Function: PROCESS-STATE ()

```
X = MIN-STATE( )
if X = NULL then return -1
k_old = GET-KMIN( ); DELETE(X)
if k_old < h(X) then
  for each neighbor Y of X:
    if h(Y) ≤ k_old and h(X) > h(Y) + c(Y, X) then
      b(X) = Y; h(X) = h(Y) + c(Y, X)
if k_old = h(X) then
  for each neighbor Y of X:
    if t(Y) = NEW or
      (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) or
      (b(Y) ≠ X and h(Y) > h(X) + c(X, Y)) then
      b(Y) = X; INSERT(Y, h(X) + c(X, Y))
else
  for each neighbor Y of X:
    if t(Y) = NEW or
      (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) then
      b(Y) = X; INSERT(Y, h(X) + c(X, Y))
    else
      if b(Y) ≠ X and h(Y) > h(X) + c(X, Y) then
        INSERT(X, h(X))
      else
        if b(Y) ≠ X and h(X) > h(Y) + c(Y, X) and
          t(Y) = CLOSED and h(Y) > k_old then
          INSERT(Y, h(Y))
return GET-KMIN( )
```

If X is current robot position, success

Path through X is no longer optimal due to new information. Go through Y

See if neighbors of X can go through new lower cost X due to updated information. Also, update out-of-date costs to each neighbor of X.

Propagate changes to *NEW* states and descendents of X.

If change in X can lower costs in non-descendent states, queue for processing

If path cost of X can be lowered through neighbor, queue Y for processing

Function: PROCESS-STATE ()

```
X = MIN-STATE( )
if X = NULL then return -1
k_old = GET-KMIN( ); DELETE(X)
```

Select X in OPEN with mininum k

If OPEN empty, then failure

Get min value of k on OPEN, set t(X)=CLOSED

Path through X no longer optimal (RAISE state)

```
if k_old < h(X) then
  for each neighbor Y of X:
    if h(Y) ≤ k_old and h(X) > h(Y) + c(Y,X) then
      b(X) = Y; h(X) = h(Y) + c(Y,X)
```

If cheaper to go thru nhbr Y, then relink path from X

Path through X is still optimal

```
if k_old = h(X) then
  for each neighbor Y of X:
    if t(Y) = NEW or
      (b(Y) = X and h(Y) ≠ h(X) + c(X,Y)) or
      (b(Y) ≠ X and h(Y) > h(X) + c(X,Y)) then
      b(Y) = X; INSERT(Y, h(X) + c(X,Y))
```

Nhbr Y is unvisited

Cost is out of date

Path thru Y is higher cost than thru X

Relink Y's path thru X, put on OPEN

A new cheaper path may have been found in sensory update

```
else
  for each neighbor Y of X:
    if t(Y) = NEW or
      (b(Y) = X and h(Y) ≠ h(X) + c(X,Y)) then
      b(Y) = X; INSERT(Y, h(X) + c(X,Y))
    else
      if b(Y) ≠ X and h(Y) > h(X) + c(X,Y) then
        INSERT(X, h(X))
      else
        if b(Y) ≠ X and h(X) > h(Y) + c(Y,X) and
          t(Y) = CLOSED and h(Y) > k_old then
          INSERT(Y, h(Y))
return GET-KMIN( )
```

Nhbr Y is unvisited

Cost is out of date

Relink Y's path thru X, put on OPEN

This step prevents loops in plan

Path thru Y may be lower cost alternative

Put Y on OPEN for later processing

Originally stated D* Algorithm

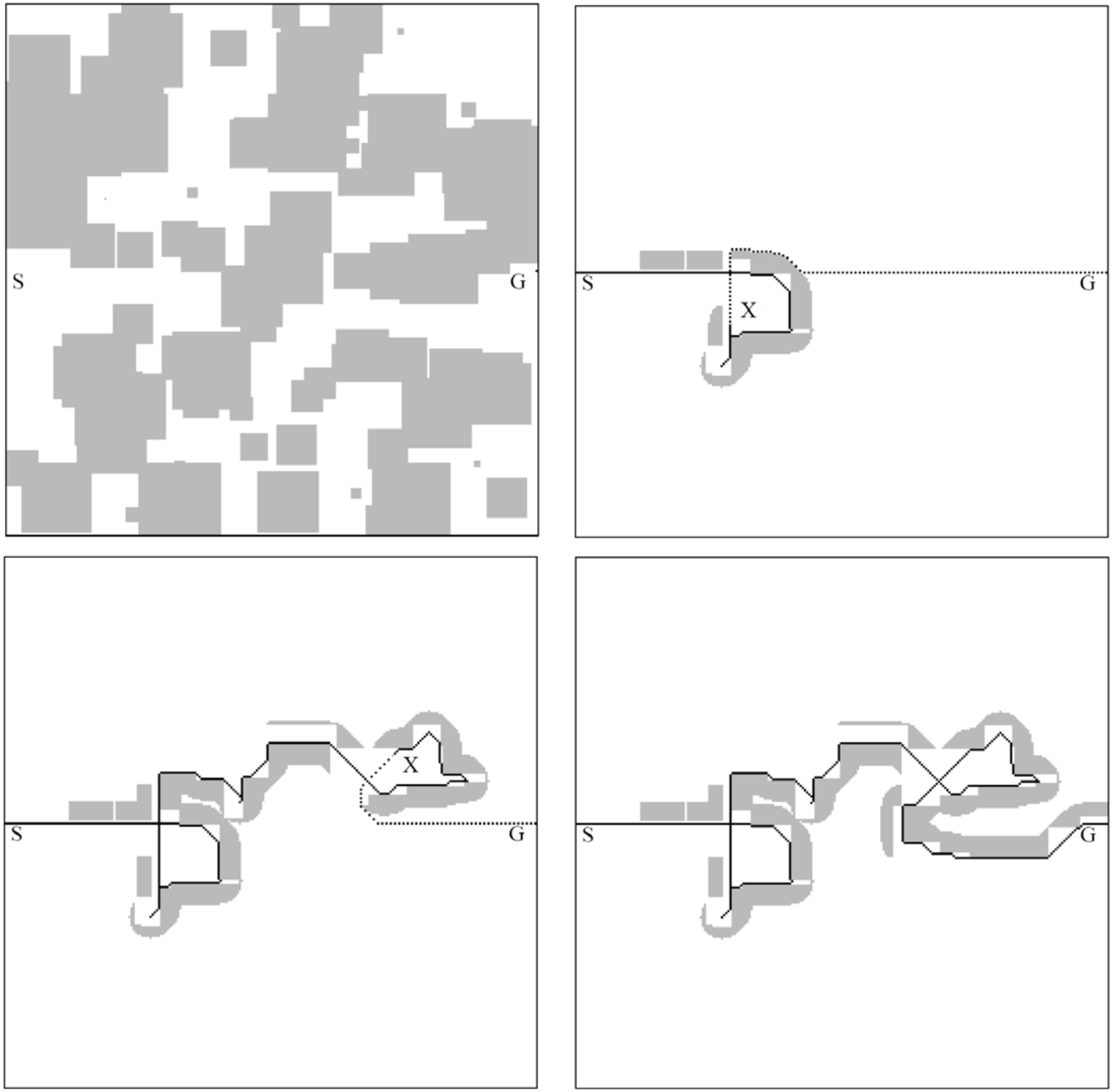
```
h(G)=0;
do
{
        k_min=PROCESS-STATE();
}while(k_min != -1 && start state not removed from open list);
if(k_min == -1)
  { goal unreachable; exit;}
else{
    do{

        do{
                trace optimal path();
          }while ( goal is not reached && map == environment);

        if ( goal_is_reached)
        { exit;}
        else
        {
            Y= State of discrepancy reached trying to move from some State X;
            MODIFY-COST(Y,X,newc(Y,X));
            do
            {
                k_min=PROCESS-STATE();
            }while(k_min< h(X) && k_min != -1);
            if(k_min==-1)
                exit();
        }
    }while(1);
}
```

Function: MODIFY-COST (X, Y, cval)

L1 $c(X, Y) = cval$

L2 if $t(X) = CLOSED$ then $INSERT(X, h(X))$

L3 return $GET-KMIN(\ )$


S
G
S
X
G
S
X
G
S
G