New Multiple Pairs Shortest Paths Algorithms

By

I-Lin Wang

Prof. Ellis Johnson

Prof. Joel Sokol

ISyE, GA Tech

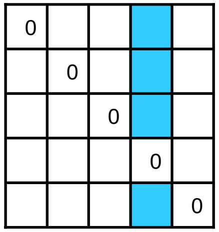
Nov 19, 2002

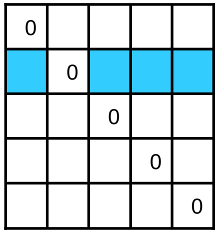
Overview of this talk

- Path Algebra
- Review SSSP, APSP algorithms
  - SSSP: LC, LS
  - APSP: FW, Carré
- Propose 3 new MPSP algorithms
  - SLU
  - DLU1
  - DLU2
  - Implementation issues
- Discussion
  - General arc cost, Negative cycle
  - Complexity

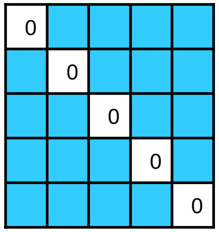
Shortest Path Problem/Algorithm

- Single Source Shortest Path (SSSP)
  - Nonnegative Arc Cost
  - General Arc Cost

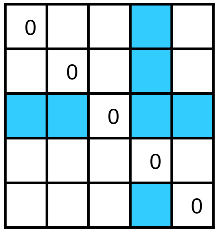


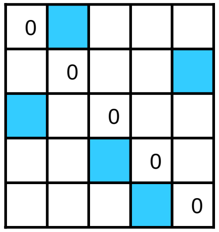


- All Pairs Shortest Paths (APSP)
  - Combinatorial Type Algorithms
  - Algebraic Type Algorithms
  - LP Type Algorithms



- Multiple Pairs Shortest Paths (MPSP)





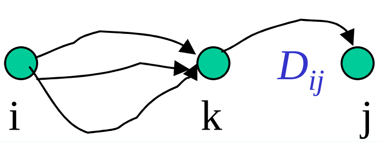
Path Algebra

- Problem: Given a measure matrix C
  - NO (1)multiple arcs, (2)loops (C_{ii}=0), (3)negative cycles
  - C_{ij}=arc length of (i,j) if it exists , or ∞ o.w.
  - What is the distance matrix D?
    - D_{ij}=distance from i to j , or ∞ if i can't reach j

- Define path algebra

Original definition	a⊕b	a⊗b	e	0(Null)
Path algebra	min{a,b}	a+b	0	∞

$$D_{ij} = \begin{cases} \min\limits_{k \neq i,j} \left(C_{ik} + D_{kj} \right) & , i \neq j \\ 0 & , i = j \end{cases} \quad \Leftrightarrow \quad D = C \otimes D \oplus I_n$$



ALL-ALL SP Algorithm

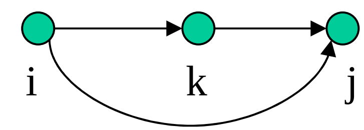
- Solving $D = C \otimes D \oplus I_n$ is like solving system of linear equations

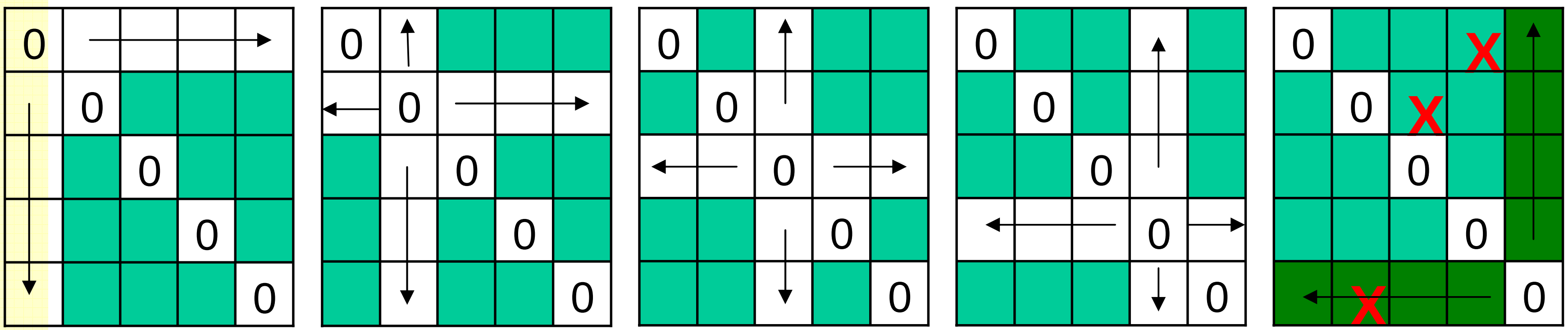
- Jacobi : Bellman-Ford $O(n^2m)$
- Gauss-Seidel : Ford-Fulkerson
- Gauss-Jordan : Floyd-Warshall Algorithm $O(n^3)$

- Gauss : $O(n^3)$: Carré , DLU
  - LU factorization (once)
  - forward elimination (for each node)
  - backward substitution (for each node)
  - Same # iterations as Floyd-Warshall, but can decompose to solve some-some shortest path problem

Floyd-Warshall Algorithm

- Idea: update C_{ij} by $\min\{C_{ij}, C_{ik}+C_{kj}\}$ where $k \neq i, j$
  (triple comparison)





Total # triple comparison = $n(n-1)(n-2)$

It is $O(n^3)$ → not good for sparse graph.

Q={(1,4),(2,3),(5,2)}

Note:

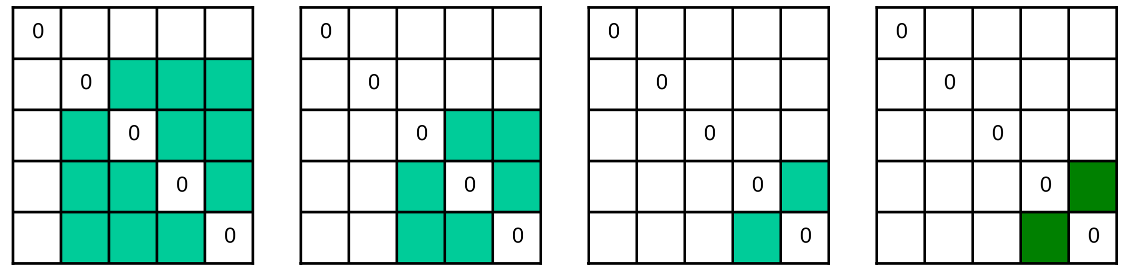
- In the beginning of the last iteration, we already got optimal solution for [n]-ALL and ALL-[n]

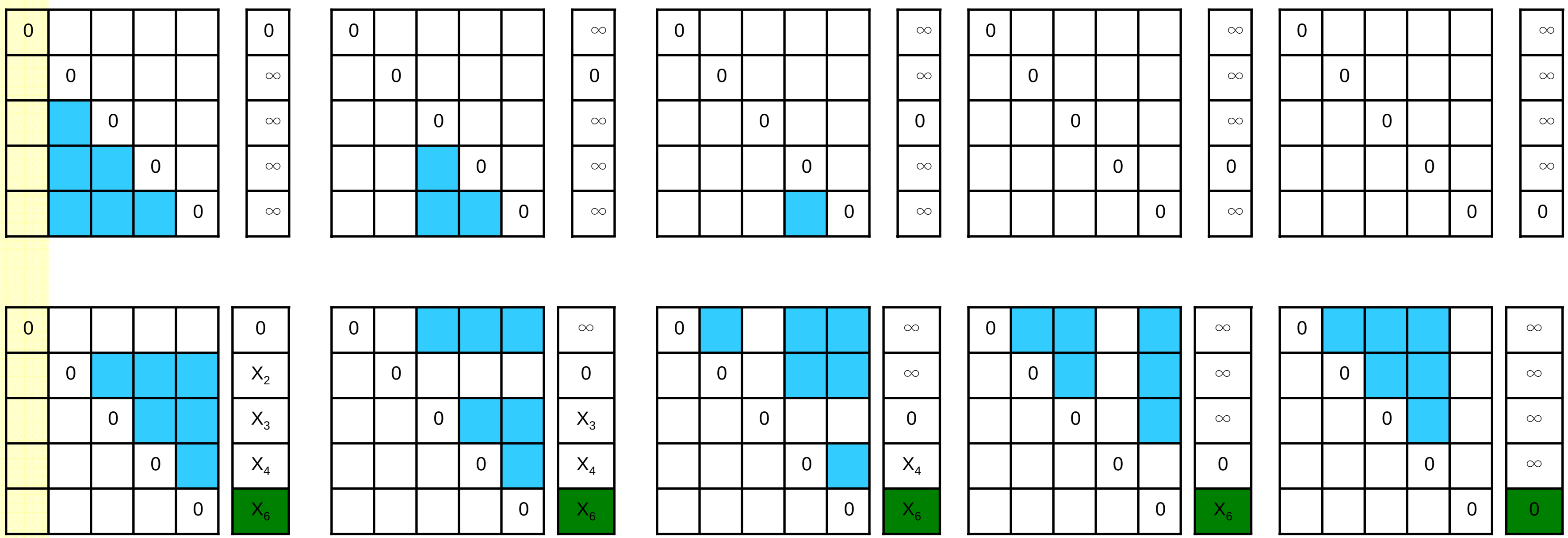
Illustration of Carré's Algorithm

LU() $\frac{n(n-1)(n-2)}{3}$

Forward() $\frac{n(n-1)(n-2)}{6}$

Backward() $\frac{n(n-1)(n-2)}{2}$

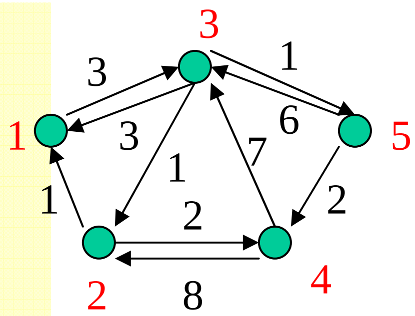


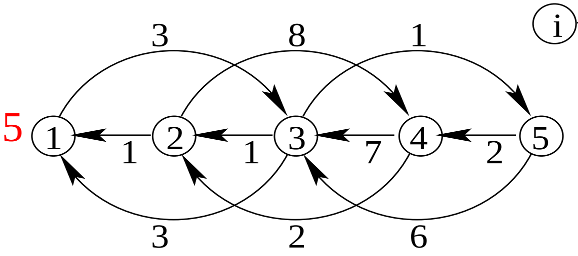


ALL-[1] ALL-[2] ALL-[3] ALL-[4] ALL-[5]

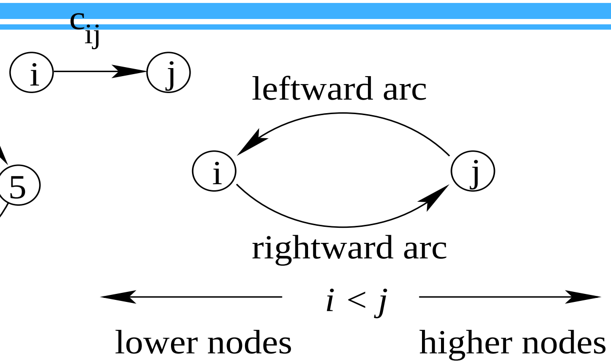
(5,2) (2,3) (1,4)

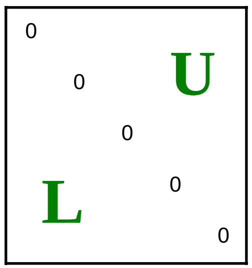
Definitions for Algorithm DLU

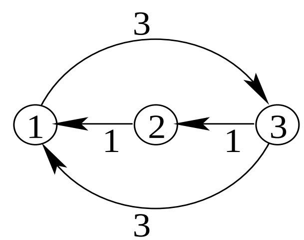




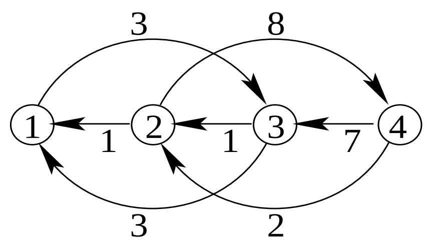
Original Graph



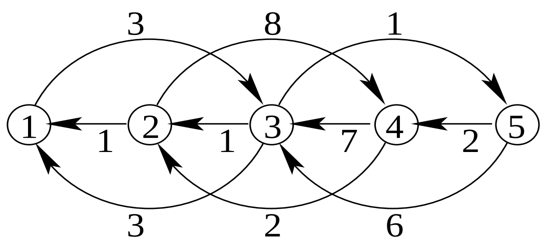




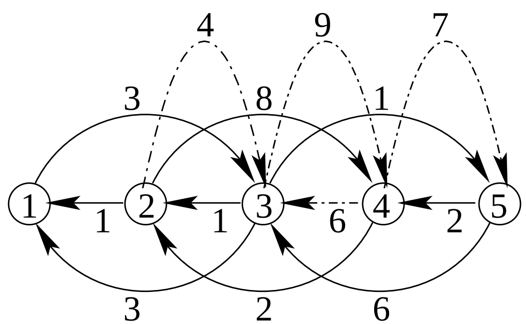
Subgraph H([1,3])



Subgraph H([1,3]U4)

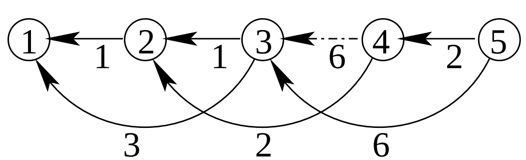


Subgraph H([1,4]U5)

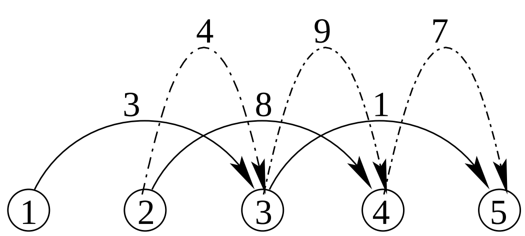


Augmented Graph G'

Dotted arcs have been added or modified



G'_L



G'_U

Algorithm $DLU_1(Q)$

- Get x^*_{st} for OD pairs (s,t): $s \geq k_0$ $t \geq j_0$ or $s \geq i_0$ $t \geq i_0$
  where $i_0 = \min_i s_i$, $j_0 = \min_i t_i$, $k_0 = \min_i \{\max\{s_i, t_i\}\}$

Algorithm ***$DLU_1(Q=\{(s_i,t_i),i=1,\ldots,q\})$***
begin
LU;
Acyclic_L(j_0);
Acyclic_U(i_0);
Reverse_LU(k_0);
end

0			X	
	0	X		
		0		
			0	
	X			0

$Q:=\{(s_i,t_i),i=1,\ldots,q\}$
$i_0=\min_i s_i$
$j_0=\min_i t_i$
$k_0=\min_i \{\max\{s_i,t_i\}\}$

$Q=\{(1,4),(2,3),(5,2)\}$
$i_0=1$
$j_0=2$
$k_0=\min \{4,3,5\}=3$

LU: $\forall$ (s,t) , get x^*_{st} in H([1,min{s,t}]) , construct Augmented Graph G'

Acyclic_L(j_0): $\forall$ (s,t) $s>t\geq j_0$, get x^*_{st} in H([1,s]) , compute shortest path in G'_L

Acyclic_U(i_0): $\forall$ (s,t) $t>s\geq i_0$, get x^*_{st} in H([1,t]) , compute shortest path in G'_U

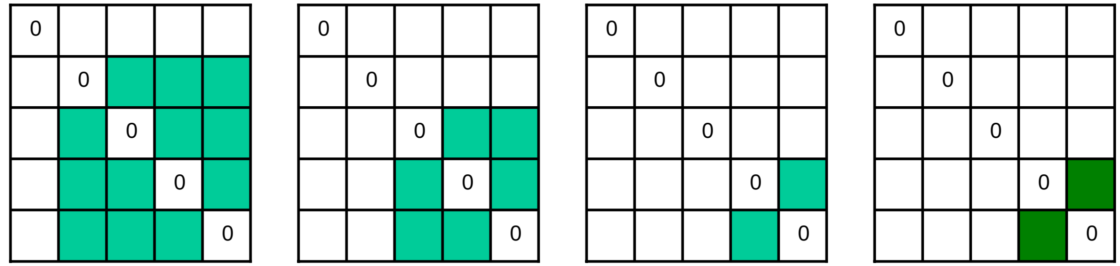
Reverse_LU(k_0): $\forall$ (s,t) satisfying $s\geq k_0$ $t\geq j_0$, or $s\geq i_0$ $t\geq i_0$
get x^*_{st} in H([1,max{s,t}] ∪ r), r=(max{s,t}+1),…,n
compare with x^*_{st} obtained in H([1,max{s,t}]), done!

Operations of Algorithm DLU_1

ALL-ALL

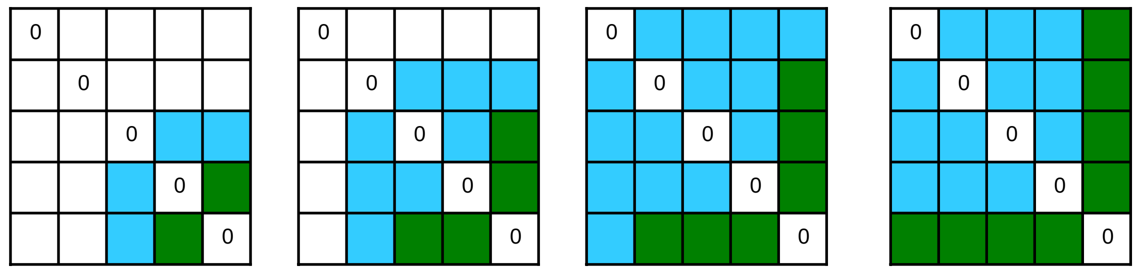
$i_0=1$, $j_0=1$, $k_0=2$

LU $\frac{n(n-1)(n-2)}{3}$



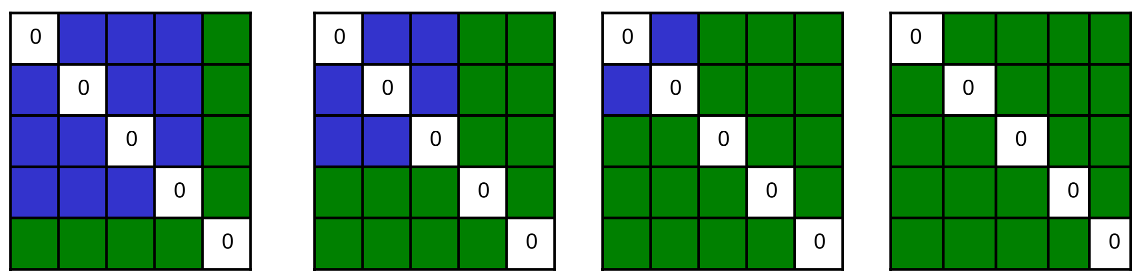
Acyclic_L
Acyclic_U

$\frac{n(n-1)(n-2)}{3}$



Reverse_LU

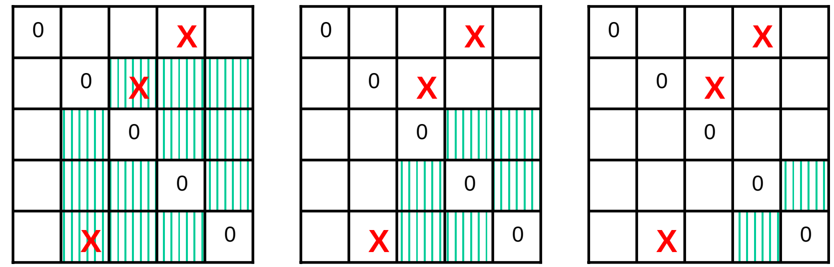
$\frac{n(n-1)(n-2)}{3}$



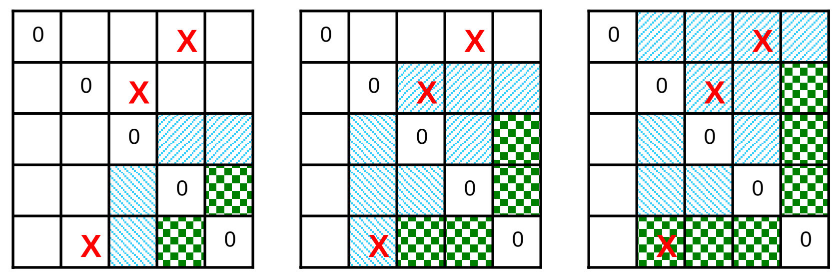
Example of Algorithm DLU_1

Q={(1,4),(2,3),(5,2)}
$i_0=1, j_0=2$
$k_0=min\ \{4,3,5\}=3$

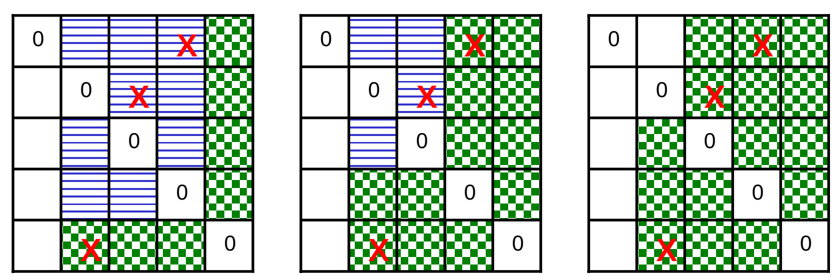
- x Requested OD pair
- Updated entries by *G_LU*
- Updated entries by *Acyclic_L*
- Updated entries by *Acyclic_U*
- Updated entries by *Reverse_LU*
- Optimal entries



(a) Procedure *G_LU*



(b) Procedure *Acyclic_L(2)* and ***Acyclic_U(1)***

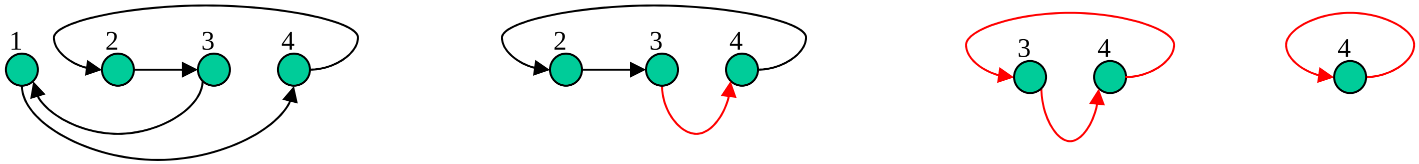


(c) Procedure *Reverse_LU(3)*

Properties of Algorithm DLU_1

Good:

- Efficient if only x^*_{st} are requested
  (if need to trace paths, set i_0=1)
- Work for general arc costs, can detect any negative cycle



- A good node ordering decreases fill-in arcs
  - Preprocessing (Markowitz's rule, MMD, MND..etc)
- Save ½ storage/computation for undirected graphs.
- Acyclic graphs: same as topological ordering

Bad:

- Redundant work on unrequested OD pairs
- Difficulty to trace shortest path

Algorithm DLU_2

Algorithm ***$DLU_2(Q=\{(s_i,t_i),i=1,\ldots,q\})$***
begin
  LU;
  for i=1~q
    Get_D(s_i,t_i);
    if $x^*_{si,ti} \neq \infty$ **&** need to trace path
      Get_P(s_i,t_i);
end

- attacks requested OD pairs directly
- fewer operations than DLU1
- easier to trace path

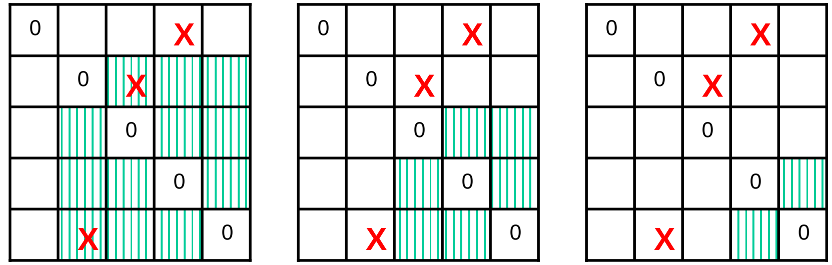
Procedure ***Get_D(s_i,t_i)***
begin
  Get_D_L(t_i);
  Get_D_U(s_i);
  Min_add(s_i,t_i);
end

Procedure ***Get_P(s_i,t_i)***
begin
  let $k:=succ_{si,ti}$
  while $k \neq t_i$ **do**
    Get_D(k,t_i);
    let k:=succsi,ti
end

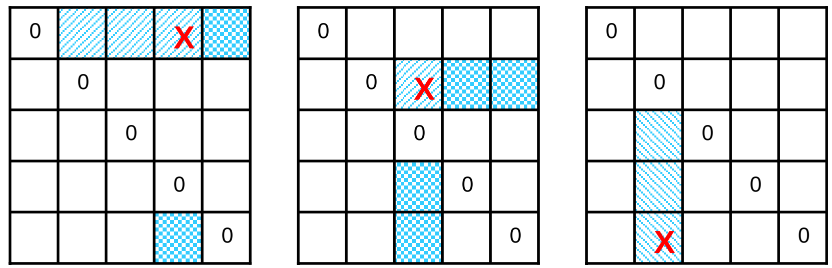
Example of Algorithm DLU_2

Q={(1,4),(2,3),(5,2)}

- Requested OD pair
- Updated entries by *G_LU*
- Updated entries by *Get_D_L*
- Updated entries by *Get_D_U*
- Updated entries by *Min_add*

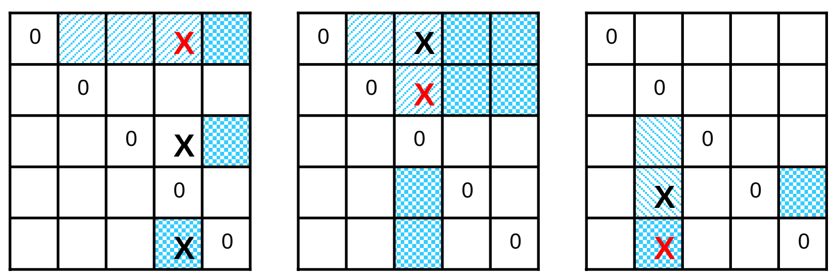


(a) Procedure *G_LU*



Get_D(1,4) ***Get_D(2,3)*** ***Get_D(5,2)***

(b) Procedure *Get_D(s,t)*



Get_P(1,4) ***Get_P(2,3)*** ***Get_P(5,2)***

1→3→5→4 ***2→1→3*** ***5→4→2***

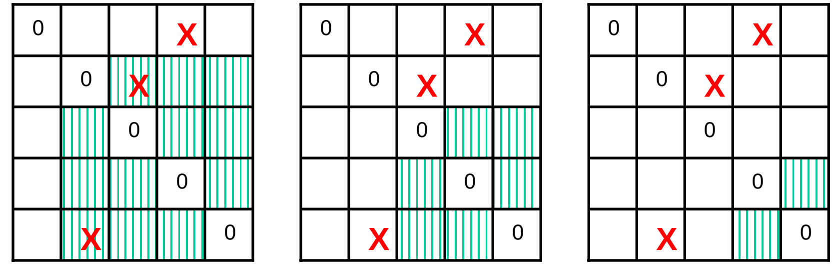
(c) Procedure *Get_P(s,t)*

Q={(1,4),(2,3),(5,2)}

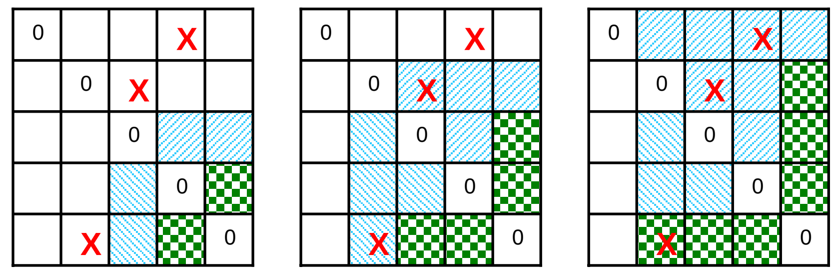
$i_0=1, j_0=2$

$k_0=min\ \{4,3,5\}=3$

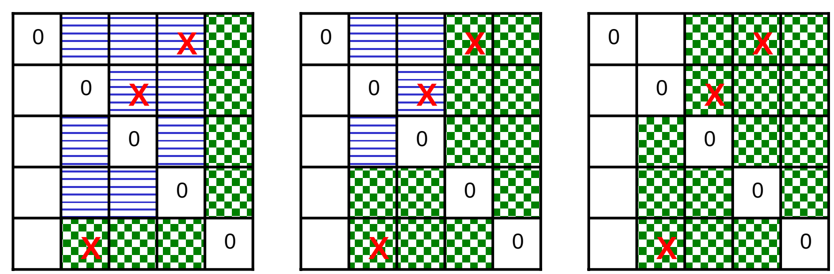
- x Requested OD pair
- Updated entries by *G_LU*
- Updated entries by *Acyclic_L*
- Updated entries by *Acyclic_U*
- Updated entries by *Reverse_LU*
- Optimal entries



(a) Procedure *G_LU*



(b) Procedure *Acyclic_L(2)* and ***Acyclic_U(1)***

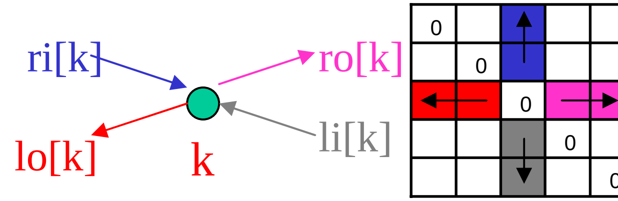


(c) Procedure *Reverse_LU(3)*

Example of Algorithm DLU_2

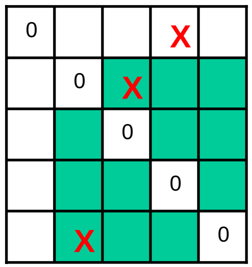
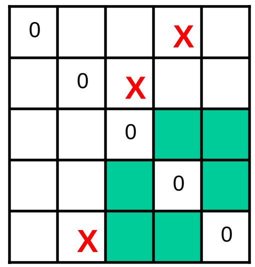
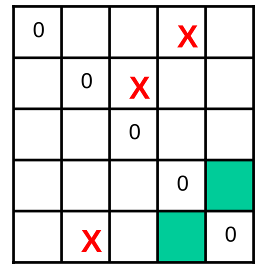
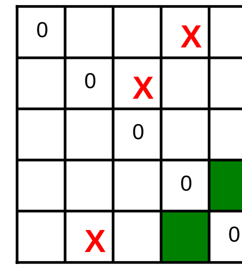
SOME-SOME

Q={(1,4),(2,3),(5,2)}



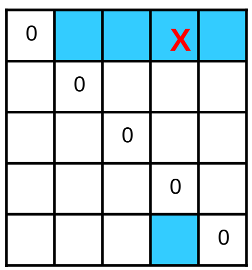
LU

O(Σ ro(k)li(k))

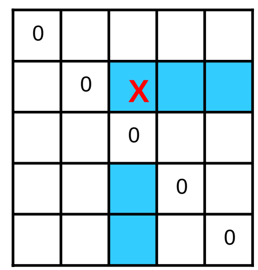





Get_D(1,4)

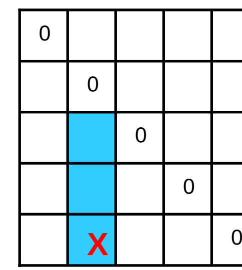
O(min{q*nnz(LU),n³})



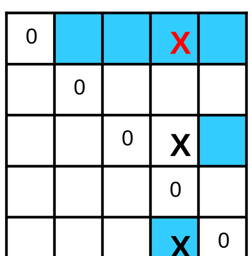
Get_D(2,3)



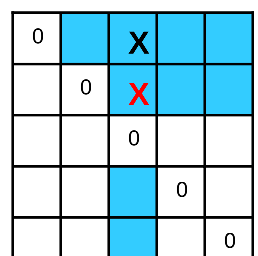
Get_D(5,2)



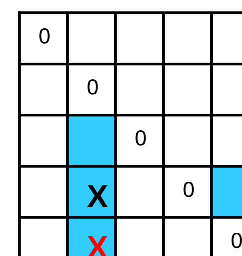
Get_P(1,4)
1-3-5-4



Get_D(2,3)
2-1-3



Get_D(5,2)
5-4-2



Implementation

- Preprocess: 8 pivot rules (MKZ(S,D),MMD,MND,…)
- SLU_1: sparse implementation of Carré's Algorithm
  - Code generation (e.g.: n=1000,m=6000,500MB->200MB->5MB->0MB!)
  - Bucket(slu11) , 1 heap(slu12) , 2 heaps(slu13)
- SLU_2: sparse implementation of Algorithm DLU_2
  - Bucket(slu21) , 1 heap(slu22)

Compare with: (Cherkassky et al. 1996 Math.Program.)

- gor1 , bfp , thresh , pape , two-q , dikh , dikbd , dikr , dikba

Test cases:

- 4 network generators: spgrid , sprand , netgen , gridgen
- 3 OD generators

Results & Conclusion

- Computational Results:
  - Depend on topology , node ordering , OD pairs
  - In general can not beat Cherkassky et al.
  - Markowitz's rule usually gives better ordering
  - Definitely much better than Floyd-Warshall Algorithm
- Conclusion:
  - Simple , efficient for SOM-SOME shortest paths
  - 'Ad hoc' code , exploits properties of graphs
  - Theoretically suitable for dense graphs
  - Need more storages ($O(n^2)$)
  - Complexity $O(\Sigma_{k=1\sim n} ro(k) li(k) + \min\{q^* nnz(LU), n^3\})$
  - Iterative methods(stability) vs. direct methods(sparsity)

Thank you

- Questions ?

- Contact

I-Lin Wang (王逸琳)

ilin@isye.gatech.edu

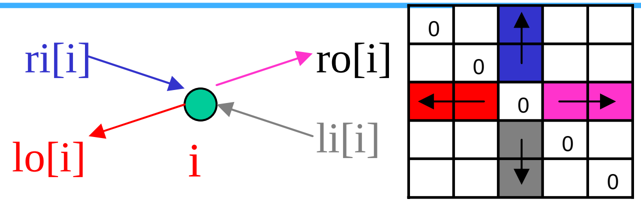
http://www.isye.gatech.edu/~ilin

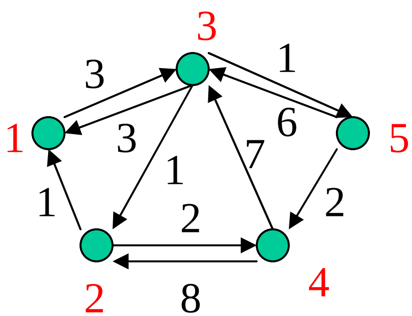
Industrial & System Engineering (ISyE)

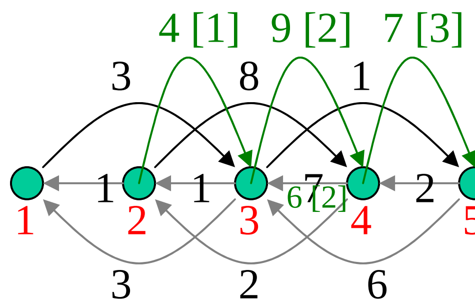
Georgia Institute of Technology (GA Tech)

Example of Algorithm SLU_1

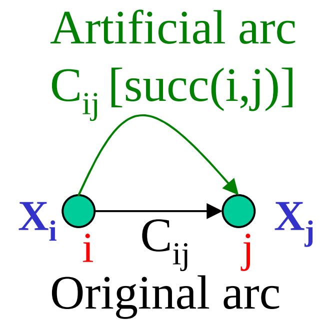
- Augmented Graph:
  - Graph obtained by **LU** operations



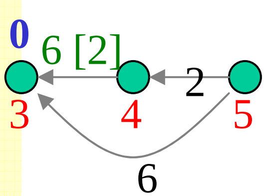




Augmented Graph

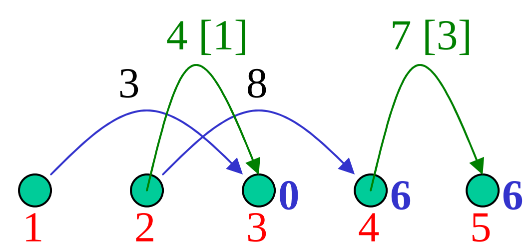


- e.g. ALL-[3]



Forward(3)

Node i	3	4	5
X_i	0	6	6
succ(i)	-	2	3



Backward(1)

Node i	1	2	4	5
X_i	3	4	6	6
succ(i)	3	1	2	3