanzará una situación estable.

Este es el motivo por el que resulta interesante definir diferentes criterios de parada. Teniendo en cuenta las diferentes situaciones, hemos definido tres criterios de parada que son comprobados de manera independiente en cada nodo. El algoritmo se detiene si: (1) en las últimas i iteraciones (cálculo de mensajes) se obtiene el mismo valor de mensaje, (2) en las últimas i iteraciones se repite la misma secuencia de valores (esto es, se detecta una situación cíclica, donde mensajes con valores m_1,m_2,...,m_i son calculados repetidamente), o (3) se calcula un número máximo de mensajes.

3.3. Parámetros iniciales

Además de las políticas de planificación y de los criterios de parada existen algunos parámetros iniciales que deben ser considerados. Algunos deben ser fijados por el usuario, y están ligados con el problema que se desea solucionar. Por ejemplo, los valores de las funciones de los nodos factor, o los valores iniciales de los mensajes de cada nodo. También es posible indicar que un nodo siempre va a enviar el mismo mensaje (valor) a sus vecinos. En nuestra implementación, se han añadido además los siguientes parámetros:

- Diferencia permitida: cuando se comparan dos mensajes, se considerarán iguales siempre y cuando la diferencia entre am-

bos sea inferior al valor definido en este parámetro.

- Número máximo de mensajes: contemplado para detener la ejecución del algoritmo si no se alcanza una situación estable tras calcular dicho número de mensajes.
- Número de comparaciones necesarias: fija el número de comparaciones requeridas (entre mensajes) de cara a considerar una situación estable.
- Tamaño de la cache: determina el número máximo de mensajes a almacenar en cada nodo.
- Algoritmo: permite escoger entre sum-product o max-product.

4. Diseño paralelo

En los últimos años, la popularización de los procesadores multi-núcleo, los clusters de ordenadores y las grids computacionales han impulsado el diseño de aplicaciones paralelas. Siguiendo esta tendencia, hemos diseñado una aplicación paralela que puede ser ejecutada en diferentes plataformas, tales como multiprocesadores o clusters de ordenadores.

La aplicación ha sido implementada utilizando una combinación de Message Passing Interface (MPI) y POSIX threads. De cara a la aplicación, existiría una paralelización ideal, en la que cada nodo esta relacionando con un procesador, siendo este responsable de recibir, calcular, y enviar mensajes. No obstante, esta aproximación depende directamente del número de nodos, y teniendo en cuenta los tamaños de los problemas que se abordan habitualmente (desde cientos a miles de nodos), sería inviable desde el punto de vista de los recursos necesarios.

Basándonos en esta idea inicial, proponemos una solución en la que cada proceso se responsabilizará de un grupo de nodos del factor graph. De esta forma, el número de procesadores no dependerá del número de nodos, permitiendo que el algoritmo sea ejecutado en un amplio conjunto de escenarios. Al tratarse de un algoritmo con un alto grado de comunicación, es importe tratar de diseñar la aplicación de manera que ello no suponga un cuello de botella desde el punto de vista de rendimiento.

A nivel MPI, hemos utilizado un esquema maestro-trabajador, en el que el proceso maestro se utiliza para distribuir los parámetros de configuración (estructura del factor graph, políticas, valores iniciales), y para recoger los resultados. Por otra parte, los procesos trabajador se encargarán de recibir, calcular, y enviar mensajes para el subconjunto de nodos que gestionan.

En dichos procesos trabajador hemos decidido utilizar dos tipos de hilos: (1) un hilo para recibir mensajes MPI, (2) uno o más hilos para procesar los mensajes recibidos, realizar los cálculos pertinentes y enviar los correspondientes mensajes.

En los Algoritmos 1 y 2 se describe el pseudocódigo para el maestro y los trabajadores.

```
Paso 1. Obtener estructura del factor
        graph desde un fichero
Paso 2. Obtener la configuración
        inicial desde un fichero
Paso 3. Enviar la estructura y la con-
        figuración a los trabajadores
Paso 4. Esperar a que terminen los
        trabajadores y almacenar
        sus mensajes
Paso 5. Enviar la orden de parar
Paso 6. Mostrar los resultados
Paso 7. Parar.
```

Algoritmo 1: Pseudocódigo del proceso maestro

```
Paso 1. Recibir estructura y confi-
        guración desde el maestro
Paso 2. Para cada nodo n
            Si Comienza enviando
                Buscar la regla
                Calcular mensages
                Enviar mensajes
            Fin Si
        Fin Para
Paso 3. Mientras Nodos no fijos
            Esperar un mensaje
```

```
            Para cada nodo n
                Buscar regla
                Si reglaEncontrada
                   Calcular mensages
                   Comprobar condición
                   de parada
                   Enviar mensajes
                Fin Si
            Fin Para
        Fin Mientras
Paso 4. Enviar mensajes finales
        al maestro
Paso 5. Esperar la orden parar
Paso 6. Parar.
```

Algoritmo 2: Pseudocódigo del proceso trabajador

5. Experimentos

De cara a estudiar el rendimiento de esta implementación paralela, hemos realizado una serie de experimentos preliminares sobre un cluster de ordenadores utilizando una conocida función de optimización, checkerboard. En este problema, presentado en [1], partimos de un casillero que contiene $s \times s$ posiciones, donde cada posición puede tomar un valor 0 ó 1. El objetivo es que las posiciones tomen valores de tal forma que una posición con valor 1 estará rodeada en sus cuatro direcciones básicas por valores 0, y viceversa. La función de evaluación cuenta el número de valores vecinos correctos, siendo la dimensión del problema $n = s^2$, y el valor máximo $2s(s-1)$.

Este problema se representa mediante factor graphs añadiendo nodos factor horizontal y verticalmente, entre cada par de posiciones (nodos variable) del casillero. La función factor toma un valor 0 cuando sus argumentos tienen el mismo valor, y 1 en caso contrario.

Los experimentos se han realizado en un cluster de ordenadores de 4 nodos interconectados mediante una red Gigabit Ethernet conmutada. Cada nodo dispone de dos procesadores Intel Xeon (2.4GHz, 512KB de memoria cache, tecnología hyperthreading), y 2GB de memoria RAM (compartida). El sistema operativo es Linux, la implementación de MPI es MPICH2[1] (versión 1.0.5), y el compilador de c++ es la versión 9.1 de Intel.

Respecto al problema, hemos considerado conveniente utilizar un tamaño que nos permita realizar un elevado número de experimentos en un tiempo razonable, buscando además que las conclusiones obtenidas sean válidas (salvando las distancias) para otros tamaños. Es necesario recalcar que en estos experimentos no buscamos resolver el problema, sino que nos centramos en estudiar el comportamiento de la versión paralela. Por ello, el tamaño del casillero será de 60×60, lo que supone un factor graph de 10.680 nodos. Respecto al resto de parámetros, la política de planificación está basada en reglas, los nodos variable comienzan enviando mensajes y, a partir de ahí, cada uno de los nodos calculará y enviará nuevos mensajes hacia sus vecinos cada vez que reciba mensajes de todos y cada uno de ellos. La diferencia permitida se ha fijado en 1.0e-3, el número máximo de mensajes es 30, el tamaño de la cache es 20, 10 es el número de comparaciones a realizar, y el algoritmo es max-product. Finalmente, el subconjunto de nodos a asignar a cada trabajador se calcula en función del número de trabajadores (procesos MPI), tratando de equilibrar el número de nodos variable y nodos factor repartidos en cada subconjunto.

Los experimentos se han realizado combinando diferentes números de nodos, activando y desactivando la opción de hyperthreading (HT). Sin HT, se ejecuta en cada nodo biprocesador un proceso MPI con dos hilos de cálculo (en adelante M+T), o dos procesos MPI con un único hilo (en adelante MPI). Con HT activado, duplicamos el número de procesos MPI.

Según puede observarse en los resultados obtenidos (ver Cuadro 1), la eficiencia del programa paralelo sobre un único nodo (ordenador) es bastante buena (hasta el 90 %), pero al utilizar más nodos la aplicación acusa la comunicación entre ellos, notándose un importante descenso del rendimiento al pasar a dos nodos. Dicho descenso se suaviza al aumentar el numero de nodos.

Respecto a la tecnología HT, esta resulta en

[1] http://www-unix.mcs.anl.gov/mpi/mpich2

Cuadro 1: Tiempo de ejecución (en segundos) para el problema 60×60, utilizando hasta un total de 4 nodos (8 CPUs), sin y con hyperthreading, para las implementaciones MPI y M+T. El tiempo de ejecución de la versión secuencial es de 225 segundos.

Hyperthreading desactivado								
	# procesadores							
	2		4		6		8	
	tiempo	eficiencia	tiempo	eficiencia	tiempo	eficiencia	tiempo	eficiencia
MPI	157	0,72	89	0,63	64	0,59	52	0,55
M+T	124	0,91	99	0,57	72	0,52	58	0,48
Hyperthreading activado								
	# procesadores							
	2		4		6		8	
	tiempo	eficiencia	tiempo	eficiencia	tiempo	eficiencia	tiempo	eficiencia
MPI	140	0,80	75	0,75	63	0,59	59	0,48
M+T	139	0,81	81	0,69	59	0,63	48	0,58

general beneficiosa para la mayoría de los casos, y particularmente para la versión M+T. Este hecho confirma que la aplicación tiene ciertas necesidades de comunicación que permiten que un incremento del número de procesos (sobrecarga de procesadores) resulte positivo. No obstante, teniendo en cuenta que esta es una aproximación preliminar, resulta necesario ampliar los experimentos realizados de cara optimizar en lo posible el rendimiento de la aplicación.

6. Conclusiones y trabajo futuro

En este trabajo se ha presentado una primera aproximación a la paralelización del algoritmo BP, y particularmente para la aproximación basada en factor graphs. Dicho algoritmo se utiliza habitualmente para obtener probabilidades marginales, o el estado más probable de una distribución de probabilidad. El algoritmo se basa en un intercambio constante de mensajes entre nodos hasta que se cumpla una condición de parada.

Antes de comenzar con el diseño de la versión paralela se procedió a estudiar el comportamiento del algoritmo, tratando de crear una herramienta lo más flexible posible. De esta manera, el usuario puede decidir la política de planificación, los valores iniciales, los nodos que comenzarán a enviar mensajes, o las condiciones que determinarán que la ejecución finalice. La implementación se ha realizado utilizando MPI y threads, siguiendo el esquema maestro-trabajador, donde el maestro reparte las configuraciones iniciales y espera para recoger los resultados, mientras los trabajadores ejecutan los cálculos necesarios para un subconjunto de nodos. Nuestros experimentos iniciales muestran que la aproximación paralela resulta interesante, si bien se hace necesario estudiar su escalabilidad en detalle y sobre un número de ordenadores mayor.

Finalmente, tenemos la intención de extender este trabajo en diferentes líneas. Por un lado, comprobar las planificaciones. Es decir, una vez que el usuario ha diseñado unas reglas de planificación adecuadas para un problema, sería interesante comprobar que no existan bloqueos (que todos los nodos se queden detenidos esperando un mensaje que no llega). Por otro lado, teniendo en cuenta la importancia que tiene la comunicación entre nodos, sería interesante diseñar un algoritmo que distribuya los nodos del factor graph en subconjuntos de manera que se minimicen las comunicaciones entre los mismos. Finalmente, existen herramientas que permiten realizar un seguimiento de las ejecuciones a diferentes niveles, de manera que se puedan localizar cuellos de botella que afecten al rendimiento (velocidad) del algoritmo paralelo.

Referencias

[1] S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. Technical report, Carnegie Mellon Report, CMU-CS-97-107, 1997.

[2] M. Bayati, D. Shah, and M. Sharma. Maximum weight matching via max-product belief propagation. *IEEE Transactions on Information Theory*, Accepted for publication.

[3] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.

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

[5] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors. *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms.* Springer-Verlag, 2006.

[6] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng. Turbo Decoding as an Instance of Pearl's "Belief Propagation."Algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152, 1998.

[7] H. Mühlenbein and R. Höns. The factorized distributions and the minimum relative entropy principle. In M. Pelikan, K. Sastry, and E. Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, Studies in Computational Intelligence, pages 11–38. Springer-Verlag, 2006.

[8] H. Mühlenbein, T. Mahnig, and A. Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):213–247, 1999.

[9] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In H. M. Voigt, W. Ebeling, I. Rechenberger, and H. P. Schwefel, editors, *PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 178–187. Springer, 1996.

[10] V. K. Namasivayam and V. K. Prasanna. Scalable Parallel Implementation of Exact Inference in Bayesian Networks. In *ICPADS (1)*, pages 143–150. IEEE Computer Society, 2006.

[11] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, Palo Alto, CA, 1988.

[12] T. J. Richardson and R. L. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2):599–618, 2001.

[13] R. Santana. *Advances in Probabilistic Graphical Models for Optimization and Learning: Applications in Protein Modelling.* PhD thesis, 2006.

[14] C. Yanover and Y. Weiss. Finding the M most probable configurations using loopy belief propagation. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16.* MIT Press, Cambridge, MA, 2004.

[15] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.