Computing the Maximum Independent Set of a Graph by Message-Passing

Matthew G. Parker et al *

August 31, 2014

Consider an n-vertex simple graph, $\mathcal{G}$, as described by the $n \times n$ binary matrix, Γ. We wish to find a maximum independent set, $\alpha(\mathcal{G})$, of vertices in $\mathcal{G}$ by using a message-passing algorithm on a graph related to $\mathcal{G}$.

1 A binary graph

Associate every vertex, v_j, of $\mathcal{G}$ to a length-two vector x_j, and a length 2^{d_j+1} vector, f_j, where d_j is the vertex degree of v_j in $\mathcal{G}$. In the final answer each vertex v_j shall be biased towards a 'white' colour, as represented by $x_j = (1, 0)$, up to normalisation, or biased towards 'black', as represented by $x_j = (0, 1)$, up to normalisation. The v_j that are biased towards black shall identify the independent set. The a priori 'soft information' entered into each x_j variable is given by $g_j = (1, 1)$ to represent that we have no a priori knowledge as to the final colour of v_j (this could be modified slightly as, for a random graph, the a priori probabilities of white and black are not equal). The f_j vectors identify allowable configurations and these configurations are weighted towards the more desirable configurations. We proceed by example. Consider the graph $\mathcal{G} = 10, 02$ (the 'path graph' P_3) . Then f_1 and f_2 are both functions of two binary variables, so are of length $2^2 = 4$. Moreover, f_0 is a function of three binary variables, so is of length $2^3 = 8$. The factor graph is shown in fig 1.

Table 1 shows the function biases, chosen to favour the solutions with most blacks, i.e. where $c > b > a > 0$, and $e > d > 0$.

We now explain these assignments to the f_j. Consider f_0. We see that f_0 only allows the following scenarios for $(x_0x_1x_2)$, namely $(000), (100), (010), (001), (011)$. We see, for instance, that (101) means that $x_0 = x_2 = 1$, implying that both v_0 and v_2 are black - but they are connected, so this is impossible. Likewise (110) and (111) also imply connected black vertices, so these scenarios are also disallowed. But (000) is never going to be part of a final solution as (100) is always to be preferred, so we do not include (000) in our allowable scenarios. For the allowed scenarios for f_0, $(100), (010), (001), (011)$, we prefer (011) over $(100), (010), (001)$

*

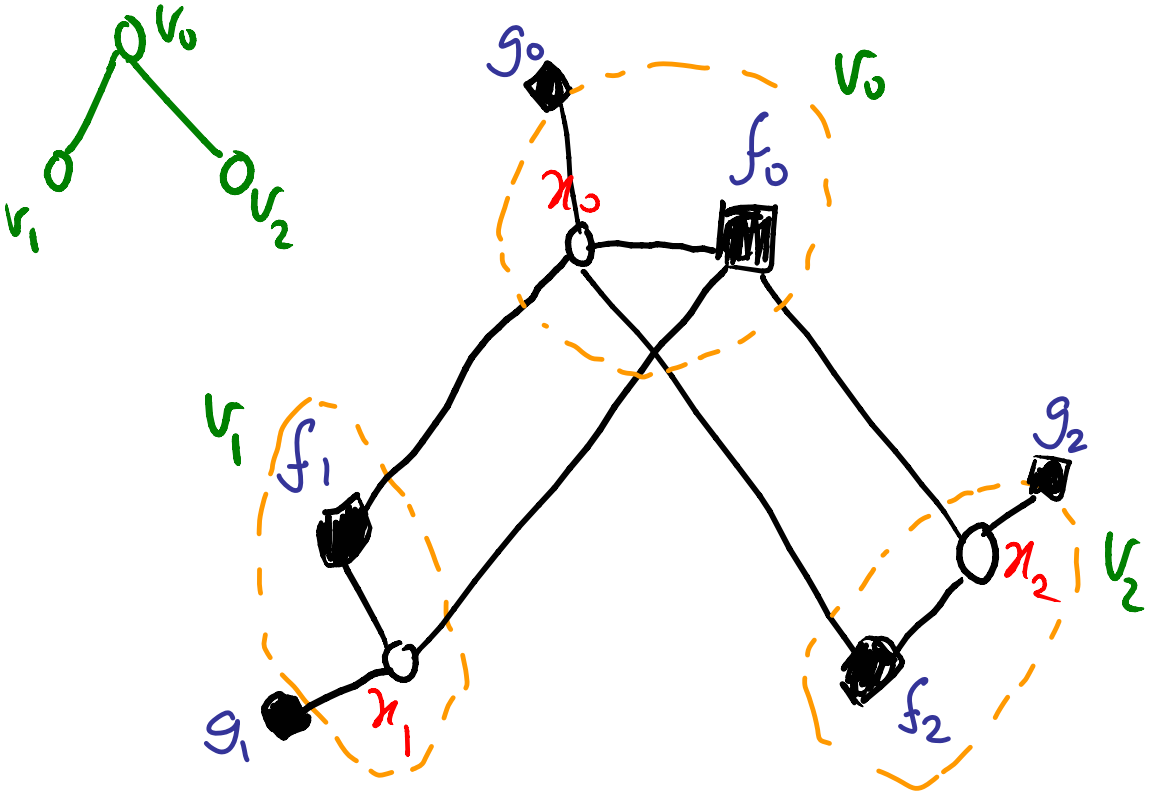

Figure 1: Binary factor graph for finding $\alpha(P_3)$

as we are trying to maximise the number of black vertices. These preferences are forced by assigning $c > b > 0$. Similarly, for f_1, the allowed scenarios are $(10-), (01-)$, and $(11-)$ is disallowed. The scenarios $(10-), (01-)$ are equally preferred so we assign both of them the value $e > 0$. Our complete 'code' is given by $f_0 f_1 f_2$. There are two 'codewords' $(100), (011)$. The two codewords describe possible independent scenario configurations but, by assigning $c > b > 0$, $e > 0$, we force the 'most likely' codeword to be (011). It is expected that implementing the sum-product algorithm (SPA) or max-product algorithm (MPA) on the associated factor graph will force a convergence to (011). This codeword precisely identifies the maximum independent set $\{1, 2\}$. The exact assignments to b, c, e remain unclear, apart from the requirements $c > b > 0$ and $e > 0$.

2 An $\mathbb{F}_4$ graph

Associate every vertex, v_j, of $\mathcal{G}$ to a length-four vector x_j, and a length 4^{d_j+1} vector, f_j, where d_j is, again, the vertex degree of v_j in $\mathcal{G}$. We associate $\mathcal{G}$ with the self-dual $\mathbb{F}_4$-

x_0	0	1	0	1	0	1	0	1
x_1	0	0	1	1	0	0	1	1
x_2	0	0	0	0	1	1	1	1
f_0	0	b	b	0	b	0	c	0
f_1	0	e	e	0	0	e	e	0
f_2	0	e	0	e	e	0	e	0
$f_0f_1f_2$	0	be^2	0	0	0	0	ce^2	0

Table 1: Function factorizations for binary solution

additive code with parity-check matrix $H_4 = G_4 = \begin{matrix} w & 1 & 1 \\ 1 & w & 0 \\ 1 & 0 & w \end{matrix}$. This code has 8 codewords $(000), (w11), (1w0), (w^2w^21), (10w), (w^21w^2), (0ww), (ww^2w^2)$. Let $r = (r_0r_1r_2) \in \mathbb{F}_2^3$ be such that $r_j = 1$ iff v_j is coloured 'black'. For instance, for $r = (101)$, $rG_4 = (101)G_4 = (w^21w^2)$, so the codeword (w^21w^2) is associated with the scenario v_0 and v_2 'black', and v_1 'white'. We don't want this scenario as v_0 is connected to v_2 in $\mathcal{G}$ - this connection between v_0 and v_2 is reflected be (w^21w^2) containing a w^2 element. As we do not want this scenario we shall bias our constraints so that the probability of this codeword is very low or zero. It is clear that any codeword with one or more w^2 elements is not a candidate solution. The codeword we wish to decode to is $(0ww)$ as this occurs when $r = (011)$, associated with the scenario v_0 white and v_1 and v_2 black, and this identifies the maximum independent set $\{1, 2\}$. So we wish to bias our constraints so that the sum-product algorithm decodes to $(0ww)$. In general, there may be more than one maximum independent set, i.e. more than one correct answer.

Length-4 vectors are passed along the edges, i.e. the vector sent to and from x_j is (p, q, r, s), where, by convention, we $\frac{p}{p+q+r+s}$, $\frac{q}{p+q+r+s}$, $\frac{r}{p+q+r+s}$, and $\frac{s}{p+q+r+s}$, are the probabilities of $x_j = 0$, 1, w, and w^2, respectively. In the final answer, if v_j is coloured white then $r_j = 0$, therefore $x_j \in \{0, 1\}$, and this is represented by a soft approximation to the vector $x_j = (1, 0, 0, 0)$ or $x_j = (0, 1, 0, 0)$, (meaning that $x_j = 0$ or $x_j = 1$, respectively). If v_j is finally coloured black then $r_j = 1$ and $x_j = w$ and this is represented by a soft approximation to the vector $x_j = (0, 0, 1, 0)$). We bias our constraints so that the scenario $x_j = w^2$ ($x_j = (0, 0, 0, 1)$) is of low or zero probability as this identifies connected black vertices. If we choose zero probability for $x_j = w^2$, then we only have to pass length 3 vectors along edges.

The a priori 'soft information' entered into each x_j variable is given by $g_j = (1, 1, 1, 0)$ (or $g_j = (1, 1, 1)$ if we ignore w^2) to represent that we have no a priori knowledge as to the final colour of v_j, except that we do not allow for the final possibility that $x_j = w^2$. The f_j vectors identify allowable configurations and these configurations are weighted towards the more desirable configurations. We proceed by example. Consider, once again, the graph $\mathcal{G} = 10, 02$. Then f_1 and f_2 are both functions of two quaternary variables, so are of length $4^2 = 16$. Moreover, f_0 is a function of three quaternary variables, so is of length $2^3 = 8$. The factor graph is identical to that for the binary case so is shown in fig 1, where the labels are

x_0	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w
x_1	0	0	0	1	1	1	w	w	w	0	0	0	1	1	1	w	w	w	0	0	0	1	1	1	w	w	w
x_2	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	w	w	w	w	w	w	w	w	w
f_0	0	0	b	0	0	b	0	b	0	0	0	b	0	0	b	0	b	0	0	0	b	0	0	b	0	c	0	0
f_1	0	0	0	0	0	e	e	e	0	0	0	0	0	0	e	e	e	0	0	0	0	0	0	e	e	e	e	0
f_2	0	0	0	0	0	0	0	0	0	0	0	e	0	0	e	0	0	e	e	e	0	e	e	0	e	e	e	0
$f_0f_1f_2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	be^2	0	0	0	0	0	0	0	0	0	ce^2	0	0	

Table 2: Function factorizations for $\mathbb{F}_4$-additive solution

now interpreted differently.

Ignoring the possibility of one or more x_j being w^2 we have the assignments shown in table 2, where we choose $c > b > 0$, $e > 0$, so as to bias our result in favour of the codeword $(0ww)$.

We see, for instance, that f_0 only allows the following scenarios for $(x_0x_1x_2)$, namely $(000), (w00), (010), (w10), (1w0), (001), (w01), (011), (w11), (1w1), (10w), (11w), (0ww)$. For instance, $(10w)$ means that $x_0 = 1$, $x_1 = 0$, $x_2 = w$, implying that v_2 is black and v_0 and v_1 are white. Note that f_0 does not allow, for instance, (www), even though it satisfies the Hermitian constraint, because it signifies that v_0 and v_1 are connected, and that v_0 and v_2 are connected ... both these scenarios are disallowed by the independent set condition. The complete code is described by the constraint $f_0f_1f_2$ and identifies the 2 codewords as $(w11)$ and $(0ww)$, which occur with probabilities (assuming normalisation) be^2 and ce^2, respectively. By making $c > b > 0$, $e > 0$, we (hopefully) force the graph to decode to the codeword $(0ww)$, thereby identifying the maximum independent set of $\mathcal{G}$ to be $\{1, 2\}$.

Some questions remain:

- Will the system converge?
- How should we choose b, c, etc?
- Should be zero the probabilities of forbidden scenarios or just set such probabilities to be low?
- Should we modify the a priori soft information from g_j to reflect that, for a graph $\mathcal{G}$, of a certain size, the probabilities of a vertex being black or white are not equal (because $|\alpha| << n/2$ for large graphs where n is large). This should enhance decoding I think.
- If we have more than one maximum independent set then the method will try to decode to more than one solution. How can we encourage convergence to just one solution? Observe that any set of equally valid solutions can always be ordered, arbitrarily, by a lexicographical ordering of the w positions. For instance, let $0w10w1w$ and $w0110ww$ be two maximum independent set solutions. Then we could choose to favour $0w10w1w$ as the binary expansion of 0100101 is less than that of 1000011 (with lsbs on the left). So we could:
  - Tweak the initial soft information slightly away from $g_j = (g_{0,j}, g_{1,j}, g_{w,j}, g_{w^2,j}) = (1, 1, 1, 0)$, so that $g_{w,j} < g_{w,k}$ if $j < k$.

– Tweak the biases of f_j slightly so that f_k favours words with ws in more than f_j if $k > j$.

- Is there any real difference between the binary graph solution and the $\mathbb{F}_4$-additive graph solution?

2.1 Dynamic decoding on the $\mathbb{F}_4$ graph - this section needs updating but is correct if you set $a = d = 0$ throughout

Unlike the binary solution, the $\mathbb{F}_4$ solution sits 'inside' an additive code. For instance, the code $H_4 = G_4 = \begin{matrix} w & 1 & 1 \\ 1 & w & 0 \\ 1 & 0 & w \end{matrix}$ generates the 8 codewords $(000), (w11), (1w0), (w^2w^21), (10w), (w^21w^2), (0ww), (ww^2w^2)$, which comprise the additive $[3, 2^3]$ code, and where H_4 describes the graph $\mathcal{G}$. The codewords that represent feasible solutions to the independent set problem for graph $\mathcal{G}$ are a **subset** of these 8 codewords, namely $(000), (1w0), (w11), (10w), (0ww)$, i.e. there does not exist a codeword that is an independent set solution that is not one of these 8 codewords. Also, no codeword with one or more w^2 elements can be a solution and, conversely, all codewords without one or more w^2 elements is a possible solution. In contrast the binary solution comprises the solutions $(000), (100), (010), (001), (011)$ but does not sit inside a binary linear code of length 3.

Intuitively one feels the fact that the $\mathbb{F}_4$ solution sits inside an additive code should enhance decoding by message-passing. At the very least it allows us to perform dynamic decoding by local complementation (LC) as LC does not change (up to element permutations) the solution space of 2^n codewords, so the subset of codewords that are independent set solutions are still contained in this solution space. But how do we track functional probabilities across graphs in the LC orbit? To understand the question as well as some possible answers, let us consider, once again, the graph, $\mathcal{G}$, of fig 1 and table 2 with $H_4 = G_4 = \begin{matrix} w & 1 & 1 \\ 1 & w & 0 \\ 1 & 0 & w \end{matrix}$.

We implement an LC on vertex 0 by adding row 0 to row i iff vertex i is connected to vertex 0 in the associated graph. So we obtain

$$H_4^0 = G_4^0 = \begin{matrix} w & 1 & 1 \\ w^2 & w^2 & 1 \\ w^2 & 1 & w^2 \end{matrix}.$$

To convert this matrix into graph form we would perform the permutation $w^2 \leftrightarrow 1$ on row 0, and $w^2 \leftrightarrow w$ on rows 1 and 2, to obtain

$$\tilde{H}_4^0 = \tilde{G}_4^0 = \begin{matrix} w & 1 & 1 \\ 1 & w & 1 \\ 1 & 1 & w \end{matrix},$$

which describes graph $\mathcal{G}^0$.

However, as we are explicitly discussing the codewords here it is perhaps easier for us to work with H_4^0 and implicitly remember how to interpret this matrix as the appropriate graph, $\mathcal{G}^0$. The problem we have to solve is how LC on vertex 0 updates functions f_1 and f_2 to f_1^0 and f_2^0, i.e. the functions at vertices 1 and 2. For instance, whilst f_1 is a function of just x_0 and x_1, as evidenced by row 1 of H_4, we see that f_1^0 is a function of x_0, x_1, and x_2, as evidenced by the non-zero entries of row 1 of H_4^0. So we could use the same a, b, c, structure as used for f_0 of H_4, with the difference that w^2 and 1 swap roles for vertex 0, and w^2 and w swap roles for vertex 1.

Function update rule no. 1:
LC at 0 is performed by vertex 0 sending row 0 of H_4 to its neighbouring vertices, i.e. to rows 1 and 2. Then vertices 1 and 2 can locally construct rows 1 and 2 of H_4^0, respectively, and must update their local functions, f_1^0 and f_2^0 accordingly. For instance, function f_1^0 should reflect the constraint described by row 1 of H_4^0, namely (w^2w^21). Vertex 1 keeps, permanently, in memory, row 1, r_1, of the initial matrix, H_4, i.e. $r_1 = (1w0)$, which tells the vertex that it is vertex 1 and that it was initially connected to vertex 0. Row 1 of H_4^0 tells vertex 1 that it is now connected to vertices 0 and 2. But it is important to remember that we are trying to find a maximum independent set for the original graph, $\mathcal{G}$, not the new graph, $\mathcal{G}^0$.

The constraint (w^2w^21) restricts to the following set of codewords, $\mathcal{S}_{(w^2w^21)} = \mathcal{S}_{r_1^0}$, as follows:

$$\mathcal{S}_{(w^2w^21)} = \mathcal{S}_{r_1^0} = \{e(\{1,w\},\{1,w\},\{w\})^T, \qquad e \in \mathbb{F}_2^3, \mathrm{wt}(e) \text{ even}\},$$

where we omit all codewords with w^2 elements. Vertex 1 has the row $r_1 = (1w0)$ stored so it knows that no codeword solutions may be coloured black at positions 0 and 1. Such codewords form the set $\mathcal{S}_1^B = \{(ww0),(ww1),(www)\}$, so the final set of allowed codewords, $\mathcal{S}_1^0 = \mathcal{S}_{r_1^0} \setminus \{\mathcal{S}_{r_1^0} \cap \mathcal{S}_1^B\}$. Explicitly,

$$\begin{array}{rcl}
\mathcal{S}_{r_1^0} = \{(\{0\},\{0\},\{0,1\})\} & = & \{(000),(001)\}, \\
\cup\{(\{1,w\},\{1,w\},\{0,1\})\} & = & \{(110),(w10),(1w0),(ww0), \\
 & & (111),(w11),(1w1),(ww1)\} \\
\cup\{(\{1,w\},\{0\},\{w\})\} & = & \{(10w),(w0w)\} \\
\cup\{(\{0\},\{1,w\},\{w\})\} & = & \{(01w),(0ww)\}.
\end{array}$$

$$\mathcal{S}_1^B = \{(w,w,0),(w,w,1),(w,w,w)\},$$

and

$$\begin{array}{rl}
\mathcal{S}_1^0 = & \{(000),(001),(110),(w10),(1w0),(111), \\
 & (w11),(1w1),(10w),(w0w),(01w),(0ww)\}.
\end{array}$$

We now assign probabilities a, b, and c, to codewords in $\mathcal{S}_1^0$ with w weight 0, 1, and 2, respectively, where $a < b < c$, and thereby compute function f_1^0, as shown in table 2.1. Also shown in table 2.1 is f_2^0 which is computed in a similar way. Comparing with table 2 we see that the global function $f_0f_1f_2$ is not the same as $f_0f_1^0f_2^0$, but the same five possible solutions

x_0	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w
x_1	0	0	0	1	1	1	w	w	w	0	0	0	1	1	1	w	w	w	0	0	0	1	1	1	w	w	w
x_2	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	w	w	w	w	w	w	w	w	w
f_0	a	0	b	a	0	b	0	b	0	a	0	b	a	0	b	0	b	0	0	b	0	0	b	0	c	0	0
f_1^0	a	0	0	0	a	b	0	b	0	a	0	0	0	a	b	0	b	0	0	b	c	b	0	0	c	0	0
f_2^0	a	0	0	a	0	0	0	b	c	0	a	b	0	a	b	b	0	0	0	b	0	0	b	0	c	0	0
$f_0 f_1^0 f_2^0$	a^3	0	0	0	0	0	0	b^3	0	0	0	0	0	0	b^3	0	0	0	0	b^3	0	0	0	0	c^3	0	0

Table 3: Function factorizations for $\mathbb{F}_4$-additive solution after LC_0 (rule 1)

x_0	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w	0	1	w
x_1	0	0	0	1	1	1	w	w	w	0	0	0	1	1	1	w	w	w	0	0	0	1	1	1	w	w	w
x_2	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	w	w	w	w	w	w	w	w	w
f_0	a	0	b	a	0	b	0	b	0	a	0	b	a	0	b	0	b	0	0	b	0	0	b	0	c	0	0
$\mathcal{I}_{f_0}$	1	0	1	1	0	1	0	1	0	1	0	1	1	0	1	0	1	0	0	1	0	0	1	0	1	0	0
f_1^0	d	0	0	0	0	e	0	e	0	d	0	0	0	0	e	0	e	0	0	d	0	0	0	0	e	0	0
f_2^0	d	0	0	d	0	0	0	d	0	0	0	e	0	0	e	0	0	0	0	e	0	0	e	0	e	0	0
$f_0 f_1^0 f_2^0$	ad^2	0	0	0	0	0	0	bed	0	0	0	0	0	0	be^2	0	0	0	0	bde	0	0	0	0	ce^2	0	0

Table 4: Function factorizations for $\mathbb{F}_4$-additive solution after LC_0 (rule 2)

exist. Moreover the maximum probability solution remains at $(0ww)$, where the probability has changed from ce^2 to c^3.

This LC update rule is interesting because we actually modify the global function whilst still encouraging convergence towards the same global solution - in this case $(0ww)$. So we have diversity of function. The only messages passed along edges are the soft information vectors during the sum-product (or max-product) algorithm, and the matrix row information. Specifically, if we do LC_i then we pass row i to all vertices currently connected to vertex i. It remains to specify probability profiles for 2, 3, 4, etc ... vertices. For vertices of degree 2 or 3 we can specify d and e, or a, b, and c, respectively. Similarly, for degree 4 vertices we may use 4 probabilities f, g, h, and e, where $f < g < h < e$.

Function update rule no. 2:
Instead of updating the function factors according to the new constraints, as specified by the new rows, another option is to do as follows. Consider, once again, that we do LC_0 on H_4 to obtain H_4^0, i.e. we obtain $\mathcal{G}^0$ from $\mathcal{G}$. Let $\mathcal{I}_{f_0}(x_0, x_1, x_2) = 1$ if $f_0(x_0, x_1, x_2) > 0$ and $\mathcal{I}_{f_0}(x_0, x_1, x_2) = 0$ if $f_0(x_0, x_1, x_2) = 0$. Then, referring to table 2, we can just make $f_1^0 = \mathcal{I}_{f_0} f_1$ and $f_2^0 = \mathcal{I}_{f_0} f_2$, giving table 2.1. This is realised by passing row 0 of H_4 and $\mathcal{I}_{f_0}$ to vertices 1 and 2. Rows 1 and 2 of H_4^0 are then computed at vertices 1 and 2 respectively. Row 1 is $(w^2 w^2 1)$ so vertex 1 knows that it is now connected to vertices 0 and 2 (as the first and third entries in $(w^2 w^2 1)$ are non-zero) so f_1^0 is taken over all three variables. Similarly for f_2^0.

This rule guarantees that the global function remains unchanged. But what if we do LC_0 again? Then we obtain $(\mathcal{G}^0)^0 = \mathcal{G}$ from $\mathcal{G}'$. This is realised by once again passing row 0 of H_4 and $\mathcal{I}_{f_0}$ to vertices 1 and 2. Rows 1 and 2 of $(H_4^0)^0 = H_4$ are then computed at vertices 1 and 2 respectively. Row 1 is $(1w0)$ so vertex 1 knows that it is now connected only to vertex 0 and not to 2 (as the first entry of $(1w0)$ is non-zero, but the third entry is zero). So $f_1' = (f_1^0)^0$ $(= f_1?)$ is only taken over variables 0 and 1. We use a special 'summary': $f_1'(x_0, x_1) = \max_{x_2}\{f_1^0(x_0, x_1, x_2)\}$. For instance, $f_1'(0,0) = \max\{f_1^0(0,0,0), f_1^0(0,0,1), f_1^0(0,0,w)\} = \max\{d, d, 0\} = d$, $f_1'(1,0) = \max\{f_1^0(1,0,0), f_1^0(1,0,1), f_1^0(1,0,w)\} = \max\{0, 0, d\} = d$,

and $f_1' = f_1 = (d, d, 0, 0, 0, e, e, e, 0)$. Similarly, $f_2' = f_2$ and the system reverts to that shown in table 2. Note that we did not make use of $\mathcal{I}_{f_0}$ so we shouldn't have sent it. So we should be able to improve the message-passing efficiency later, I think.

Of the two function update rules discussed above, update rule no 1 looks better as no (exponential size) function vectors need be passed along graph edges.

References