Shieh Yueh-Chang

FD-LCS Algorithm 說明

- 1 - FD-LCS Algorithm

```
FD_LCS( G, L )
    for all vertex v in G
            find "top", "oper", and "bottom" vertexes;
            Ru[v] = its resource-used;
    Initialization( top, bottom, A, R, S )
    if ( there is no feasible solution )
            return "No Feasible";
    do
            calculate_distribution( oper, A, R, Ru, p, q )
            calculate_total_force( G, tF );
            find the smallest tF[v, l];
            A[v] = l;
    `       R[v] = l;
            S[v] = 0;
    while ( there are vertexes not yet scheduled )
    return outcome;
```

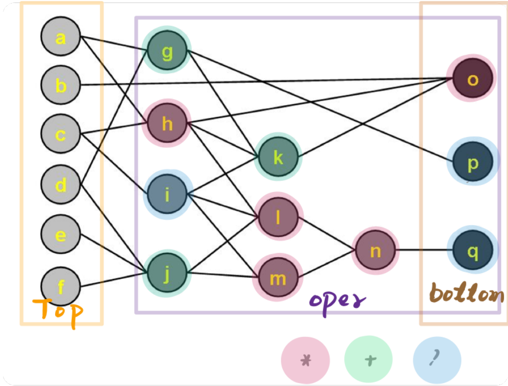


Figure 1. G is a directed graph obtained by parsing a BLIF file. Vertexes using resource "*" are colored in red. Vertexes using resource "+" are colored in green.
Vertexes using resource "!" are colored in blue.

Definition

G : graph
L : latency constraint
top : vertexes in G with in-degree of 0
oper : vertexes in G with in-degree > 0
bottom : vertexes in G with out-degree of 0
Ru[v] : resource used by vertex v in G, including { *, +, ' }
A[v] : arrive time of vertex v in G
R[v] : require time of vertex v in G
S[v] : slack of vertex v in G
p[v, l] : probability density of vertex v in G on cycle l
q[r, l] : distribution of resource r on cycle l
tF[r, l] : total force of resource r on cycle l

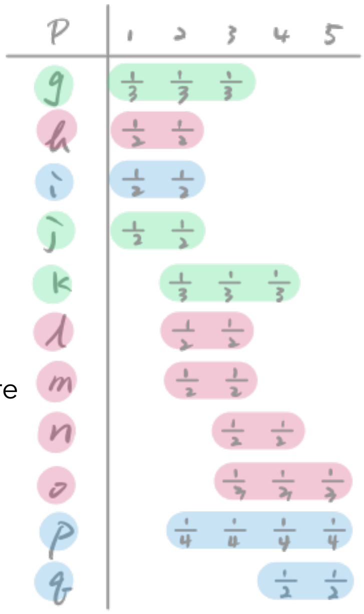


Figure 2. This table suggests informations such as arrive-time, require-time, slack, resource-used of a vertex, probability density. For example, a vertex "g" uses a resource "+", finding an arrive time on cycle 1, a require time on cycle 3, thus, a vertex G accounts for 1/3 probability density on cycle 1, 2, 3, and likewise are the other vertexes.

We obtain a graph G by parsing a BLIF file illustrated in figure 1. There are vertexes a, b, c, d, ..., p, q in graph G.

Firstly, find those vertexes belong to "top", "oper", "bottom" by traversing all vertexes in G. Then, initializing Ru[v] to the resource used by v, which can be obtained from the logic function of each vertexes.

A table is illustrated in figure 2. This table suggests several informations that this algorithm keeps. Firstly, p[v, l] is a two dimension array with the first dimension indexed by v, which is a vertex in G, and with the second dimension indexed by l, which is a # ranged from 1 to L, where L is the latency constraint. The value of p[v, l] stands for according probability density of vertex v over the cycles l. The color suggests the resource a vertex v used recorded in Ru[v]. The colored marks within each rows indicated arrive time, require time, and slack of the according vertex.

To obtain informations in the table, we initialize arrive time by traversing vertexes starting form "top" in breathe-first-search order; then, we initialize require time and slack by traversing vertexes starting form "bottom" in breathe-first-search order. The details are showed in algorithm: Initialization(top, bottom, A, R, S).

After initialize A, R, S, we can check whether there are some S[v] < 0, where v is a vertex in oper. Some S[v] < 0 suggests it is not feasible.

```
Initialization( top, bottom, A, R, S )
        push all nodes in "top" paired up with -1 into Q;
        while ( Q is not empty )
                pop element ( v, c ) from Q;
                if c > A[v]
                        A[v] = c;
                        for all successors s of v
                                push ( v, c + 1 ) into Q;
        push all nodes in "bottom" paired up with L - 1 into Q;
        while ( Q is not empty )
                pop element ( v, c ) from Q;
                if c < R[v]
                        R[v] = c;
                        S[v] = R[v] - A[v]
                        for all predecessors p of v
                                push ( v, c - 1 ) into Q;
```

Definition

Q : a FIFO queue whose elements are pairs of a vertex and an int
A[v] : arrive time of vertex v in G
R[v] : require time of vertex v in G
S[v] : slack of vertex v in G

Now it is feasible for sure.

We calculate each resource's distribution on each cycles. For example, a vertex g has a probability density of 1/3 scheduled on cycle 1, and a vertex j has a probability density of 1/2 scheduled on cycle 1 as well. Both of which use resource "+"; thus, there is a 1/3 + 1/2 distribution density of resource "+" on cycle 1. Likewise are other vertexes.

As an analogy to the physical formula: Force = constant * displacement, the mechanism for calculating "total force" is illustrated as follows.

Firstly, q[r, l], where r is a resource, l ranges from 1 to L, is viewed as the constant and p[v, l], where v is a vertex, l ranges from 1 to L, is viewed as the displacement. We define:

$$selfF[v,l] = \sum_{l \in \{A[v], A[v]+1, \ldots, R[v]\}} q[Ru[v],l] * p[v,l]$$

$$= q[Ru[v],l] - p[v,l] * \sum_{l \in \{A[v], A[v]+1, R[v]\}} q[R[v],l]$$

$$= q[Ru[v],l] - BaseF[v]$$

```
calculate_distribution( oper, A, R, Ru, p, q )
        for all vertex v in oper
                for i = A[v] to R[v]
                        q[Ru[v]] += p[v, i];
```

Definition

oper : vertexes in G with in-degree > 0
A[v] : arrive time of vertex v in G
R[v] : require time of vertex v in G
Ru[v] : resource used by vertex v in G, including { *, +, ‘ }
p[v, l] : probability density of vertex v in G on cycle l
q[r, l] : distribution of resource r on cycle l

q(r, l)	1	2	3	4	5
*	0.5	1.5	1.83	0.83	0.33
+	0.83	1.17	0.67	0.33	0.00
?	0.5	0.75	0.25	0.75	0.75

Figure 3. q[r, l] is a two dimension array with the first dimension indexed by r, a resource-used by vertexes, and with the second dimension indexed by l, a cycle ranged from 1 to L. Suggested in this figure, the resource "*" has distributions of 0.5, 1.5, 1.83, 0.83, 0.33 over cycle 1 to 5, respectively; the resource "+" has distributions of 0.83, 1.17, 0.67, 0.33, 0.00 over cycle 1 to 5, respectively; the resource "*" has distributions of 0.5, 0.75, 0.25, 0.75, 0.75

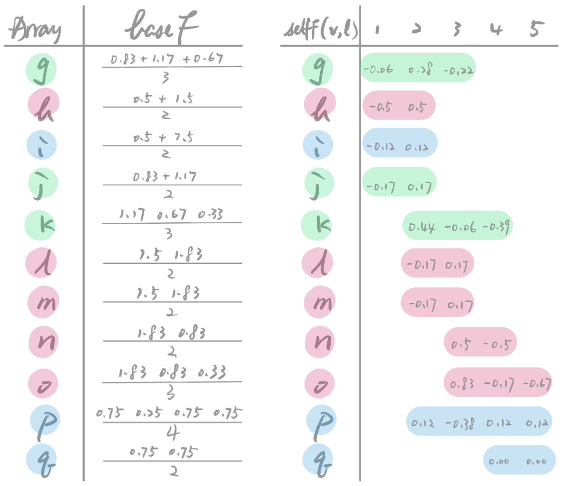


Figure 4. baseF[g] = (0.83 + 1.17 + 0.67) / 3; sefF[g, 1] = 0.83 - baseF[g] = -0.06; sefF[g, 2] = 1.17 - baseF[g] = 0.28; selfF[g, 3] = 0.67 - baseF[g] = -0.22, and so on. The greater negativity of self[v, l] suggests that v scheduled on cycle l is more desirable since it lower the distribution of the resource over that cycle.

And the calculation is demonstrated in figure 4. Furthermore, we define aF[v, l], for all v is a vertex in G, and for all l ranges A[v] to R[v], as the force generated by moving R[v] forward to l; likewise, we define dF[v, l], for all v is a vertex in G, and for all l ranges A[v] to R[v], as the force generated by postponing A[v] to l.

$$aF[v,l] = \frac{\sum_{i \in \{A[v], \ldots, l\}} q[Ru[v], i]}{l - A[v] + 1} - baseF[v]$$

$$dF[v,l] = \frac{\sum_{i \in \{l, \ldots, R[v]\}} q[Ru[v], i]}{R[v] - l + 1} - baseF[v]$$

Then, we define pF[v, l], for all v is a vertex in G, and for all l ranges A[v] to R[v], as the sum of aF[p, k], where p belongs to ancestors of v, and where k is the new R[p] after scheduling v on cycle l. Again, we define dF[v, l], for all v is a vertex in G, and for all l ranges A[v] to R[v], as the sum of dF[s, k], where s belongs to descendants of v, and where k is the new A[p] after scheduling v on cycle l. Now we define tF[v, l] = self[v, l] + pF[v, l] +sF[v, l], for all vertex v in G, for all i ranges form 1 to L.

$$pF[v,l] = \sum_{p \in ancestors\ of\ v} aF[p,k],\ where\ k\ is\ the\ new\ R[p]\ after\ scheduling\ v\ on\ cycle\ l$$

$$sF[v,l] = \sum_{s \in descendants\ of\ v} dF[s,k],\ where\ k\ is\ the\ new\ A[s]\ after\ scheduling\ v\ on\ cycle\ l$$

$$tF[v,l] = self[v,l] + pF[v,l] + sF[v,l]$$

```
calculate_total_force( oper, A, R, aF, dF, pF, sF, selfF, tF )
    for all vertex v in oper
        for i = A[v] to R[v]
            baseF[ v ] += distribution[ Ru[v] ][ i ];
        baseF[v] /= S[v] + 1;
    for all vertex v in oper
        selfF[v, A[v]] = aF[v, A[v]] = q[Ru[v], A[v]] - baseF[v];
        selfF[v, R[v]] = dF[v, R[v]] = q[Ru[v], R[v]] - baseF[v];
        for i =  A[v] + 1 to R[v] - 1
            selfF[v][i] = q[Ru[v], i] - baseF[v];
            for j = A[v] to i
                aF[v, i] += q[Ru[v], j];
            aF[v, i] = aF[v, i] / ( i - A[v] + 1 ) - baseF[v];
            for j = i to R[v]
                dF[v, i] += q[Ru[v], j];
            dF[v, i] = dF[v, i] / ( R[v] - i + 1 ) - baseF[v];
    for all vertex v in oper
        for l = A[v] to R[v]
            for all predecessor p of v
                k = R[p] that changed when v scheduled on cycle l;
                pF[v, l] += aF[ p, k];
            for all descendant s of v
                k =  A[s] that changed when v scheduled on cycle l;
                sF[v, l] += dF[ s, k];
    for all vertex v in oper
        for l = A[v] to R[v]
            tF[v, l] = self[v, l] + pF[v, l] +sF[v, l]
```

Definition

oper : vertexes in G with in-degree > 0
A[v] : arrive time of vertex v
R[v] : require time of vertex v
S[v] : slack of vertex v
tF[r, l] : total force of resource r on cycle l
pF[r, l] : predecessor force of resource r on cycle l
sF[r, l] : successor force of resource r on cycle l
selfF[r, L] : self force of resource r on cycle l

- 2 - Experimental result

Comparing several benchmarks, starting from using the minimum latency constraints and incrementing it by 20 for each test. As the result suggests, MR-LCS uses less gates as the latency constraint given become larger; and thus, using larger latency constraint improves the result. However, FD-LCS uses less gates only when the latency constraint given is tight; and uses more gates when latency constraint become greater. FD-LCS has a better result when the latency constraint given is tight but has a worse result when the latency constraint given is large.

To conclude, when using tight latency is needed, it is best to use FD-LCS; However, it suggests that using MR-LCS is a better choice when latency constraint given is large.

aoi_9symml.blif		AND	OR	NOT
11	MR	32	15	3
	FD	17	9	6
31	MR	18	12	1
	FD	7	6	5
51	MR	38	1	1
	FD	17	9	6
71	MR	18	1	1
	FD	17	9	6
91	MR	17	1	1
	FD	17	9	6

aoi_cht.blif		AND	OR	NOT
7	MR	68	35	1
	FD	18	14	3
27	MR	52	15	1
	FD	18	8	1
47	MR	32	29	1
	FD	18	8	1
67	MR	12	9	1
	FD	18	8	1
87	MR	26	1	1
	FD	18	8	1

aoi_t481.blif		AND	OR	NOT
9	MR	219	17	13
	FD	153	16	15
29	MR	120	11	1
	FD	59	13	13
49	MR	100	10	1
	FD	153	16	15
69	MR	80	6	1
	FD	153	16	15
89	MR	60	1	1
	FD	153	16	15

aoi_alu4.blif		AND	OR	NOT
37	MR	13	15	5
	FD	7	11	7
57	MR	4	10	2
	FD	7	8	4
77	MR	3	11	7
	FD	7	9	5
97	MR	13	5	6
	FD	7	10	8
117	MR	7	6	1
	FD	7	10	8

oi_cm138a.bl		AND	OR	NOT
3	MR	1	8	3
	FD	1	8	2
23	MR	1	1	1
	FD	1	1	1
43	MR	1	1	1
	FD	1	1	1
63	MR	1	1	1
	FD	1	1	1
83	MR	1	1	1
	FD	1	1	1

aoi_x2.blif		AND	OR	NOT
5	MR	7	6	9
	FD	5	5	7
25	MR	1	1	1
	FD	4	3	3
45	MR	1	1	1
	FD	4	3	3
65	MR	1	1	1
	FD	4	3	3
85	MR	1	1	1
	FD	4	3	3

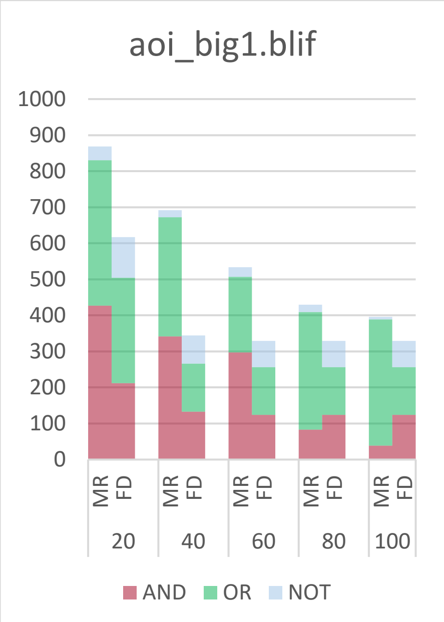
aoi_big1.blif		AND	OR	NOT
20	MR	427	403	39
	FD	212	292	113
40	MR	342	330	20
	FD	133	133	78
60	MR	297	210	27
	FD	124	132	73
80	MR	83	326	21
	FD	124	132	73
100	MR	39	350	7
	FD	124	132	73

aoi_des.blif		AND	OR	NOT
19	MR	90	239	36
	FD	140	128	44
39	MR	220	101	26
	FD	140	128	32
59	MR	30	35	22
	FD	140	128	32
79	MR	184	44	12
	FD	140	128	32
99	MR	20	23	8
	FD	140	128	32

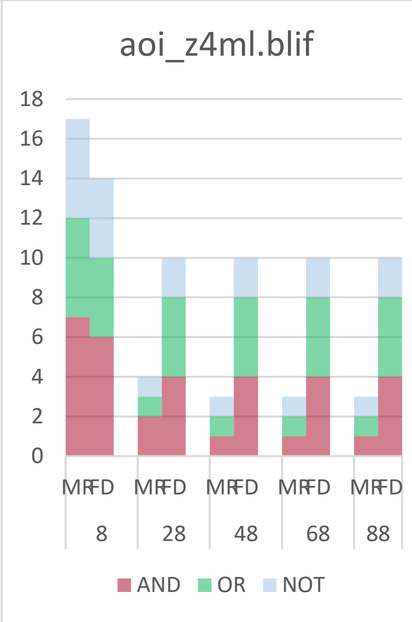
aoi_z4ml.blif		AND	OR	NOT
8	MR	7	5	5
	FD	6	4	4
28	MR	2	1	1
	FD	4	4	2
48	MR	1	1	1
	FD	4	4	2
68	MR	1	1	1
	FD	4	4	2
88	MR	1	1	1
	FD	4	4	2

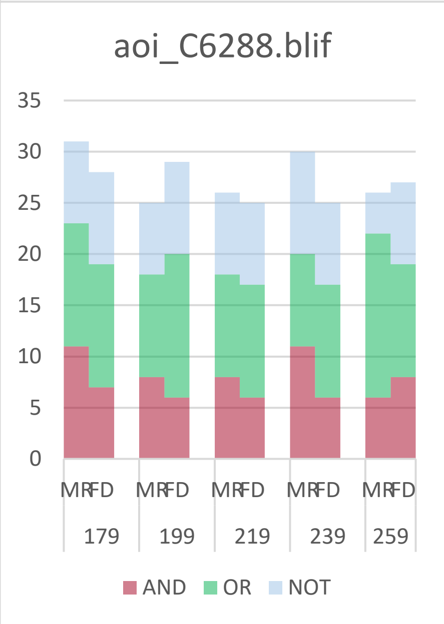
aoi_C6288.blif		AND	OR	NOT
179	MR	11	12	8
	FD	7	12	9
199	MR	8	10	7
	FD	6	14	9
219	MR	8	10	8
	FD	6	11	8
239	MR	11	9	10
	FD	6	11	8
259	MR	6	16	4
	FD	8	11	8

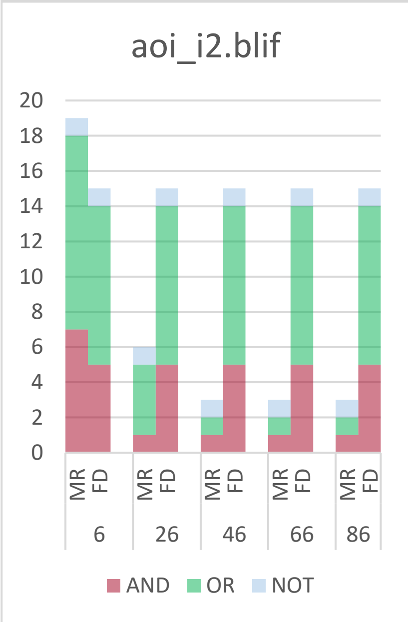
aoi_i2.blif		AND	OR	NOT
6	MR	7	11	1
	FD	5	9	1
26	MR	1	4	1
	FD	5	9	1
46	MR	1	1	1
	FD	5	9	1
66	MR	1	1	1
	FD	5	9	1
86	MR	1	1	1
	FD	5	9	1


aoi_big1.blif
1000
900
800
700
600
500
400
300
200
100
0
MR
FD
20
40
60
80
100
AND
OR
NOT


aoi_des.blif
400
350
300
250
200
150
100
50
0
MR
FD
19
39
59
79
99
AND
OR
NOT


aoi_z4ml.blif
18
16
14
12
10
8
6
4
2
0
MR
FD
8
28
48
68
88
AND
OR
NOT


aoi_C6288.blif
35
30
25
20
15
10
5
0
MR
FD
179
199
219
239
259
AND
OR
NOT


aoi_i2.blif
20
18
16
14
12
10
8
6
4
2
0
MR
FD
6
26
46
66
86
AND
OR
NOT