

# LAGraph Algorithms

LAGraph Working Group

October 21, 2020

## Abstract

Theoretical documentation for LAGraph.

## 1 Notation

Table 1 contains the notation of GraphBLAS operations

Additionally, we use  $D = \text{diag}(J, n)$  to construct a diagonal matrix  $D \leftarrow \{J, J, [1, 1, \dots, 1]\}$ . The elements of the matrix are  $D(j, j) = 1$  for  $j \in J$ .

Initializing scalars, vectors, and matrices (GraphBLAS methods):

- $s = \text{fp64}()$
- $u = \text{fp32}(n)$
- $A = \text{uint16}(m, n)$
- $A = \text{int64}(k, m)$

## 2 Algorithms

LAGraph [12] implements graph algorithms using the GraphBLAS C API [13].

An incomplete list of GraphBLAS algorithms:

- MSBFS [8], bidirectional BFS [8], pushpull BFS [19]
- DFS [16]
- weakly connected components [21]
- SCC (LAGraph)
- SSSP, delta-stepping [17]
- triangle count [1, 5], triangle enumeration [1], item local clustering coefficient [4]
- $k$ -truss [5]
- betweenness centrality [12]
- closeness centrality [8]
- DNN algorithm [10, 7]
- PageRank variants (at least 2) [4], IISWC paper, ...
- Louvain [11]
- property graphs: incremental TTC case [9], SIGMOD 2014 Contest [8] Roi Lipman's talk<sup>1</sup>
- CFPQs based on a string of GRADES/ADBIS/other papers [2, 3, 14, 18, 15]
- Implementations: GBTL<sup>2</sup>, SuiteSparse:GraphBLAS [6], GraphBLAST [20]

---

<sup>1</sup><http://wiki.ldbcouncil.org/pages/viewpage.action?pageId=106233859&preview=/106233859/111706128/LDBC-July-2019.pdf>

<sup>2</sup><https://github.com/cmu-sei/gbtl>

op./method	name	notation
<b>mxm</b>	matrix-matrix multiplication	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{A} + . \times \mathbf{B}$
<b>vxm</b>	vector-matrix multiplication	$\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{u} + . \times \mathbf{A}$
<b>mxv</b>	matrix-vector multiplication	$\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{A} + . \times \mathbf{u}$
<b>eWiseAdd</b>	element-wise addition set union of patterns	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{A} + \mathbf{B}$ $\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{u} + \mathbf{v}$
<b>eWiseMult</b>	element-wise multiplication set intersection of patterns	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{A} \times \mathbf{B}$ $\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{u} \times \mathbf{v}$
<b>extract</b>	extract submatrix extract column vector extract row vector extract subvector	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{A}(\mathbf{I}, \mathbf{J})$ $\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{A}(:, \mathbf{j})$ $\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{A}(\mathbf{i}, :)$ $\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{u}(\mathbf{I})$
<b>assign</b>	assign matrix to submatrix with mask for $\mathbf{C}$ assign scalar to submatrix with mask for $\mathbf{C}$ assign vector to subvector with mask for $\mathbf{w}$ assign scalar to subvector with mask for $\mathbf{w}$	$\mathbf{C} \langle \mathbf{M} \rangle (\mathbf{I}, \mathbf{J}) + = \mathbf{A}$ $\mathbf{C} \langle \mathbf{M} \rangle (\mathbf{I}, \mathbf{J}) + = \mathbf{s}$ $\mathbf{w} \langle \mathbf{m} \rangle (\mathbf{I}) + = \mathbf{u}$ $\mathbf{w} \langle \mathbf{m} \rangle (\mathbf{I}) + = \mathbf{s}$
<b>subassign (GxB)</b>	assign matrix to submatrix with submask for $\mathbf{C}(\mathbf{I}, \mathbf{J})$ assign scalar to submatrix with submask for $\mathbf{C}(\mathbf{I}, \mathbf{J})$ assign vector to subvector with submask for $\mathbf{w}(\mathbf{I})$ assign scalar to subvector with submask for $\mathbf{w}(\mathbf{I})$	$\mathbf{C}(\mathbf{I}, \mathbf{J}) \langle \mathbf{M} \rangle + = \mathbf{A}$ $\mathbf{C}(\mathbf{I}, \mathbf{J}) \langle \mathbf{M} \rangle + = \mathbf{s}$ $\mathbf{w}(\mathbf{I}) \langle \mathbf{m} \rangle + = \mathbf{u}$ $\mathbf{w}(\mathbf{I}) \langle \mathbf{m} \rangle + = \mathbf{s}$
<b>apply</b>	apply unary operator	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{f}(\mathbf{A})$ $\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{f}(\mathbf{u})$
<b>select (GxB)</b>	apply select operator	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{select}(\mathbf{A}, \mathbf{f}(\mathbf{k}))$ $\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{select}(\mathbf{low} \leq \mathbf{A} \leq \mathbf{up})$ $\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{select}(\mathbf{u}, \mathbf{f}(\mathbf{k}))$ $\mathbf{w} \langle \mathbf{m} \rangle + = \mathbf{select}(\mathbf{low} \leq \mathbf{u} \leq \mathbf{up})$
<b>reduce</b>	reduce matrix to column vector reduce matrix to scalar reduce vector to scalar	$\mathbf{w} \langle \mathbf{m} \rangle + = [\mathbf{+} \mathbf{A}]$ $\mathbf{s} + = [\mathbf{+} \mathbf{A}]$ $\mathbf{s} + = [\mathbf{+} \mathbf{u}]$
<b>transpose</b>	transpose	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{A}'$
<b>kroncker</b>	Kronecker multiplication	$\mathbf{C} \langle \mathbf{M} \rangle + = \mathbf{kron}(\mathbf{A}, \mathbf{B})$
<b>new</b>	new matrix new vector	$\mathbf{A} = \mathbf{TYPEPRECISION}(\mathbf{n}, \mathbf{m})$ $\mathbf{u} = \mathbf{TYPEPRECISION}(\mathbf{n})$
<b>build</b>	build matrix from index/value arrays build vector from index/value arrays	$\mathbf{C} \leftarrow \{\mathbf{I}, \mathbf{J}, \mathbf{X}\}$ $\mathbf{w} \leftarrow \{\mathbf{I}, \mathbf{X}\}$
<b>extractTuples</b>	extract index/value arrays	$\{\mathbf{I}, \mathbf{J}, \mathbf{X}\} \leftarrow \mathbf{A}$ $\{\mathbf{I}, \mathbf{X}\} \leftarrow \mathbf{u}$
<b>dup</b>	duplicate matrix duplicate vector	$\mathbf{C} \leftarrow \mathbf{A}$ $\mathbf{w} \leftarrow \mathbf{u}$
<b>extractElement</b>	extract scalar element	$\mathbf{s} = \mathbf{A}(\mathbf{i}, \mathbf{j})$ $\mathbf{s} = \mathbf{u}(\mathbf{i})$
<b>setElement</b>	set element	$\mathbf{C}(\mathbf{i}, \mathbf{j}) = \mathbf{s}$ $\mathbf{w}(\mathbf{i}) = \mathbf{s}$

Table 1: GraphBLAS operations and methods based on [6]. *Notation:* Matrices and vectors are typeset in bold, starting with uppercase ( $\mathbf{A}$ ) and lowercase ( $\mathbf{u}$ ) letters, respectively. Scalars including indices are lowercase italic ( $\mathbf{s}$ ,  $\mathbf{i}$ ,  $\mathbf{j}$ ) while arrays are uppercase italic ( $\mathbf{X}$ ,  $\mathbf{I}$ ,  $\mathbf{J}$ ).  $+$  and  $\times$  are the addition and multiplication operators forming a semiring and default to conventional arithmetic  $+$  and  $\times$  operators.  $\odot$  is the apply operator. Masks  $\langle \mathbf{M} \rangle$  and  $\langle \mathbf{m} \rangle$  are used to selectively write to the result matrix/vector. The complements of masks  $\langle \mathbf{M} \rangle$ ,  $\langle \mathbf{m} \rangle$  can be selected with the negation symbol, denoted with  $\langle !\mathbf{M} \rangle$ ,  $\langle !\mathbf{m} \rangle$ , respectively. Masks with “replace” semantics (annihilating all elements outside the mask) are denoted with  $\langle \langle \mathbf{M} \rangle \rangle$ ,  $\langle \langle !\mathbf{M} \rangle \rangle$ ,  $\langle \langle \mathbf{m} \rangle \rangle$ , and  $\langle \langle !\mathbf{m} \rangle \rangle$ . The structure of the mask is denoted with  $\langle \{\mathbf{M}\} \rangle$ ,  $\langle !\{\mathbf{M}\} \rangle$ ,  $\langle \{\mathbf{m}\} \rangle$ , and  $\langle !\{\mathbf{m}\} \rangle$ .

---

**Algorithm 1:** Breadth-first search.

---

```
1 Function BFS
2   frontier(s) = TRUE
3   for level = 1 to n - 1 do
4     seen <frontier>= level
5     frontier << !seen>>= frontier any.pair A
```

---

---

**Algorithm 2:** Breadth-first search (push/pull).

---

```
1 Function BFS
2   frontier(s) = TRUE
3   for level = 1 to n - 1 do
4     seen <frontier>= level
5     push = use some heuristics to determine whether to push/pull
6     if push then
7       frontier << !seen>>= frontier any.pair A
8     else
9       frontier << !seen>>= A any.pair frontier
```

---

---

**Algorithm 3:** Multi-source breadth-first search.

---

**Data:** ...

**Result:** ...

```
1 Function MSBFS
2   Frontier <- diag(S,n)
3   for level = 1 to n - 1 do
4     Seen <Frontier>= level
5     Frontier << !Seen>>= Frontier any.pair A
```

---

---

**Algorithm 4:** Betweenness centrality.

---

```
1 Function MSBFS
2   // The NumSp structure holds the number of shortest paths for each node and starting
   // vertex discovered so far.
3   // Initialized to source vertices.
4   NumSp <- {s, [1, 1, ..., 1]}
5   // The Frontier holds the number of shortest paths for each node and starting vertex
   // discovered so far.
6   // Initialized to source vertices.
7   Frontier <NumSp>= A(s,:)
8   d = 0
9   // The Sigmas matrices store frontier information for each level of the BFS phase.
10  // BFS phase (forward sweep)
11  do
12    // Sigmas[d](:,s) = dth level frontier from source vertex s
13    Sigmas[d] = bool(n, nsver)
14    Sigmas[d](:, :) = Frontier // Convert matrix to Boolean
15    NumSp = NumSp + Frontier // Accumulate path counts
16    Frontier <<NumSp>>= A' + . × Frontier // Update frontier
17  while nvals(Frontier) > 0
18  NumSpInv = fp32(n, nsver)
19  NumSpInv = 1.0 DIV NumSp
20  BCU = fp32(n, nsver)
21  BCU(:) = 1.0 // Make BCU dense, initialize all elements to 1.0
22  W = fp32(n, nsver)
23  // Tally phase (backward sweep)
24  for i = d - 1 downto 0 do
25    W << Sigmas[i] >>= NumSpInv DIV BCU
26    W << Sigmas[i - 1] >>= A + . × W // Add contributions by successors and mask with that
    // BFS level's frontier.
27    BCU += W × NumSp
28  // Row reduce BCU and subtract nsver from every entry to account for 1 extra value
    // per BCU row element
29  delta = [+ BCU]
30  delta MINUS = nsver
```

---

---

**Algorithm 5:** PageRank (used in Graphalytics).

---

**Data:** alpha constant (damping factor)

**Result:** ...

```
1 Function PageRank
2   pr(:) = 1/n
3   outdegrees = [+j A(:, j)]
4   for k = 1 to numIterations do
5     importance = pr DIV outdegrees
6     importance = times(importance, alpha) // apply the times(x, s) = x · s operator
7     importance = importance + . × A
8     danglingVertexRanks < !outdegrees >= pr(:)
9     totalDanglingRank = [+ danglingVertexRanks(i)](alpha)/(n)
10    pr(:) = (1 - alpha)/(n) + totalDanglingRank
11    pr < importance >= pr + importance
```

---

---

**Algorithm 6:** Algebraic Bellman-Ford for SSSP.

---

```
1 Function SSSP
2    $d(s) = 0$ 
3   for  $k = 1$  to  $n - 1$  do
4      $dtmp = d \text{ min.plus } A$ 
5     if  $dtmp = d$  then
6       break
7      $d \leftarrow dtmp$ 
```

---

---

**Algorithm 7:** Delta-stepping SSSP.

---

**Data:**

$A, A_H, A_L \in \text{fp}(|V|, |V|)$

$s, i \in \text{uint}()$

$\Delta \in \text{fp}()$

$t, t_{\text{Req}} \in \text{fp}(|V|)$

$t_{B_i}, e \in \text{uint}(|V|)$

```
1 Function DeltaStepping
2    $A_L = \text{select}(0 < A \leq \Delta)$ 
3    $A_H = \text{select}(\Delta < A)$ 
4    $t(:) = \infty$ 
5    $t(s) = 0$ 
6   while  $nvals(\text{select}(i\Delta \leq t)) \neq 0$  do
7      $s = 0$ 
8      $t_{B_i} = \text{select}(i\Delta \leq t < (i+1)\Delta)$ 
9     while  $t_{B_i} \neq 0$  do
10       $t_{\text{Req}} = A_L' + . \times (t \times t_{B_i})$ 
11       $e = \text{select}(0 < e + t_{B_i})$ 
12       $t_{B_i} = \text{select}(i\Delta \leq t_{\text{Req}} < (i+1)\Delta) \times (t_{\text{Req}} \min_+ t)$ 
13       $t_{\text{Req}} = A_H' + . \times (t \times e)$ 
14       $t = t \min t_{\text{Req}}$ 
15       $i = i + 1$ 
```

---

---

**Algorithm 8:** All-pairs shortest path (Floyd–Warshall algorithm).

---

```
1 Function FloydWarshall
2    $D \leftarrow A$ 
3   for  $k = 1$  to  $n$  do
4      $D = D \text{ min}[D(:, k) \text{ min.plus } (D(k, :))]$ 
```

---

---

**Algorithm 9:** FastSV algorithm.

---

```
1 Function FastSV
2   n = nrow(A)
3   gf = f
4   dup = gf
5   mngf = gf
6   {I,X} <- f
7   repeat
8     // Step 1: Stochastic hooking
9     mngf = mngf min A
10    mngf = mngf second.min gf
11    f(X) = f min mngf
12    // Step 2: Aggressive hooking
13    f = f min mngf
14    // Step 3: Shortcutting
15    f = f min gf
16    // Step 4: Calculate grandparents
17    {I,X} <- f
18    gf = f(X)
19    // Step 5: Check termination
20    diff = dup ≠ gf
21    sum = [+i diff(i)]
22    dup = gf
23  until sum == 0
```

---

---

**Algorithm 10:** Triangle count (Cohen's algorithm).

---

```
1 Function TriangleCount
2   L = tril(A)
3   U = triu(A)
4   B = L + . × U
5   C = B × A
6   t = [+ C]/2
```

---

---

**Algorithm 11:** Triangle count (Sandia).

---

```
1 Function TriangleCount
2   L = tril(A)
3   C <L>= L + . × L
4   t = [+ C]
```

---

---

**Algorithm 12:** Triangle count (FLAME).

---

```
1 Function TriangleCountFlame
2   for i = 2 to n - 1 do
3     A20 = A(i + 1 : n, 0 : i - 1)
4     a10 = A(0 : i - 1, i)
5     a12 = A(i, i + 1 : n)
6     t += a10 + . × A20 + . × a12
```

---

---

**Algorithm 13:** Local clustering coefficient.

---

```
1 Function PageRank
2   Tri <A>= A + . × A // compute triangle count matrix
3   tri = [+ Tri] // reduce to triangle count vector
4   deg = [+ A] // reduce to vertex degree vector
5   wed = perm2(deg) // apply perm2(x) = x · (x - 1) to get wedge count vector
6   lcc = tri DIV wed // LCC vector
```

---

---

**Algorithm 14:**  $k$ -truss algorithm.

---

```
1 Function KTruss
2   C <- A
3   nonzeros <- nvals(C)
4   for i = 1 to n - 1 do
5     C < C >= C + .1 and C
6     C = select(C ≥ k - 2)
7     if nonzeros == nvals(C) then
8       break
9     nonzeros <- nvals(C)
```

---

---

**Algorithm 15:** Louvain algorithm (WIP).

---

```
1 Function Louvain
2   G + = G'
3   k = [+ A]
4   m = (1)/(2)[+ k]
5   S <- I
6
7   vertices_changed <- nvals(k)
8   while vertices_changed > 0 do
9     for j ∈ range(|V|) do
10      v = G(j, :)
11      tq = v any.pair S
12      sr = S(j, :)
13      S(j, :) = empty
14
15      q <- k
16      q < k > × = - k(j)/m
17      q + = v
18      q1 < tq > = q + . × S
19
20      t = (q1 == [max q1])
21      while nvals(t) ≠ 1 do
22        p = random() × t
23        t = (p == [max p])
24      S(j, :) = t
25
26      if nvals(sr × t) == 0 then
27        vertices_changed = nvals(k)
28      vertices_changed = vertices_changed - 1
```

---

---

**Algorithm 16:** Community detection using label propagation (for undirected graphs).

---

```
1 Function CDLP
2   L <- diag([0, 1, ..., n - 1])
3   for k = 1 to t do
4     F = A any . second L // Frequency matrix
5     {I, -, X} <- F
6     merge_sort_pairs(I, X)
7     labels = for each row in I, select min mode value from X
```

---

## References

- [1] A. Azad, A. Buluç, and J. R. Gilbert, “Parallel triangle counting and enumeration using matrix algebra,” in *GABB at IPDPS*. IEEE Computer Society, 2015, pp. 804–811. [Online]. Available: <https://doi.org/10.1109/IPDPSW.2015.75>
- [2] R. Azimov and S. Grigorev, “Context-free path querying by matrix multiplication,” in *GRADES-NDA at SIGMOD*. ACM, 2018, pp. 5:1–5:10. [Online]. Available: <https://doi.org/10.1145/3210259.3210264>
- [3] —, “Path querying with conjunctive grammars by matrix multiplication,” *Programming and Computer Software*, vol. 45, no. 7, pp. 357–364, 2019. [Online]. Available: <https://doi.org/10.1134/S0361768819070041>
- [4] M. Aznaveh, J. Chen, T. A. Davis, B. Hegyi, S. P. Kolodziej, T. G. Mattson, and G. Szárnyas, “Parallel GraphBLAS with OpenMP,” in *CSC*. SIAM, 2020, pp. 138–148. [Online]. Available: <https://doi.org/10.1137/1.9781611976229.14>
- [5] T. A. Davis, “Graph algorithms via SuiteSparse:GraphBLAS: Triangle counting and K-truss,” in *HPEC*. IEEE, 2018. [Online]. Available: <https://doi.org/10.1109/HPEC.2018.8547538>
- [6] —, “Algorithm 1000: SuiteSparse:GraphBLAS: Graph algorithms in the language of sparse linear algebra,” *ACM Trans. Math. Softw.*, 2019. [Online]. Available: <https://doi.org/10.1145/3322125>
- [7] T. A. Davis, M. Aznaveh, and S. P. Kolodziej, “Write quick, run fast: Sparse deep neural network in 20 minutes of development time via SuiteSparse:GraphBLAS,” in *HPEC*. IEEE, 2019, pp. 1–6. [Online]. Available: <https://doi.org/10.1109/HPEC.2019.8916550>
- [8] M. Elekes, A. Nagy, D. Sándor, J. B. Antal, T. A. Davis, and G. Szárnyas, “A GraphBLAS solution to the SIGMOD 2014 programming contest using multi-source BFS,” in *HPEC*, 2020.
- [9] M. Elekes and G. Szárnyas, “An incremental GraphBLAS solution for the 2018 TTC Social Media case study,” in *GrAPL at IPDPS*. IEEE, 2020, pp. 203–206. [Online]. Available: <https://doi.org/10.1109/IPDPSW50202.2020.00045>
- [10] J. Kepner, M. Kumar, J. E. Moreira, P. Pattnaik, M. J. Serrano, and H. M. Tufo, “Enabling massive deep neural networks with the GraphBLAS,” in *HPEC*. IEEE, 2017, pp. 1–10. [Online]. Available: <https://doi.org/10.1109/HPEC.2017.8091098>
- [11] T. M. Low, D. G. Spampinato, S. McMillan, and M. Pelletier, “Linear algebraic louvain method in python,” in *GrAPL at IPDPS*. IEEE, 2020, pp. 223–226. [Online]. Available: <https://doi.org/10.1109/IPDPSW50202.2020.00050>
- [12] T. Mattson, T. A. Davis, M. Kumar, A. Buluç, S. McMillan, J. E. Moreira, and C. Yang, “LAGraph: A community effort to collect graph algorithms built on top of the GraphBLAS,” in *GrAPL at IPDPS*, 2019. [Online]. Available: <https://doi.org/10.1109/IPDPSW.2019.00053>
- [13] T. G. Mattson, C. Yang, S. McMillan, A. Buluç, and J. E. Moreira, “GraphBLAS C API: Ideas for future versions of the specification,” in *HPEC*. IEEE, 2017. [Online]. Available: <https://doi.org/10.1109/HPEC.2017.8091095>
- [14] N. Mishin, I. Sokolov, E. Spirin, V. Kutuev, E. Nemchinov, S. Gorbatyuk, and S. Grigorev, “Evaluation of the context-free path querying algorithm based on matrix multiplication,” in *GRADES-NDA at SIGMOD*. ACM, 2019, pp. 12:1–12:5. [Online]. Available: <https://doi.org/10.1145/3327964.3328503>
- [15] E. Orachev, I. Epelbaum, R. Azimov, and S. Grigorev, “Context-free path querying by Kronecker product,” in *ADBIS*, ser. Lecture Notes in Computer Science, vol. 12245. Springer, 2020, pp. 49–59. [Online]. Available: [https://doi.org/10.1007/978-3-030-54832-2\\_6](https://doi.org/10.1007/978-3-030-54832-2_6)
- [16] D. G. Spampinato, U. Sridhar, and T. M. Low, “Linear algebraic depth-first search,” in *ARRAY at PLDI*. ACM, 2019, pp. 93–104. [Online]. Available: <https://doi.org/10.1145/3315454.3329962>
- [17] U. Sridhar, M. Blanco, R. Mayuranath, D. G. Spampinato, T. M. Low, and S. McMillan, “Delta-stepping SSSP: From vertices and edges to graphblas implementations,” in *GrAPL at IPDPS*. IEEE, 2019, pp. 241–250. [Online]. Available: <https://doi.org/10.1109/IPDPSW.2019.00047>
- [18] A. Terekhov, A. Khoroshev, R. Azimov, and S. Grigorev, “Context-free path querying with single-path semantics by matrix multiplication,” in *GRADES-NDA at SIGMOD*. ACM, 2020, pp. 5:1–5:12. [Online]. Available: <https://doi.org/10.1145/3398682.3399163>



- [19] C. Yang, A. Buluç, and J. D. Owens, “Implementing push-pull efficiently in GraphBLAS,” in *ICPP*. ACM, 2018, pp. 89:1–89:11. [Online]. Available: <https://doi.org/10.1145/3225058.3225122>
- [20] —, “GraphBLAST: A high-performance linear algebra-based graph framework on the GPU,” *CoRR*, vol. abs/1908.01407, 2019, <http://arxiv.org/abs/1908.01407>. [Online]. Available: <http://arxiv.org/abs/1908.01407>
- [21] Y. Zhang, A. Azad, and Z. Hu, “FastSV: A distributed-memory connected component algorithm with fast convergence,” in *PPSC*. SIAM, 2020, pp. 46–57. [Online]. Available: <https://doi.org/10.1137/1.9781611976137.5>