A Sublinear Space, Polynomial Time Algorithm for Directed s-t Connectivity

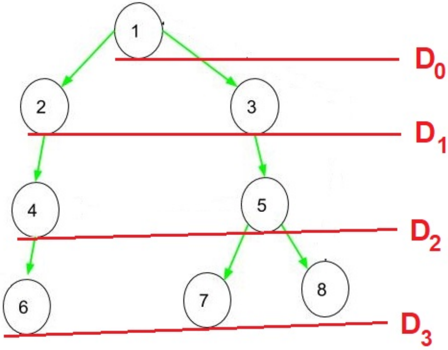
Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, Baruch Schieber

STCON in Directed Unique-Path Graphs

Sampath Kannan, Sanjeev Khanna, Sudeepa Roy

STCON = $\{(G, s, t) |$t is reachable from s in directed graph G$\}$
USTCON = $\{(G, s, t) |$s,t are connected in undirected graph G$\}$

- USTCON $\in$ L [Reingold,05]
- STCON $\in$ DSPACE($log^2 n$) [Savitch's theorem]
- BFS/DFS solves STCON using O(n log n) space and O(m+n) time.
- STCON can be solved using $\frac{n}{2^{\Theta(\sqrt{\log n})}}$ space and polynomial time. [BBRS,98]



D_i := Set of vertices in i-th level of BFS tree

Fix an integer λ.

Divide the levels $D_0, D_1, ..., D_{n-1}$ of the bfs tree rooted at s into λ equivalence classes $C_0, C_1, ..., C_{\lambda-1}$ where $C_i = \{v \in V(G)|$ dist(s,v) = i (mod λ)$\}$.

There exists some C_i with number of vertices $\leq \dfrac{n}{\lambda}$

Lemma

Every vertex reachable from s is at distance at most λ from $C_i \cup \{s\}$ $\forall i \in \{0, 1, ..., \lambda - 1\}$.

Proof.

Let v be a vertex reachable from s such that dist(s,v) = $\lambda q + r$
where $0 \leq q \leq \frac{n}{\lambda}, 0 \leq r \leq \lambda - 1$.
Consider the shortest path P from s to v.
If $r \geq i$, then let u be the vertex on P such that dist(s,u) = $\lambda q + i$. Otherwise let u be the vertex on P such that dist(s,u) = max$\{0, \lambda(q-1)+i\}$. Then $u \in C_i$ and dist(u,v) $\leq \lambda$. ∎

Algorithm BFS(G,s,t):

for i=0 to $\lambda - 1$:

  $C_i = \{s\}$

  for every vertex v at distance i from s:

    if $|C_i| < \frac{n}{\lambda}$:

      Add v to C_i

    else:

      i := i+1

  for q=1 to $\frac{n}{\lambda}$:

    S := φ

    for every vertex v at distance λ from C_i:

      if $|C_i| + |S| < \frac{n}{\lambda}$:

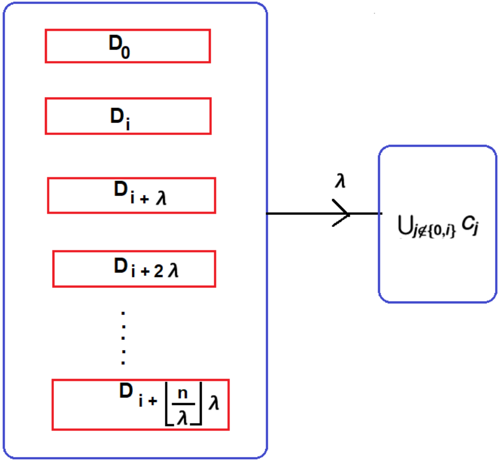
        Add v to S

      else:

        i := i+1

    $C_i := C_i \cup S$

  Check if t is within distance λ from C_i



Space complexity: $O(\frac{n \log n}{\lambda} + S_{PATH}(\lambda, n))$

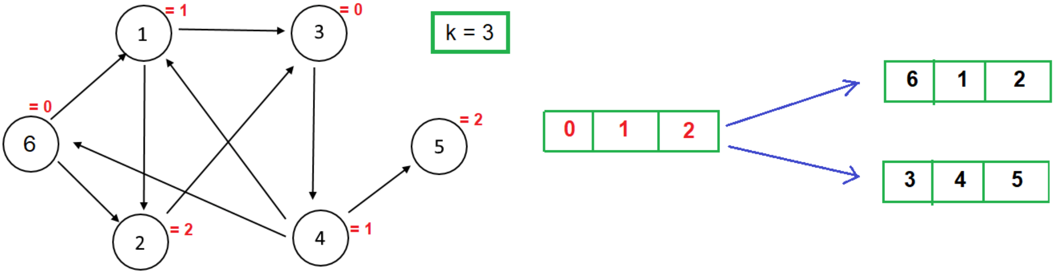
Time complexity: $O(\frac{n^3}{\lambda} . T_{PATH}(\lambda, n))$

$f(n)$**-bounded STCON:** Given an n-vertex directed graph G and two vertices s,t of G, check whether t is within distance f(n) of s

Idea:

Fix an integer k. We divide the vertices into k sets according to their vertex number modulo k.

Suppose we want to find all paths from s of length $\leq L = f(n)$. We can enumerate all (L+1)-digit k-ary numbers (s mod k, $a_1, a_2, ..., a_L$) and for each such number, find all L-length paths $(s, u_1, u_2, ..., u_L)$ such that $(s, u_1, u_2, ..., u_L)$ mod k = (s mod k, $a_1, a_2, ..., a_L$).



Algorithm for L-bounded STCON(G,s,t,L):

Set $V_0 := \{s\}$

for all (L+1)-digit k-ary number (s mod k, $a_1, a_2, ..., a_L$):

  for i=1 to L:

    for all $u \in V_{i-1}$ AND $v = a_i$ mod k:

      if (u,v) is an edge:

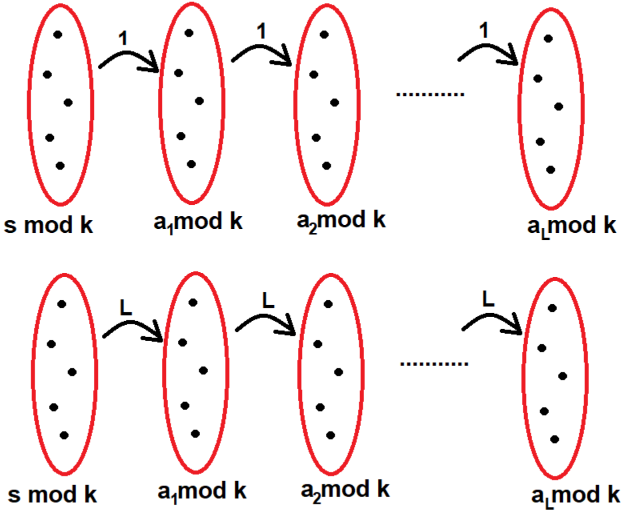
        Add v to V_i $\backslash\backslash V_i$ contains all vertices reachable from s using a path that maps to (s mod k, $a_1, a_2, ..., a_L$) modulo k.

        if v==t:

          Return "Connected!"

        Erase V_{i-1} from work tape

We don't need to maintain the list of vertices V_i. Instead we can maintain a $\left\lceil \frac{n}{k} \right\rceil$-bit vector where j-th entry of the vector will be equal to 1 if $(kj + a_i) \in V_i$. Then the algorithm will run in O($\frac{n}{k}$+ L log k) space and O($k^L L.\frac{n^2}{k^2}$) time.


1
1
1
....
s mod k
a1mod k
a2mod k
aLmod k
L
L
L
....
s mod k
a1mod k
a2mod k
aLmod k

While computing V_i, instead of looking for vertices at distance 1 from V_{i-1}, we can look for vertices at distance L^{r-1} recursively and that would give us an algorithm for L^r-bounded STCON.

Short paths recursive algorithm:

SPR(G, L, r, s mod k, t mod k, V_s): \\V_s is a subset of vertices with vertex number = s mod k

if r=0:

    for all $u \in V_s$ and $v = t$ mod k:

        if (u,v) is an edge:

            add v to V_t

else:

    $V_0 := V_s$

    for all (L+1)-digit k-ary number (s mod k, $a_1, a_2, ..., a_L$):

        for i=1 to L:

            V_i = SPR(G, L, r-1, a_{i-1} mod k, a_i mod k, V_{i-1})

            Erase V_{i-1} from work tape

Return $V_t \cap V_L$

Then $S_{PATH}(n, L^r) = O(\frac{n}{k} +$ L log k) + $S_{PATH}(n, L^{r-1}) \implies S_{PATH}(n, L^r) = O(r(\frac{n}{k} +$ L log k))

and $T_{PATH}(n, L^r) = O(k^L L).T_{PATH}(n, L^{r-1})$ where $T(n, 1) = \frac{n^2}{k^2}$

$\implies T_{PATH}(n, L^r) = O((k^L L)^r.\frac{n^2}{k^2})$

If we take $\lambda = L^r$, then space complexity of the first algorithm is

$S(n) = O(\frac{n \log n}{L^r} + r(\frac{n}{k} +$ L log k)) and time complexity is

T(n) = O$(\frac{n^3}{L^r}.k^{Lr}L^r.\frac{n^2}{k^2}) = O(n^5 k^{Lr-2})$

Taking L=4, r = $\sqrt{\text{log n}}, k = 2^{\Theta(\sqrt{\text{log n}})}$ gives us $S(n) = \frac{n}{2^{\Theta(\sqrt{\log n})}}$ and $T(n) = n^{O(1)}$.

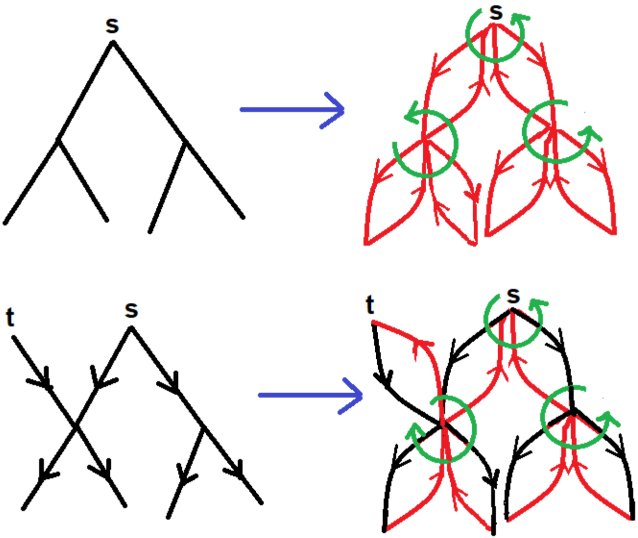
- STCON in regular digraphs can be solved using O(log n) space. [RTV,06]
- STCON in planar DAGs with single source can be solved using O(log n) space. [ABCDR,06]
- STCON in directed unique-path graphs is solvable using $O(n^{\epsilon}\log n)$ space and $O(n^{\frac{1}{\epsilon}})$ time. [KKR,08]

Definition

A directed graph G is a unique-path graph w.r.t. a source vertex s if there is at most one simple path from s to every vertex $v \in V(G)$. For every vertex x, $N^{+}(x) = \{v \in V(G) | \text{ (x,v) } \in E(G)\}$ and $N^{-}(x) = \{v \in V(G) | \text{ (v,x) } \in E(G)\}$.

Remark

DFS-tree of a unique-path graph doesn't have any forward edges or cross edges.



$\forall v \in V(G), next_v : N(v) \to N(v)$ is a bijective function.
Edge successor function is defined as succ(u,v) = (v,$next_v(u)$).
Then the sequence e,succ(e), $succ^2(e)$,$succ^3(e)$,.... is a dfs traversal of the tree.

Algorithm:

Step 1:
Current vertex x is either a newly discovered vertex or DFS has backtracked to x from some $v \in N^+(x)$
Initially x = s
if x is a newly discovered vertex:
    Ask the oracle for the first vertex in $N^+(x)$
else if DFS has backtracked from v to x:
    Ask the oracle for successor of v in $N^+(x)$
Step 2:
if oracle says there are no more vertices in $N^+(x)$:
    Perform the **backtrack step** find Parent[x], set current vertex x := Parent[x] and go back to step 1
else if oracle returns a vertex $y \in N^+(x)$:
    Perform the **discovery step** to check whether (x,y) is a back-edge.
    if (x,y) is a back edge: \\This is the only bad case
        then ask the oracle for successor of y in $N^+(x)$ and go back to step 2
    else:
        if y==t:
            Return "t is reachable from s"
        else:
            Set current vertex x:= y and go back to step 1

For a unique-path graph, if we store the entire active path, we can perform the backtrack step and the discovery step easily.
L-bounded DFS is a DFS which backtracks whenever length of the active path exceeds L.

Lemma

In a unique-path graph, L-bounded DFS can be implemented in O(L log n) space and O(n+mL) time.

O($\sqrt{n}$ log n)-space algorithm for DFS on unique-path graphs:

Maintaining landmark vertices: In stead of storing the entire active path, we will store landmark vertices which are a few evenly spaced vertices on the active path from s to current vertex x. The landmark vertices will be $s = z_0, z_1, ..., z_{p-1}, z_p = x$ where for $0 \leq i \leq p - 1$, z_i is the vertex at distance $i\lfloor\sqrt{n}\rfloor$ from s on the current active path.

Backtrack step from current vertex x:

for all $v_i \in N^-(x)$:

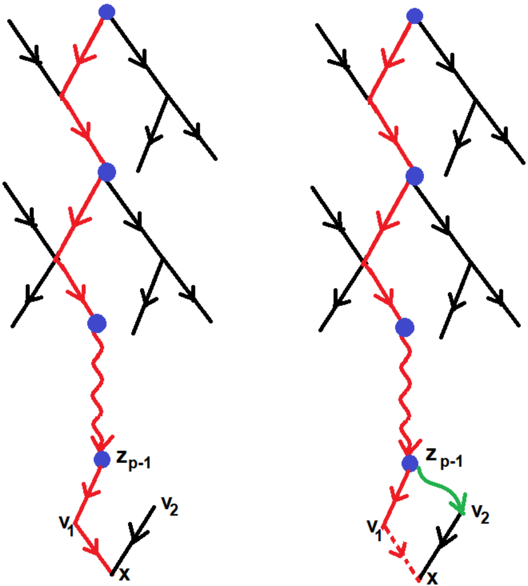
  Perform a $\sqrt{n}$-bounded DFS from z_{p-1} in $G \backslash (v_i, x)$

  If x isn't reached in this DFS:

    Return v_i as Parent[x]

Space: O($\sqrt{n}$ log n)

Time: O(mn + $m^2\sqrt{n}$)



Discovery step from current vertex x:

Oracle has returned a vertex y and we have to check whether (x,y) is a back edge i.e. whether y has been visited before.

Let Z(y) be the set of landmark vertices reachable from y in $\sqrt{n}$-bounded DFS.

If y has been visited before, then there exist two landmark vertices z_{j-1} and z_j such that y lies between these two landmark vertices on the active path.

Lemma

$z_k \notin Z(y)$ for k > j i.e. z_j is the landmark vertex of highest index in Z(y).

Proof.

Otherwise there are two distinct simple paths from y to z_k - one given by the $\sqrt{n}$-bounded DFS and other one is along the active path via z_j. ∎

Discovery step for edge (x,y):

if $y \in \{z_0, z_1, ..., z_{p-1}\}$:
    Return "y has been visited before"
Perform $\sqrt{n}$-bounded DFS from y
if $Z(y) == \varphi$:
    Return "y has not been visited before"
Let j be the highest index such that $z_j \in Z(y)$
if $j == 0$:
    Return "y has not been visited before"
if $j \leq p - 1$:
    Perform $\sqrt{n}$-bounded DFS from z_{j-1} and terminate as soon as z_j or y is reached
    if y is reached:
        Return "y has been visited before"
    else:
        Return "y has not been visited before"
if j==p:
    Perform $2\sqrt{n}$-bounded DFS from z_{p-2} and terminate as soon as x or y is reached
    if y is reached:
        Return "y has been visited before"
    else:
        Return "y has not been visited before"

Space: O($\sqrt{n}$ log n)
Time: O(mn + $m^2\sqrt{n}$)
Hence we get a O($\sqrt{n}$ log n) space and polynomial time algorithm for STCON in unique-path graphs.

This can be improved to O(n^ϵ) space and $n^{O(\frac{1}{\epsilon})}$ time algorithm for all $\epsilon > 0$.