$O(\log n)$의 병렬 시간이 소요되는 Solid Grid 그래프를 위한 Depth-First Search 알고리즘

(An $O(\log n)$ Parallel-Time Depth-First Search Algorithm for Solid Grid Graphs)

허 준 호 † R. S. Ramakrishna††
(Jun-Ho Her) (R. S. Ramakrishna)

요 약 본 논문은 평면 그래프를 위한 병렬 depth-first search (DFS) 알고리즘 [SIAM J. Comput., 19 (1990) 678-704]을 비 평면일 (non-planar) 수 있는 grid 그래프의 한 종류인 solid grid 그래프에 대해서도 수행 가능하도록 확장된 알고리즘을 제안한다. 제안 알고리즘은 Priority PRAM 모델에서 $O(n/\sqrt{\log n})$개의 프로세서로 수행했을 때 $O(\log n)$의 병렬 시간이 소요된다. 우리의 지식으로, 이는 비 평면 그래프를 위한 첫 번째 결정적 NC (deterministic NC) 알고리즘이다.

키워드 : 병렬 알고리즘, 그래프 알고리즘, (비)평면 그래프

Abstract We extend a parallel depth-first search (DFS) algorithm for planar graphs to deal with (non-planar) solid grid graphs, a subclass of non-planar grid graphs. The proposed algorithm takes time $O(\log n)$ with $O(n/\sqrt{\log n})$ processors in Priority PRAM model. In our knowledge, this is the first deterministic NC algorithm for a non-planar graph class.

Key words : parallel algorithm, graph algorithm, (non)planar graph

1. 서 론

그래프에 대한 DFS 트리는 많은 다른 그래프 문제들을 해결하기 위한 수단으로 활용되곤 했다[1,2]. 뿐만 아니라, DFS 트리는 인공지능 분야에서도 유용한 도구로 활용된다[3]. 하지만 [4,5]에서 일반적으로 DFS 문제는 병렬화가 근본적으로 힘들다는 추측이 제기되었다. 일반 비 방향성 그래프[6]와 일반 방향성 그래프[7]에 대해서 RNC(randomized NC) 알고리즘 제안되었다. 또한 비 방향성 평면 그래프에 대해서 몇몇 결정적 NC (deterministic NC) 알고리즘이 제안되었다; 표 1은 이를 요약적으로 보이고 있다. 표 1의 결과들은 다소 오래 전에 발표된 것이지만, 비 평면인 다른 그래프에 대해서는 이렇다 할 새로운 결과가 없었다.

본 논문에서는 solid grid 그래프라고 하는 특정 비 평면 그래프에 대한 병렬 DFS 알고리즘을 고려한다. Grid 그래프는 무한 이차원 정수 격자상의 유한한 점유도(node-induced) 그래프로 정의 된다. Solid grid 그래프는 그래프 내부가 꽉 차 있어 내부의 격자에는 항상 edge가 존재하는 grid 그래프의 한 종류이다(제세한 사항은 부록 1을 참조하기 바람). 제안 알고리즘은 Hagerup의 평면 그래프 알고리즘(이후 Par_Planar_DFS라 칭함)을 서브루틴으로 활용하는 데 주요 아이디어는 저자들의 입출력 효율적인 알고리즘[8]을 바탕으로 한다.

본 논문은 다음과 같이 구성된다. 2절에서는 Hagerup의 알고리즘을 간단히 살펴보고, 3절에서는 제안 알고리즘에 대해서 기술하며, 마지막으로 4절에서는 향후 과제를 포함해서 결론을 맺는다. 본 논문에서 사용되는 기본적인 개념은 부록 1에 기술한다.

표 1 평면 그래프를 위한 병렬 DFS 알고리즘

PRAM Model	Authors	Parallel Time	Proc.
CREW	J. R. Smith [9]	$O(\log^3 n)$	$O(n^4)$
	Goldberg et al. [10]	$O(\log^3 n \cdot \log^* n)$	$O(n)$
CRCW	G. E. Shannon [11]	$O(\log^2 n)$	$O(n)$
	T. Hagerup [12]	$O(\log n)$	$O(n^3)$
	M-Y Kao et al. [13]	$O(\log^2 n)$	$O(n/\log n)$

† 학생회원 : 광주과학기술원 정보통신공학과
jhher@gist.ac.kr
†† 비 회 원 : 광주과학기술원(GIST), 정보통신공학과 교수
rsr@gist.ac.kr
논문접수 : 2006년 1월 15일
심사완료 : 2006년 4월 7일

2. Par_Planar_DFS

Par_Planar_DFS [14]는 제안 알고리즘에서 한 서브루틴의 역할을 하므로 이에 대해 되짚어 볼 필요가 있다. 이 알고리즘은 개괄적으로 다음과 같이 동작한다. 첫째, 평면 그래프의 각 face들을 루트가 포함된 face를 첫 번째 층으로 하여 주어지 평면 그래프를 여러 단계의 층으로 나누어 고려한다. 둘째로, 각 단계의 층으로부터 바깥 평면 그래프(outerplanar graph)들을 얻는다. 각 바깥 평면 그래프의 DFS-tree들을 구한 후 최종적으로 이들 DFS-tree를 합하면 주어진 평면 그래프의 DFS-tree를 얻는다. 첫 째 단계를 위해서는 G의 face-on-vertex 그래프인 G_F에 대해서 breadth-first search를 수행하게 된다. 그래프 G_F의 vertex 집합은 우선 G의 모든 vertex들을 포함하고 G의 각 face f에 대응되는 한 vertex f^*라 할 때 이러한 vertex들 역시 포함된다. 그리고 edge 집합은 G의 face인 f의 경계에 위치 한 vertex v에 대해서 모든 가능한 edge $\{v, f^*\}$들로 이루어진다. 그림 1은 이와 관련된 예를 보인다.

다음 정리는 이 알고리즘의 주요 결과를 나타낸다.

정리 1 (Hagerup[12]). *Suppose that time $T(n) \geq \log n$ and $p(n)$ processors suffice to construct a planar embedding of a planar graph on n vertices and to compute a BFS tree with a given root of a planar graph on $3n$ vertices. Then, given an undirected connected planar graph $G=(V,E)$ on n vertices and a vertex $r \in V$, a DFS tree of G rooted ar r can be computed in time $O(T(n))$ by a Priority PRAM with $p(n)$ processors.*

3. 제안 알고리즘

제안 알고리즘은 크게 다음과 같은 세 단계로 동작한다; 첫 단계에서는, 적절한 edge contraction을 수행하여 주어진 solid grid 그래프 G를 임베드 된 평면 그래프(embedded planar graph) G'로 변형한다(3.1 절). 두 번째 단계에서는, G'의 한 DFS 트리 T'를 형성하기 위해 Par_Planar_DFS를 적용한다(3.2 절). 마지막 단계에서는, 이 전 단계의 DFS 트리로부터 원래의 그래프에 대한 DFS 트리를 효과적으로 구한다(3.3 절).

3.1 그래프 변환 단계

이 단계의 주요 아이디어는 교차 대각 edge 쌍을 갖는 cell(부록 1 참조)들을 찾고, 그 cell의 수직(혹은 수평) edge 중 하나를 contract(부록 1참조)하여 임베드된 평면 그래프를 얻는 것이다. 다음은 이에 대한 슈도코드(pseudo-code)를 나타낸다.

```
Input: Solid grid graph G=(V,E).
Output: Embedded planar graph G'.
1: label every edge with '0' in parallel
2: for every subgrid pardo
3:   if the cell has intersecting diagonal
     edge pair then
4:     label a horizontal (vertical) edge
     with '1' in an interleaved manner
5: end if
6: end for
7: for every subgrid pardo
8:  contract edges labelled as '1' (remove it
      and update vertex weight and adjacency
      information) in an interleaved manner
9: end for
```

위 알고리즘의 라인 3에서 대각 교차 edge 쌍은 vertex의 좌표를 이용해 다음과 같이 간단히 인식될 수 있다. (i,j) vertex를 기준으로 하여, 만일 $\{(i+1,j), (i,j+1)\}$과 $\{(i+1,j),(i,j+1)\}$ edge가 동시에 존재하면 이는 대각 교차 edge쌍이 된다. 라인 4에서 '1'로 표기할 edge를 선택하는데 두 가지 제약을 둔다; (ㄱ)선택되는 edge들은 서로 disjoint해야 한다. (ㄴ) 알고리즘 전체에서 수평이든 수직이든 한 가지의 edge를 고정해서 선택한다. 첫 번째 제약은 edge contraction이 특정 부분에 집중적으로 발생하는 것을 방지하기 위한 것이고 두 번째 제약은 첫 번째 제약이 위배되는 것을 방지하기 위

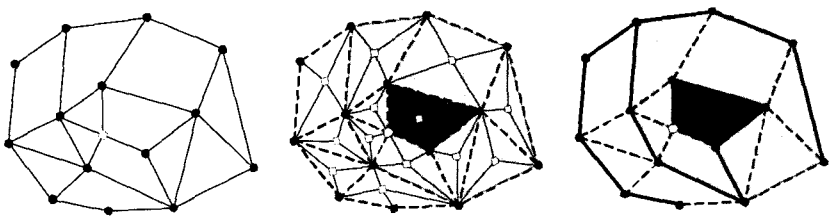


그림 1 Par_Planar_DFS의 예. (a) 주어진 평면화된 그래프 G(흰점은 root를 나타낸다). (b) G의 face-on-vertex 그래프 G_F(점선은 제외). 짙은 회색, 회색, 밝은 회색 영역은 각각 층 0, 1, 2를 나타낸다. (c) G의 DFS-tree(굵은 선)

한 것이다. 즉 수평과 수직 edge들을 임의로 선택하게 된다면 불가피하게 연속된 두 edge를 선택해야만 하는 상황이 발생하게 된다. 다음 보조 정리는 이 알고리즘에 따른 결과를 기술한다.

보조 정리 1. *상기 알고리즘은 한 solid grid 그래프로부터 한 임베드된 평면 그래프를 EREW PRAM 모델에서 $O(n/\log n)$개의 프로세서로 $O(\log n)$ 시간에 구할 수 있다.*

증명. 알고리즘의 타당성은 [8]의 보조 정리 2에 입증되어 있다. 복잡도 증명은 다음과 같다. 주어진 $\sqrt{n}\times\sqrt{n}$ 크기의 solid grid 그래프를 $\sqrt{\log n}\times\sqrt{\log n}$ 크기의 $O(n/\log n)$개의 subgrid들로 나누어, 각 for 루프에서 한 프로세서가 한 subgrid를 처리하도록 설정한다. 특히 라인 4와 8에서 interleaved manner는 동시처리에 따른 다음의 문제들을 해결하기 위한 것이다. 만일 라인 4에서 이웃하는 subgrid를 동시에 처리하게 되면, 그림 2(b)에서와 같이 edge선택의 첫 번째 제약 조건 (ㄱ)이 두 subgrid간의 경계 부분에서 깨어지게 된다. 각 프로세서가 점선을 선택해서 '1'로 표시한다면 그림에서 처럼 경계 부분에서 이웃하는 edge가 선택이 되게 된다. 라인 8에서 만일 이웃하는 subgrid에 대해서 동시에 edge contraction을 수행하게 되면, 마찬가지로 경계 부근에서 consistency가 깨어지는 현상이 발생한다. 따라서 이웃하지 않는 subgrid들을 병렬로 동시에 처리해야만 하는데, 그림 2(a)에서 보듯이 grid 그래프의 경우 이에 대한 반복은 기껏해야 4회에 불과하다(회색 subgrid들이 첫 번째로 병렬처리되고, 그 다음으로 밝은 회색 subgrid들이 처리되고 등등...). 따라서 각 for 루프 내의 시간은 $O(\log n)$이며 $O(n/\log n)$개의 프로세서가 쓰인다. 또한, 상기 interleaving에 의해서 EREW PRAM 모델에서 연산이 가능하다. □

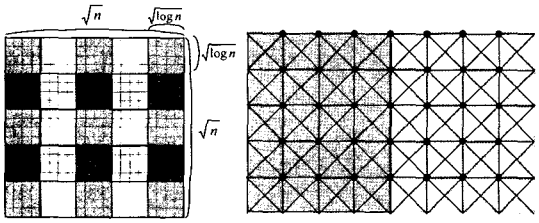


그림 2 (a) Subgrid로 나누는 방법. (b) 두 이웃하는 subgrid들

3.2 Par_Planar_DFS의 적용

2절에서 언급한 Par_Planar_DFS 알고리즘을 이용하여 변형된 그래프인 G'의 한 DFS 트리 T'를 구한다. 그런데 Par_Planar_DFS의 적용에 있어서, [8]에서 언급하듯이 이미 embedding 된 평면 그래프를 다루게 되므로 평면 embedding에 대한 연산은 우리의 고려 대상이 되지 않는다. BFS(breadth-first search) 문제는 모든 edge가 weight가 1로 통일된 상태의 SSSP(single-source shortest paths) 문제와 근본적으로 같기 때문에 복잡 도에 있어서 동일하다고 알려져 있다. 평면 그래프를 위한 현재 알려진 가장 빠른 SSSP 알고리즘은 그래프 분할을 이용한 Frederickson의 것[14]으로 이 알고리즘을 적용할 때 간단히 $\sqrt{\log n}$-분할을 채택함으로써 $O(n/\sqrt{\log n})$ 프로세서로 $O(\sqrt{\log n}\cdot\log\sqrt{\log n})=o(\log n)$ 시간이 소요 되는 병렬 처리를 수행 할 수 있다. 여기서 $\sqrt{\log n}$-분할을 수행하는 것은 변형된 그래프가 여전히 grid로 간단히 임베드될 수 있으므로 $\sqrt[4]{\log n}\times\sqrt[4]{\log n}$의 크기의 정방형 영역의 vertex들을 분할자로 고려하는 것으로 전체 복잡 도에 영향을 주지 않고 분할됨을 말한다. 따라서 우리는 다음의 따름 정리를 얻는다.

따름 정리 1. *n개의 vertex를 가지는 임베드된 비 방향성 평면 그래프 $G=(V,E)$ 와 V의 한 vertex r에 대해서, r을 루트로 하는 G의 한 DFS 트리는 Priority PRAM 모델에서 $O(n/\sqrt{\log n})$개의 프로세서로 $o(\log n)$ 시간에 구할 수 있다.*

한 solid grid 그래프 G ($N=|E|+|V|$)가 $G'(N'=|E'|+|V'|)$으로 변형되었다고 할 때, N과 N'의 관계는 $N'=\alpha N$ ($0<\alpha<1$)이므로 따름 정리 1로부터 이 단계는 Priority PRAM 모델에서 $O(n/\sqrt{\log n})$개의 프로세서로 $o(\log n)$시간에 행해진다.

3.3 최종 DFS 트리의 완성

만일 평면 그래프 변형에서 압축된 한 edge가 루트 r을 한 끝 점으로 할 때, 다른 끝 점을 r''이라 하자. G에서 r''이 제거된 (incident edge들도 제거된) 그래프를 G''이라고 한다면 G'의 DFS-tree T'로부터 $G''(=G-r'')$의 DFS-tree T''을 적절한 simple path(TSSP) 복구 과정을 되풀이함으로써 구할 수 있다.

3.3.1 한 Tree를 Tree Segment Simple Paths(TSSP)들의 Union으로 고려하기

Tree T'에서, P'_{11}, P'_{12}, …, P'_{1n}를 루트로부터 degree ≥ 3인 첫 vertex들(v'_{11}, v'_{12}, …, v'_{1n})간의 simple path들이라고 할 때, 이 들을 tree-segment simple path(TSSP)라 한다. 여기서 이 TSSP들의 *level*을 1(그림 3 참조)이라 둔다. 마찬가지로, level 2 (simple) path들 P'_{21}, P'_{22}, …, P'_{2m} 을 나타낼 수 있다. P'_{ij}는 level i에 있는 j번째 TSSP를 나타낸다. 이는 다음의 정의로 요약된다.

정의 1. *Tree-segment simple path (TSSP): 한 tree에서, 내부 vertex들의 degree가 모두 2인 한*

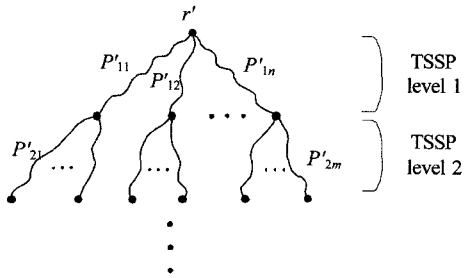


그림 3 한 신장 트리에 대한 tree-segment simple paths

*maximal simple path*를 일컫는다.

정의 2. *TSSP level: 루트를 기점으로 하는 TSSP들에 1을 부여하여 모든 TSSP들에 대한 순차적인 level 번호를 일컫는다.*

다음 보조 정리는 이러한 TSSP와 edge들간의 관계에 대한 것으로 [8]에서 증명되었다.

보조 정리 2 [8]. *한 비 방향성 그래프 G의 spanning tree T에 대해서, 모든 cross-edge 다음 중 한 경우에 해당된다:*

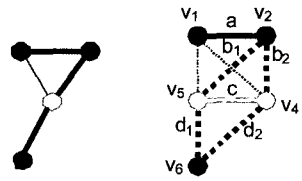
i) 같은 level의 한 TSSP와 또 다른 TSSP 사이의 한 frond

ii) 한 TSSP와 그 보다 상위 level이고 그것에 선행하지 않는 또 다른 TSSP 사이의 한 frond

3.3.2 T'의 한 TSSP를 T"의 한 TSSP로 확장

v'를 (T'의) P'_{ij}에 있는 한 vertex라 하고 G'에서 G''로의 회복 과정을 고려하자. (v'에 상응하는) 두 점 v_1, v_2 G'' 사이의 edge는 edge sequence P''_{ij} (edges_P''_{ij})에 포함시킨다. 만일 P'_{ij}상의 edge e'가 G'' 상에서 두 edge들로 확장된다면, 이러한 edge들은 'candidate' TSSP-edge라고 하고 이 중 하나를 P''_{ij}에 넣는다. P''_{ij}의 모든 candidate TSSP-edge 집합들의 집합을 $cand_{ij}$이라 하자. 그림 4의 예에서는, $cand_{ij}$는 {{*b*1,*b*2},{*d*1,*d*2}}이고 P''_{ij}는 [*v*1, *a*, *v*2, *c*, *v*6]이다.

초기$cand_{ij}$와 P''_{ij}를 계산한 다음 (T''의) P''_{ij}와 P'_{ij} 간의 일대일 대응관계를 다음 슈도 코드와 같이 성립시킨다.



(a) (b)

그림 4 candidate TSSP edge의 예(굵은 점선)

```
(Procedure: TSSP extension)
Input: initial cand_ij and initial P"_ij
Output: a DFS-tree segment simple path P"_ij
1: for every subset of cand_ij
2:   if the edge makes degree of an end-vertex
     > 2 w.r.t initial P"_ij
3:   then remove the edge from cand_ij
4:   if the edge is unique element in a subset
     of cand_ij
5: then include it P"_ij
6: if the edge does not make degree of each
   end-vertex > 2 w.r.t current P"_ij
7: then include the edge in P"_ij
```

위의 라인 2와 3은 준비단계로 이 단계가 없으면, degree가 3보다 큰 vertex가 P''_{ij}에 발생할 가능성이 있게 된다.

그림 5는 이 과정에 대한 예를 보이고 있다. 초기의 $cand_{ij}$는 {{c_1,c_2,c_3},{e_1,e_2}}이고 초기의 P''_{ij}는 [v_1,*a*,v_2,*b*, v_3,*d*,v_6]이다. 또한, 라인 2,3,4과 5 후의 $cand_{ij}$는 {{c_3}, {e_1,e_2}}이고 라인 6과 7 후의 P''_{ij}는 [v_1,*a*,v_2,*b*,v_3,c_3,v_4,*d*, v_5,e_1,v_6]이다.

3.3.3 원래 그래프의 DFS-tree 구하기

G'의 DFS-tree T'로부터 G''(=G-r'')의 DFS-tree T"을 적절한 simple path(TSSP) 복구 과정을 되풀이 하여 구하고, 이후 r''에 대한 한 DFS-tree edge를 찾아 줌으로써 G의 DFS-tree T를 최종적으로 얻는다. 다음 슈도 코드는 이를 나타낸다.

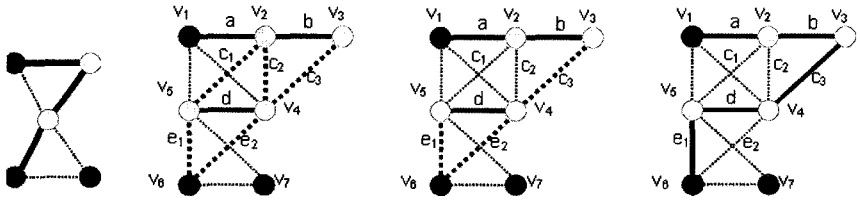


그림 5 TSSP 확장의 예

Input: Contracted graph G', DFS-tree T' of G'.
Output: Original solid graph G, DFS-tree T of G
1: **for** every subgrid **pardo**
2: extend it in an interleaved manner
3: construct initial cands and initial TSSP
4: **for** every TSSP **pardo**
5: conduct TSSP extension
6: find a DFS-tree edge for r''

보조 정리 3. *최종 트리를 완성하는 알고리즘은 EREW PRAM 모델에서 $O(n/\log n)$개의 프로세서로 $O(\log n)$ 시간안에 수행 될 수 있다.*

증명. 최종 트리의 DFS 트리 여부에 대한 타당성은 [8]에서 증명되었다. 첫 번째 for 루프는 첫 번째 알고리즘에서의 그것과 유사하다. 라인 3은 간단히 구할 수 있고, 두 번째 for 루프는 두 가지 형태로 진행된다. 크기가 log n 보다 작은 TSSP들은 한 번에(log n 크기만큼의 TSSP들을) 한 프로세서에 할당하여 처리하도록 하고 log n 보다 큰 TSSP들은 log n 만큼 나누어 다른 프로세서에 할당하여 처리하도록 한다. 따라서, TSSP extension 알고리즘은 크기가 $O(\log n)$인 TSSP를 다루게 되는데, 그 알고리즘의 $cand_{ij}$의 subset의 개수는 기껏해야 P_{ij}의 edge 수 이므로 $O(\log n)$으로 제한된다. 또한 grid 그래프의 경우 확장 이전의 edge에 대해서 확장 후에 기껏해야 4개의 대응 edge가 존재하므로, 각 $cand_{ij}$의 subset의 edge 수는 기껏해야 4개 이다. 따라서 선형적인 연산을 수행하더라도 $O(\log n)$의 시간만 소모된다.

그러므로 두 번째 for 루프는 $O(n/\log n)$개의 프로세서로 $O(\log n)$ 시간 안에 수행 될 수 있다. □

보조 정리 1, 3과 따름 정리 1로 부터 본 논문의 주 결과인 다음의 정리 2가 얻어진다.

정리 2. *임의의 solid grid 그래프의 DFS-tree는 Priority PRAM 모델에서 $O(n/\sqrt{\log n})$개의 프로세서로 $O(\log n)$ 시간에 구할 수 있다.*

4. 결 론

제안 알고리즘의 총 비용(cost)은 $O(n\sqrt{\log n})$으로 이는 평면 그래프에 대한 가장 좋은 알고리즘[11]의 $O(n\log n)$보다 효율적이다. 이것은 많은 비용이 요구되는 평면 임베딩 절차가 우리 문제에서는 필요하지 않고 grid 그래프의 점 분포가 규칙적이기 때문에 분할이 쉽게 이루어지기 때문에 타나는 결과이다. 일반적인 grid 그래프는 edge의 contract을 이용하여 평면 그래프로 변형이 힘들기 때문에 일반적인 grid 그래프에 대한 병렬 DFS 알고리즘을 개발하는 것이 당면 과제이다.

참 고 문 헌

[1] V. Ramachandran, Parallel open ear decomposition with applications to graph biconnectivity. and triconnectivity, in Synthesis of Parallel Algorithms, J. Reif, ed., Morgan-Kaufmann, pp. 275-340, 1993.

[2] R. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput., Vol.1, No.2, pp.146-160, 1972.

[3] E. Charniak and D. McDermott, Introduction to artificial intelligence, Addison-Wesley, 1985.

[4] E. Reghgati and D. Corneil, Parallel computations in graph theory, SIAM J. Comput., Vol.7, No.3, pp.230-237, 1978.

[5] J. H. Reif, Depth-first searches inherently sequential, Inform. Process. Lett., Vol.20, No.5, pp.229-234, 1985.

[6] A. Aggarwal and R. J. Anderson, A random NC algorithm for depth-first search, Combinatorica, Vol.8, No.1, pp.1-12, 1988.

[7] A. Aggarwal, Richard J. Anderson, and Ming-Yang Kao, Parallel depth-first search in general directed graphs, SIAM J. Comput., Vol.19, No.2, pp.397-409, 1990.

[8] Jun-Ho Her and R. S. Ramakrishna, External-memory depth-first search algorithm for solid grid graphs, Information Processing Letters, Vol.93, No.4, pp.177-183, 2005.

[9] J. R. Smith, Parallel algorithms for depth-first searches I. Planar graphs, SIAM J. Comput., Vol.15, No.3, pp.814-830, 1986.

[10] A. V. Goldberg, S. A. Plotkin, and G. E. Shannon, Parallel symmetry-breaking in sparse graphs, SIAM J. Discrete Math., Vol.1, No.1, pp.434-446, 1988.

[11] G. E. Shannon, A linear-processor algorithm for depth-first search in planar graphs, Inform. Process. Lett., Vol.29, No.3, pp.119-123, 1988.

[12] T. Hagerup, Planar depth-first search in O(log n) parallel time, SIAM J. Comput., Vol.19, No.4, pp.678-704, 1990.

[13] Ming-Yang Kao, Shang-Hua Teng, and K. Toyama, An optimal parallel algorithm for planar cycle separators, Proc. of the 3rd Int. Workshop on Algorithms and Data Structures, pp.407-420, 1993.

[14] Greg N. Frederickson, Fast algorithms for shortest paths in planar graphs, with applications, SIAM J. Comput, Vol.16, No.6, pp.1004-1022, 1987.

[15] L. Arge, L. Toma, and J. S. Vitter, "I/O-Efficient Algorithms for Problems on Grid-based Terrains," Journal of Experimental Algorithms, Vol.6, No.1, pp.1-19, 2001.

부 록 1

그래프 G에서 한 edge e의 contraction은 다음과 같이 이루어진다; 그 edge의 양끝 vertex들인 v_i와 v_j를 새로운 한 vertex v_k로 병합한다. v_k의 비중(weight) $w_v(v_k)$는 $w_v(v_i)+w_v(v_j)$로 정한다. edge의 비중도 이와 같이 계산한다.

높이가 h이고 너비가 w인 grid 그래프는, vertex 집합이 $\{(x,y)|\ 0\le x < h,\ 0 \le y < w\}$이고 edge 집합이 $\{\{(u,v),(x,y)\}|\ |u-x|+|v-y|=1\}$인 그래프이다. 그러나 우리는 대각 edge도 허용하는 좀 더 일반화된 grid 그래프를 고려한다; 이 경우 비평면일 가능성이 생긴다. 이런 일반 grid 그래프는 GIS[15]와 같은 응용에서 자주 등장한다. Grid 그래프의 한 vertex는 그것의 좌표인 (i,j)로 나타낸다. 또한, 네 vertex (i,j), $(i+1,j)$, $(i,j+1)$, 그리고 $(i+1,j+1)$으로 유도되는 부그래프(subgraph)를 흔히 'cell'이라고 부른다. Solid grid 그래프는 내부의 모든 vertex들이 인접 수직/수평 edge들로 연결된 grid 그래프이다. 그림 6은 일반 grid 그래프와 solid grid 그래프에 대한 예를 보인다.

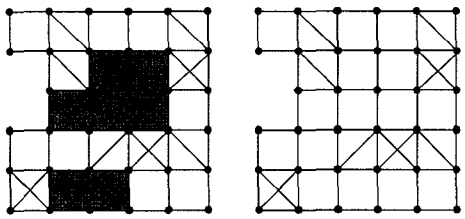


그림 6 (a) 일반 grid 그래프와 (b) solid grid 그래프 ((a)에서 회색 영역에는 수평, 수직 edge가 존재하지 않는다.)

허 준 호

2000년 2월 경북대학교 전자전기공학부 학사. 2002년 2월 광주과학기술원 정보통신공학과 석사. 2002년 3월~현재 광주과학기술원 정보통신공학과 박사과정. 관심분야는 알고리즘 설계 및 분석, 병렬 및 분산 알고리즘, 입출력 효율적인 알고리즘, 유/무선 통신망 관련 알고리즘

R. S. Ramakrishna

1966년 Univ. of Mysore, 전자공학과 학사. 1975년 Univ. of Nagpur, 전자공학과 석사. 1979년 Indian Institute of Technology (Kanpur), 전자공학과 박사 1980년~1986년 Indian Institute of Technology (Bombay), 조교수. 1986년~1990년 Indian Institute of Technology (Bombay), 부교수. 1990년~1996년 Indian Institute of Technology (Bombay), 교수. 1996년~현재 광주과학기술원(GIST), 정보통신공학과 교수. 관심분야는 컴퓨터 그래픽스, 컴퓨터 비전, 병렬 및 분산 알고리즘, Quantum Computing, Theoretical Computer Science