볼록 이분할 그래프에서 최대 매칭을 찾기 위한 개선된 Boolean 회로

박은희° 박근수
{ehpark°, kpark}@theory.snu.ac.kr

An improved Boolean Circuit for Maximum Matching in Convex Bipartite Graphs

Eunhui Park°, Kunsoo Park
School of Computer Science and Engineering, Seoul National University

요 약

Boolean 회로는 parallel 알고리즘을 위한 단순하면서도 실제적인 모델이다. Chung & Lee은 Boolean 회로 모델에서 볼록 이분할 그래프를 위한 최대 매칭을 찾는 $O(log^2n+logn \cdot loglogn \cdot logb)$depth와 $O(bn^3)$ size의 알고리즘을 제시하였다. 본 논문에서는 prefix computaion 및 ASCEND, odd-even-merge의 방법을 이용하여 이를 개선한 $O(log^2n \cdot logb)$ depth, $O(bn^2logn)$ size의 최대 매칭 알고리즘을 제시한다.

1. 서론

볼록 이분할 그래프란 vertex들의 집합 A, B로 된 이분할 그래프에서, 집합 A의 모든 vertex에 대하여, 각 vertex의 neighbor들이 연속되는 특성을 갖는 것을 말한다. 본 논문에서는 Boolean 회로 모델[1]에서 볼록 이분할 그래프의 최대 매칭을 찾는 $O(log^2n \cdot logb)$ depth, $O(bn^2logn)$ size의 Boolean 회로를 제시한다.[1] 이미 Boolean 회로에서 볼록 이분할 그래프에서 최대 매칭을 찾는 알고리즘은 Chung & Lee[2]에 의해서 제시되었다. 그러나 이 알고리즘의 depth와 size는 각각 $O(log^2n+logn \cdot loglogn \cdot logb)$, $O(bn^3)$로 size가 크다는 단점을 가지고 있다. 본 논문에서는 prefix computation 및 ASCEND, odd-even merge등의 방법을 이용하여 이 size의 단점을 개선하였다. 알고리즘의 효율은 Dekel & Sahni[3]를 따른다.

2장에서는 Boolean 회로와 볼록 이분할 그래프의 parallel 매칭에 대해서 간단히 알아본다. 3장에서는 개선된 Boolean 회로를 제시하며 4장에서 결론을 맺는다.

2. Preliminary

Boolean 회로는 input과 output, logic gate, wire로 구성된다. 여기서 input, output, logic gate들은 wire에 의해 연결된다. Boolean 회로는 프로그래밍 언어로 쓸 수 있으며, 또한 이 언어를 바로 논리회로로 구현할 수 있다는 점에서 다른 parallel model에 비하여 단순하면서도 실제적이다. Boolean 회로에서의 복잡도는 depth와 size로 표현된다. depth는 input에서 output까지의 가장 긴 path이고, size는 사용되는 input, output, logic gate 개수들의 합이다. 복잡한 Boolean 회로는 기본적인 logic gate (AND,OR,NOT)와 multiplexer(MUX), demultiplexer(DEMUX), adder(ADDER)와 같은 building block을 사용하여 표현한다.

볼록 이분할 그래프 G는 triple (A, B, E)이다. A와 B는 vertex들의 disjoint 집합이며, E는 edge들의 집합이다. E는 다음의 속성을 가진다. $(a_i, b_j) \in E$이고 $(a_i, b_{j+k}) \in E$이면 $(a_i, b_{j+q}) \in E$, $1 \le q \le k$이다. 볼록 이분할 그래프에서의 매칭이란 어떠한 두 edge도 같은 end point를 포함하지 않는 E의 부분집합이다. 최대 매칭은 가장 많은 edge를 포함하는 매칭을 의미한다. 볼록 이분할 그래프 $G=(A, B, E)$, $A=\{a_1, a_2, ..., a_n\}$, $B=\{b_1, b_2, ..., b_m\}$은 다음과 같은 triple로 표현할 수 있다.

$$T=\{(i, s_i, h_i) \mid 1 \le i \le n\}$$
$$s_i = min\{j \mid (a_i, b_j) \in E\}$$
$$h_i = max\{j \mid (a_i, b_j) \in E\}$$

1) n은 입력의 크기이고 b는 입력을 나타내기 위한 비트수이다.

3. 개선된 boolean 회로

이 절에서는 n개의 triple 형태로 주어진 볼록 이분할 그래프에서 최대 매칭을 찾는 개선된 Boolean 회로를 제시한다. 우선 Dekel & Sahni의 binary 계산 트리[3]를 만들기 위하여 입력을 s_i 값에 따라 sorting하고 같은 s_i값들을 갖는 입력으로 leaf를 구성한다. 그 다음에 Dekel & Sahni[3]에서와 같이 첫 번째 pass에서는 계산 트리의 leaf에서 root까지 올라가면서 각 노드에서 매칭 집합을 찾고, 이들의 부모 노드에서 두 자식 노드의 매칭 집합들을 merge한 것으로 다시 매칭 집합을 찾는 작업을 수행한다. 두 번째 pass에서는 root에서 leaf까지 내려오면서 원래의 볼록 이분할 그래프에서의 최대 매칭을 찾기 위하여 각 매칭 집합들을 보정한 후 leaf에서 최대 매칭 집합을 출력한다. 각 과정은 다음과 같다.

3.1 계산 트리를 위한 leaf의 구성

최대 매칭을 찾기 위한 계산 트리를 만들기 위하여, 먼저 입력들을 s_i값에 따라서 sorting을 하고 그 다음에 같은 s_i값을 갖는 것들로 분할하여 트리의 leaf를 구성한다.

Sorting은 Park & Park[1]에서 제안된 sorting circuit에 따라 수행된다. 이 때 같은 s_i에 대해서는 h_i에 따라 sorting되도록 다음과 같은 Boolean expression으로 비교하여 수행한다[2].

$$(s_i > s_j) \vee (s_i = = s_j \wedge h_i > h_j) \vee (s_i = = s_j \wedge h_i = = h_j \wedge i > j)$$

이제, 각 leaf를 같은 s_i값을 갖는 입력들로 구성한다. 입력들을 해당 leaf로 보내기 위하여 $diff$라는 것을 이용하여 각 입력마다 해당 leaf 번호, $leafn$을 정한다. $diff$는 다음과 같이 정의한다.

$diff_1 = 1$

$diff_i = \neg\ (s_i == s_{i-1})$

$leafn$은 $diff$값을 모두 더하여 찾는다.[2)]

$leafn_i = diff_1 + diff_2 + \ldots\ldots + diff_i$

표1은 sorting된 입력과 그 $diff$와 $leafn$의 예를 보여준다. 각 입력은 이 $leafn-1$을 selector로 DEMUX[1]를 이용하여 해당 leaf로 보내진다. 따라서 같은 leaf에 해당되는 입력들은 DEMUX의 같은 위치에 나타나게 된다(그림 1).

표 1 $diff$와 $leafn$

$index$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
i	7	4	10	11	1	5	8	2	13	3	6	14	16	15	12	9
s	1	1	1	5	5	8	8	8	10	12	12	12	12	12	15	15
h	3	4	5	7	9	9	9	11	11	13	13	13	15	17	16	17
$diff$	1	0	0	1	0	1	0	0	1	1	0	0	0	0	1	0
$leafn$	1	1	1	2	2	3	3	3	4	5	5	5	5	5	6	6

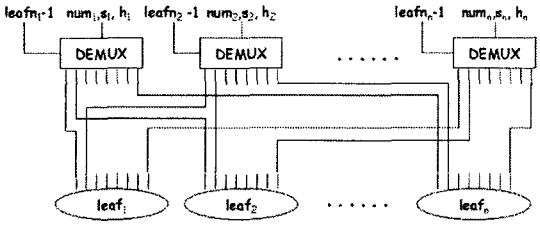

그림 1 입력의 분할

표 2 $leaf_3$의 구성

$index$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
i	0	0	0	0	0	5	8	2	0	0	0	0	0	0	0	0
s	0	0	0	0	0	8	8	8	0	0	0	0	0	0	0	0
h	0	0	0	0	0	9	9	11	0	0	0	0	0	0	0	0

입력들을 각 leaf로 분할하고 난 후에는, 각 leaf에서 실제 값이 포함된 것이 맨 앞의 위치에 나타나도록 shift시킨다. 예를 들어, 예제의 $leaf_3$은 표2와 같이 1번~5번 원소들에 모두 0을 포함한다. 이 0들은 첫 번째 pass에서 merge가 수행된 후에 각 노드의 크기가 커지지 않도록 없애주어야 한다. 이것은 cyclic shift를 수행하여 뒤로 보낼 수 있다. 몇 칸이 shift되어야 하는지는 0이 아닌 값이 처음으로 나타나는 위치 $index$에서 -1을 하면 된다. 이 때, 0인지 아닌지를 비교하는 것은 Park & Park[1]의 COMP를 이용한다.

Cyclic shift는 Preparata[4]의 ASCEND 방법을 적용하여 구현할 수 있다. 편의상 $n=2^k$개의 입력을 s만큼 shift시킨다고 하자. 먼저 s의 최하위 비트인 s_0부터 검사하여 이 비트가 1인 경우 입력의 짝수 번째 자리에 있는 것은 left group으로 홀수 번째 자리에 있는 것은 right group으로 나눈다. 0인 경우 홀수 번째 자리에 있는 입력들을 left group으로 짝수 번째 자리에 있는 입력들을 right gorup으로 나눈다. 그 다음에 left group과 right group 각각에 대해 s>>2만큼 cyclic shift를 수행한다. 그 결과 각각을 left_out, right_out이라 한다면 이들은 다음과 같이 output으로 merge된다.

$output_{2i-1} = left\_out_i \quad$ for $i = 1,2,\ldots,n/2$

$output_{2i} = (\neg\ s_0 \wedge right\_out_i) \vee (s_0 \wedge right\_out_{i+1})$

for $i = 1,2,\ldots,n/2-1$

$output_n = (\neg\ s_0 \wedge right\_out_{n/2}) \vee (s_0 \wedge right\_out_1)$

그림2는 1,2,3, ...,16을 5만큼 cyclic shift시키는 과정을 보여준다.

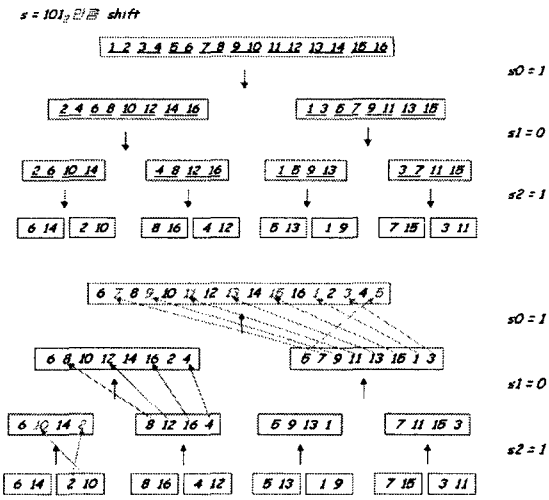

그림 2 cyclic shift

3.2 첫 번째 pass

첫 번째 pass에서는 계산 트리의 leaf에서 root까지 올라가면서 각 노드에서 최대 매칭 집합을 찾고 부모 노드에서 이를 merge한 것으로 다시 매칭 집합을 찾는 작업을 수행한다.

노드에서 최대 매칭 집합을 찾기에 앞서 먼저 각 노드에서 입력들이 집합 B로 매칭되는 범위 $[u,v]$를 찾는다. Sorting된 입력에서 서로 다른 s_i들을 $R_1, R_2, \ldots, R_k$라고 하면 $[R_i, R_{i+1}]$이 $leaf_i$의 $[u,v]$이다. 여기서, k는 leaf의 개수이며, R_{k+1}는 $max\{h_i\}+1$로 한다[3]. R_k결정을 위한 Boolean 회로는 leaf를 구성할 때와 마찬가지로 $leafn-1$을 selector로 하여 DEMUX를 이용하는데, 단 DEMUX의 입력은 서로 다른 s_i만을 찾기 위해 $s_i \wedge diff_i$로 한다. R_i는 이 DEMUX들의 i번째 출력들을 OR연산하여 얻는다. R_{k+1}는 h_j를 binary tree로 구한 후 $leafn$을 selector로 하여 DEMUX를 이용하여 구한다. 내부노드의 $[u,v]$의 경우는 left child의 u와 right child의 v로 결정된다.

이제 노드의 최대 매칭 집합을 찾는다. 이는 FEAS라는 알고리즘을 수행하여 찾는다. FEAS는 h_i에 따라 sorting된 입력에 대하여 맨 처음 입력부터 $u, u+1, u+2, \ldots, v$를 매칭시킨다. 이 때, 입력의 h_i가 매칭될 값보다 크거나 v보다 큰 경우 그 입력은 매칭되지 않는다. FEAS의 sequential version은 다음과 같다.

$j \leftarrow u$

for $i \leftarrow 1$ *to* n

  $j > v$ *return*

  $j \le h_i\ j \leftarrow j+1,\ MAT(i) \leftarrow 1$ *//MAT: matching*

FEAS의 parallel version은 $DONE(i)$을 이용한다[5].

2) 여기서, $diff$를 prefix computation방법을 이용하여 찾을 수도 있다. 그러나, prefix computation방법을 사용하면 size는 줄어들지만 시간이 늘어나게 되고 직접 더하여 찾는 방법이 전체 복잡도에 영향을 주지 않으므로 본 논문에서는 직접 더하는 방법으로 한다.

$DONE(i)$은 입력i가 매칭될 수 있는 집합B의 번호를 의미하는 것으로, $DONE(i)$과 $DONE(i-1)$이 다를 경우에만 입력i가 실제로 노드의 매칭 집합에 포함된다. $DONE(i)$은 다음과 같다.

$$DONE(i)=min_{0\le j\le i}\{h'_j+i-j\}=i+min_{0\le j\le i}\{h'_j-j\}, 1\le i\le n$$

이 때, $DONE(i)$이 노드의 v값 이상이 되지 않도록 $h'_j=max\{h_j, v\}$로 한다[2]. $DONE(i)$에서 필요한 연산은 ADD와 최소값을 찾는 min연산으로, ADD연산은 Park & Park[1]의 ADD 회로를 이용하고, min은 prefix computation방법[6]을 적용하여 수행된다. 편의상 입력의 개수가 2^k라 하면, min함수의 Boolean 회로 코드는 그림3과 같다.

```
MIN generic(n, b)
in: IN(1..n)(0..b-1)
out: OUT(1..n)(0..b-1)
if(n=2) {
begin
  pfor i from 0 to b-1 {
    OUT(1)(i) ← IN(1)(i); }
  LE generic(b)(IN(1)(0..b-1), IN(2)(0..b-1), OUT(2)(0..b-1));
end }
else {
signal: TIN(1..n/2)(0..b-1), TOUT(0..n/2)(0..b-1)
begin
  pfor i from 1 to n/2 {
    LE generic(b)(IN(2i-1)(0..b-1), IN(2i)(0..b-1), TIN(i)(0..b-1));}
  MIN generic(n/2, b)(TIN(1..n/2)(0..b-1), TOUT(1..n/2)(0..b-1));
  TOUT(0)(0..b-1) ← 0;
  pfor i from 1 to n/2 {
    LE generic(b)(TOUT(i-1)(0..b-1), IN(2i-1)(0..b-1),
                  OUT(2i-1)(0..b-1));
    pfor j from 0 to b-1 {
      OUT(2i) ← TOUT(i); } }
end }
LE generic(b)
in: A(0,b-1), B(0..b-1);
out: C(0,b-1);
signal: T(0..1);
begin
COMP generic(┌logb┐, A(0..b-1), B(0..b-1), T(0), T(1));
pfor i from 0 to b-1 {
  C(i) ← (NOT T(1) AND A(i)) OR (T(1) AND B(i));}
end
```

그림 3 Min의 Boolean 회로 코드

FEAS를 수행하여 최대 매칭 집합을 찾은 후에 부모노드에서는 이들의 merge를 수행한다. 이미 sorting된 두 리스트의 merge는 odd-even merge[7]방법을 이용한다. Boolean 회로의 특성상, 계산 트리의 모든 노드는 표1의 $leaf_3$과 같이 실제 입력 값 이외에 0을 포함하고 있고 크기는 초기 입력의 개수인 n이 된다. 따라서 merge의 모든 두 입력들도 이러한 형태로, merge를 수행한 결과는 2n이 되며, 실제 입력 값은 n개를 넘지 않으므로 출력에서 뒤의 n개는 0이 된다. 따라서 merge를 수행한 후 뒤의 n개의 결과는 잘라버린다. merge를 수행하기 전에 입력 내에서 실제 값들이 $index$ 1에서 시작하지 않는 경우 3.1의 shift를 수행한 후에 merge를 한다.

3.3 두 번째 pass

두 번째 pass에서는 root에서 leaf로 내려가면서 단순히 비교와 merge를 수행하며 진행하므로 3.2의 merge를 이용한다.

3.4 복잡도

개선된 블록 이분할 그래프에서 최대 매칭은 $O(log^2n \cdot logb)$ depth와 $O(bn^2 \cdot logn)$ size의 복잡도로 각 단계의 복잡도는 다음과 같다.

먼저 트리의 구성 단계의 복잡도는 $O(logn+logb)$ depth, $O(bn^2)$ size가 된다. 세부 단계는 다음과 같다. sorting의 경우 depth와 size는 각각 $O(logn+logb)$와 $O(bn^2)$[1]이다. $diff$는 $O(logb)$depth와 $O(bn)$size, 각 입력의 $leafn$결정은 $O(logn)$ depth와 $O(n^2)$size, 입력을 각 leaf로 보내는 것은 $O(logn)$ depth와 $O(bn^2)$size가 된다. Cyclic shift에서 재귀함수의 depth은 Boolean 회로의 특성상, 최대 shift 비트수인 $logn$이 되므로 전체 cyclic shift 과정은 $O(logn)$depth, $O(bnlogn)$size가 된다.

첫 번째 pass와 두 번째 pass는 모두 $O(log^2n \cdot logb)$depth, $O(bn^2 \cdot logn)$size이다. 우선 첫 번째 pass에서 R을 찾는 것은 $O(logn)$depth와 $O(bn^2)$size가 되고, minimum의 계산은 prefix computation의 트리 depth가 $logn$, 각 레벨의 입력의 개수는 전 단계의 반이므로 $O(logn \cdot logb)$depth, $O(bn)$size가 된다. merge는 첫 번째 pass와 두 번째 pass에서 모두 쓰이는 것으로 재귀함수의 depth는 $logn$, 각 레벨에서 연산 수는 $O(n)$이므로 $O(logn \cdot logb)$depth, $O(bnlogn)$size가 된다. 이 minimum과 merge는 최대 매칭을 찾는 계산 트리의 각 노드에서 수행되는 작업으로 계산 트리의 높이가 $logn$이므로 두 pass의 전체 depth와 size는 $O(log^2n \cdot logb)$와 $O(bn^2 \cdot logn)$이다.

4. 결론

본 논문에서는 $O(log^2n \cdot logb)$depth와 $O(bn^2 \cdot logn)$ size의 블록 이분할 그래프의 최대 매칭 알고리즘을 제시하였다. 이는 prefix computation을 적용하여 minmum을 효율적으로 계산하였으며, shift와 odd-even merge를 사용하여 Chung & Lee의 알고리즘에서 두 pass상의 sorting을 없앰으로서 개선된 결과이다. Boolean 회로의 특성을 고려할 때, 이 알고리즘은 최대 매칭을 구하는 효율적인 Boolean 회로라 할 수 있다.

참고문헌

[1] K. Park, H. Park, W.C. Jeun and S. Ha, Boolean Circuit Programming: A New Paradigm to Design Parallel Algorithms, 15th Australasian Workshop on Combinatorial Algorithms (AWOCA), pp.28-39, 2004.

[2] Y. Chung and S. Lee, A Boolean Circuit for Finding a Maximum Matching in a Convex Bipartite Graph, 2005 Korea-Japan Joint Workshop on Algorithms and Computation (WAAC), pp.158-165, 2005.

[3] E. Dekel, and S. Sahni, A Parallel Matching Algorithm for Convex Bipartite Graphs and Applicaitons to Scheduling, Journal Of Parallel And Distributed Computing 1, pp. 185-205, 1984.

[4] F. P. Preparata and J. Vuillemin, The Cube-Connected Cycles: A Versatile Network for Parallel Computation, Communications of the ACM, 24, 5, pp. 300-309, 1981.

[5] E. Dekel, and S. Sahni, Parallel scheduling algorithms, Operation Research, 31, 1, pp.24-49, 1983.

[6] R. E. Lander and M. J. Fischer, Parallel Prefix Computation, Journal of the ACM, 27, 4, pp.831-838, 1980.

[7] K. E. Batcher, Sorting Networks and their Applications, Proc. of the 1968 Spring Joint Computing Conference, 32, pp.307-314, 1968.