A Fabrication-Oriented Remeshing Method for Auxetic Pattern Extraction Supplementary Material

Extraction of auxetic patterns from a given quad mesh is shown in full detail in this supplementary document. We use pseudo-codes for reproducibility and the corresponding execution traces for further clarifications. As terminology, we name each hexagon in the auxetic pattern as a bow tie and the extracted auxetic pattern structure as a bow tie mesh in the sequel.

The extraction procedure of auxetic patterns is shown in Algorithm 1. This algorithm searches and extracts auxetic patterns on given quad mesh starting from given quad and returns the resulted structure.

Algorithm 1 SEARCH(QuadMesh, StartingQuad)

Input: Quad mesh and starting quad ID
Output: Bow tie mesh form of input quad mesh

```
1: BowTieMesh = empty set of bow ties, their points and their edges
2: QuadNeighbors = quads having a common edge with StartingQuad
3: Write QuadNeighbors
4: Read PairQuad
5: FINDBOWTIES(BowTieMesh, QuadMesh, StartingQuad, PairQuad)
6: for each quad Q ∈ QuadMesh do
7:    if Q.IsMatched then
8:       QuadNeighbors = quads having a common edge with Q
9:       for each neighbor quad NQ ∈ QuadNeighbors do
10:         if !NQ.IsMatched and Q is the only neighbor of NQ which is matched
            then
11:            NewStartingQuad = NQ
12:            NewPairQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, New-
               StartingQuad , Q)
13:            if NewPairQuad ≠ null then
14:               FINDBOWTIES(BowTieMesh, QuadMesh, NewStartingQuad, New-
                  PairQuad)
15:               break
16:            end if
17:         end if
18:      end for
19:   end if
20: end for
21: FINDINTERNALBOWTIES(BowTieMesh, QuadMesh, StartingQuad)
22: for each quad Q ∈ QuadMesh do
```

```
    if !Q.IsInternalMatched and Q does not have any internal matched neighbor then
      NewStartingQuad = Q
      FINDINTERNALBOWTIES(BowTieMesh, QuadMesh, NewStartingQuad)
    end if
  end for
  return BowTieMesh
```

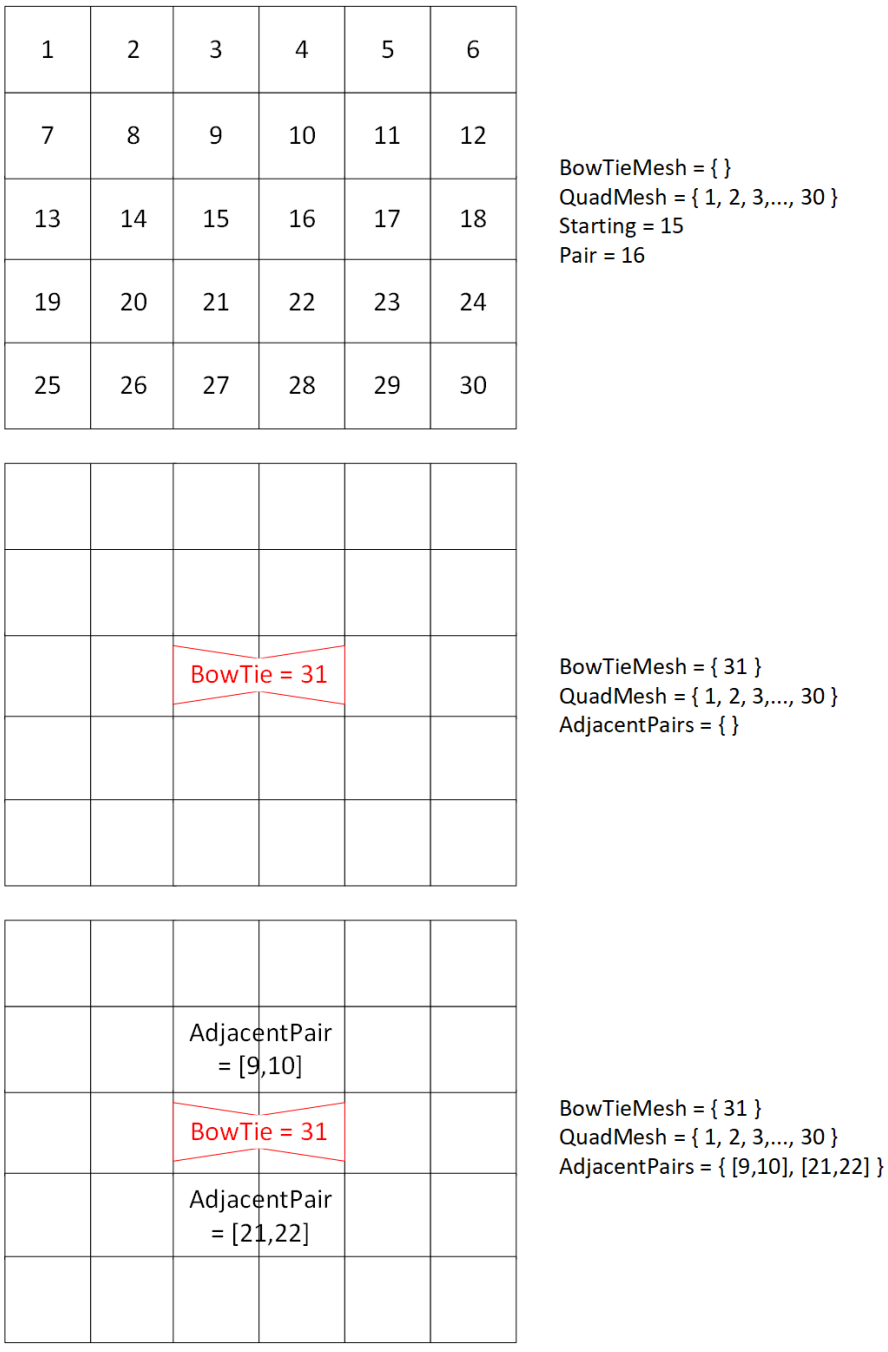
The SEARCH algorithm uses FINDBOWTIES algorithm to find bow ties and insert them into bow tie mesh structure. How to find, create and insert bow ties starting from a given quad pair is shown in Algorithm 2. This algorithm gets the quad pair, tries to create and insert a bow tie from this pair, if it succeeds then it tries to find out all bow ties can be created from neighbor quad pairs of this pair recursively, until it cannot find any bow tie. Execution of this algorithm is shown in Figure 1.

Algorithm 2 FINDBOWTIES(BowTieMesh, QuadMesh, StartingQuad, PairQuad)

Input: Bow tie mesh for inserting founded bow ties, quad mesh for searching bow ties, starting quad and starting quad's pair quad for creating bow tie and finding new bow ties

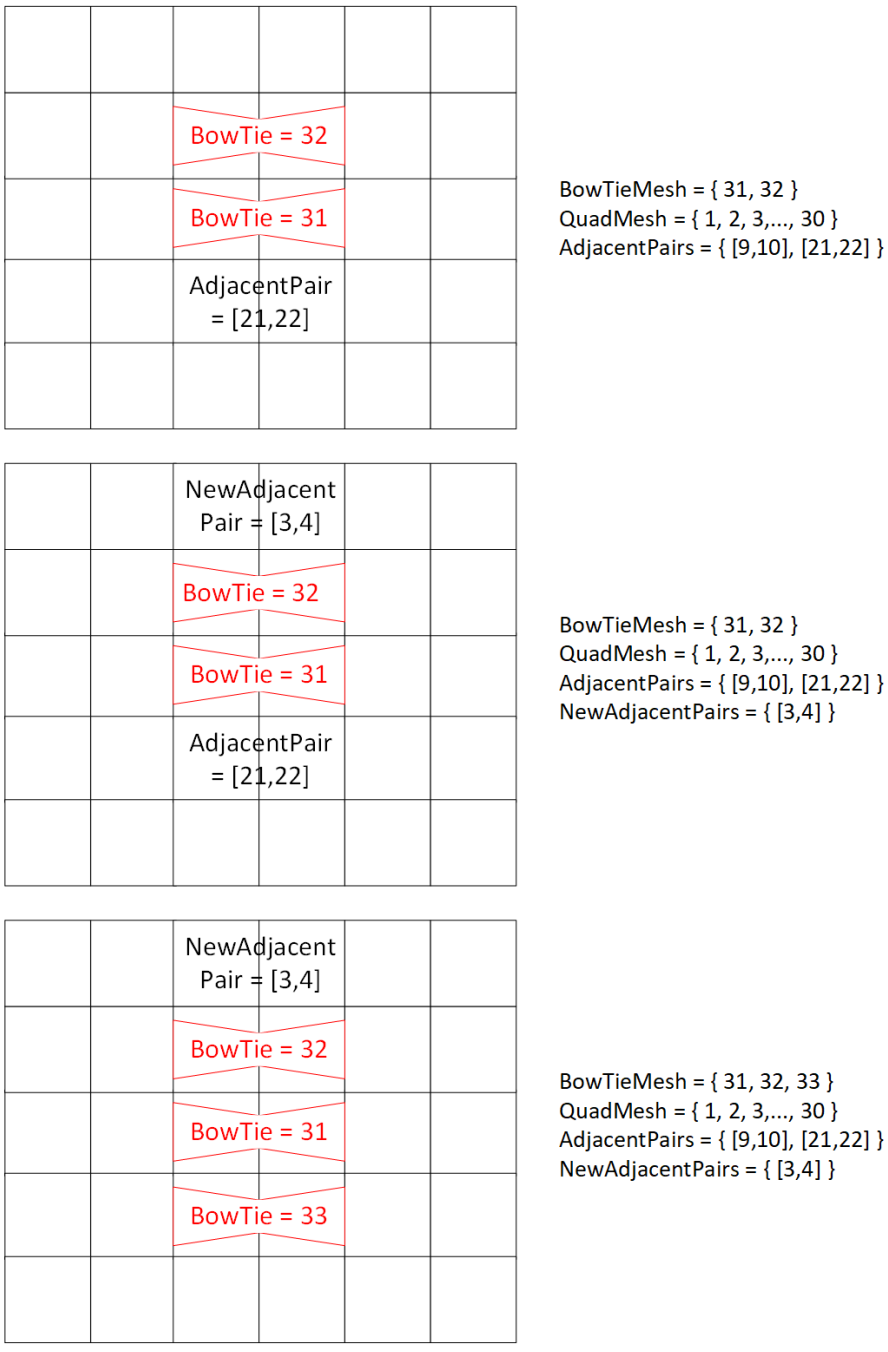
```
INSERTBOWTIE(BowTieMesh, QuadMesh, StartingQuad, PairQuad)
if bow tie was inserted from StartingQuad and PairQuad then
  AdjacentPairs = empty set of adjacent pairs of quads
  AdjacentPairs = GETADJACENTPAIRS(QuadMesh, Starting, Pair)
  while AdjacentPairs ≠ empty do
    NewAdjacentPairs = empty set of adjacent pairs of quads
    for each adjacent pair P ∈ AdjacentPairs do
      INSERTBOWTIE(BowTieMesh, QuadMesh, P.First, P.Second)
      if bow tie was inserted from P.First and P.Second then
        Add [P.First, P.Second] to FoundedBowTieTriPairs
        AdcajentPairsFounded = GETADJACENTPAIRS(QuadMesh, P.First,
        P.Second)
        Add AdcajentPairsFounded to NewAdjacentPairs
      end if
    end for
    AdjacentPairs = NewAdjacentPairs
  end while
  NewStartingQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, StartingQuad,
  PairQuad)
  if NewStartingQuad ≠ null then
    NewPairQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, NewStart-
    ingQuad , StartingQuad)
    if NewPairQuad ≠ null then
      FINDBOWTIES(BowTieMesh, Quads, NewStartingQuad , NewPairQuad)

      NewStartingQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, PairQuad,
      StartingQuad)
      if NewStartingQuad ≠ null then
```

```
24:             NewPairQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, New-
                StartingQuad , PairQuad)
25:             if NewPairQuad ≠ null then
26:               FINDBOWTIES(BowTieMesh, QuadMesh, NewStartingQuad, New-
                  PairQuad)
27:             end if
28:           end if
29:         end if
30:       end if
31:   end if
```



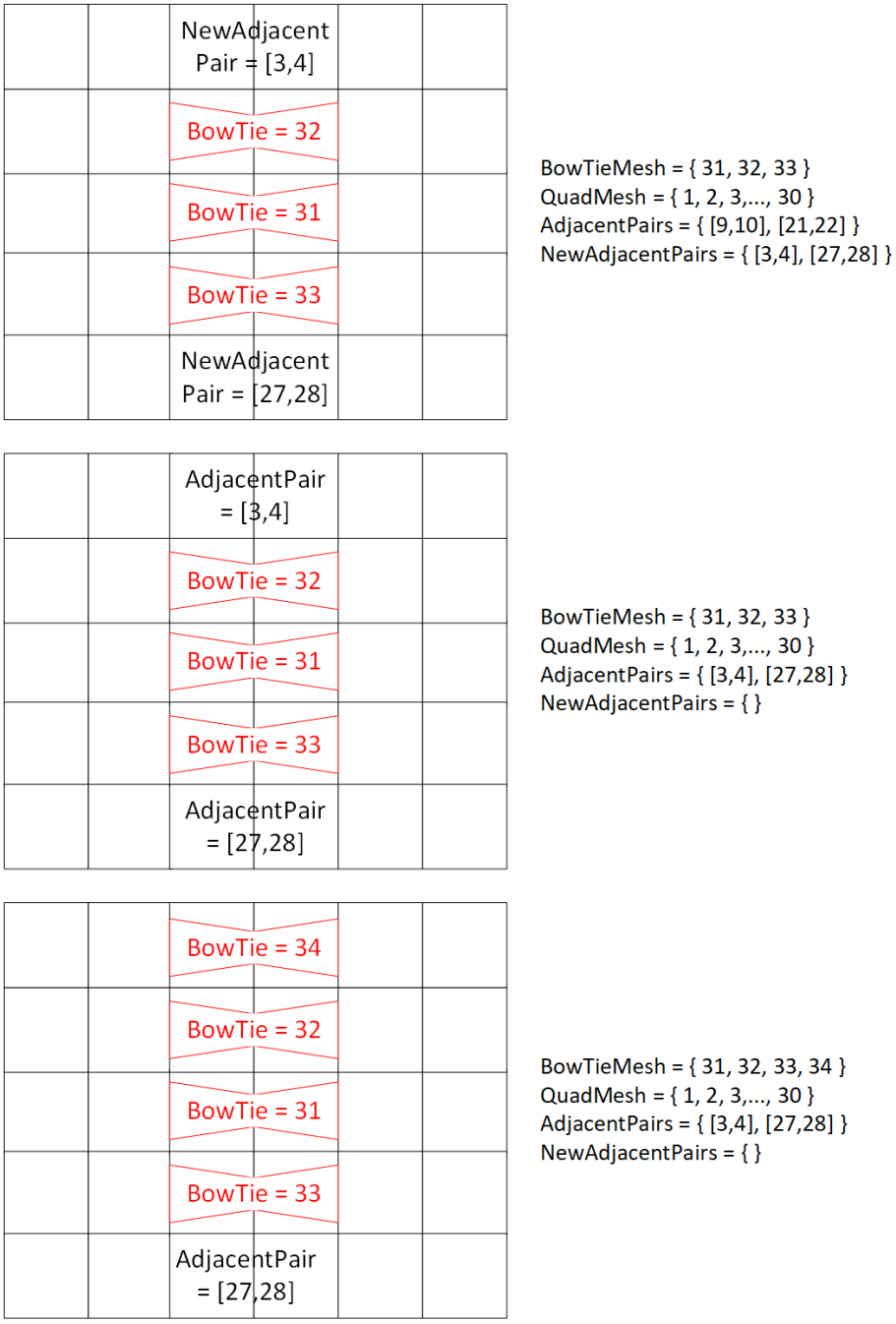
(a)

Figure 1: Execution steps of the FINDBOWTIES algorithm



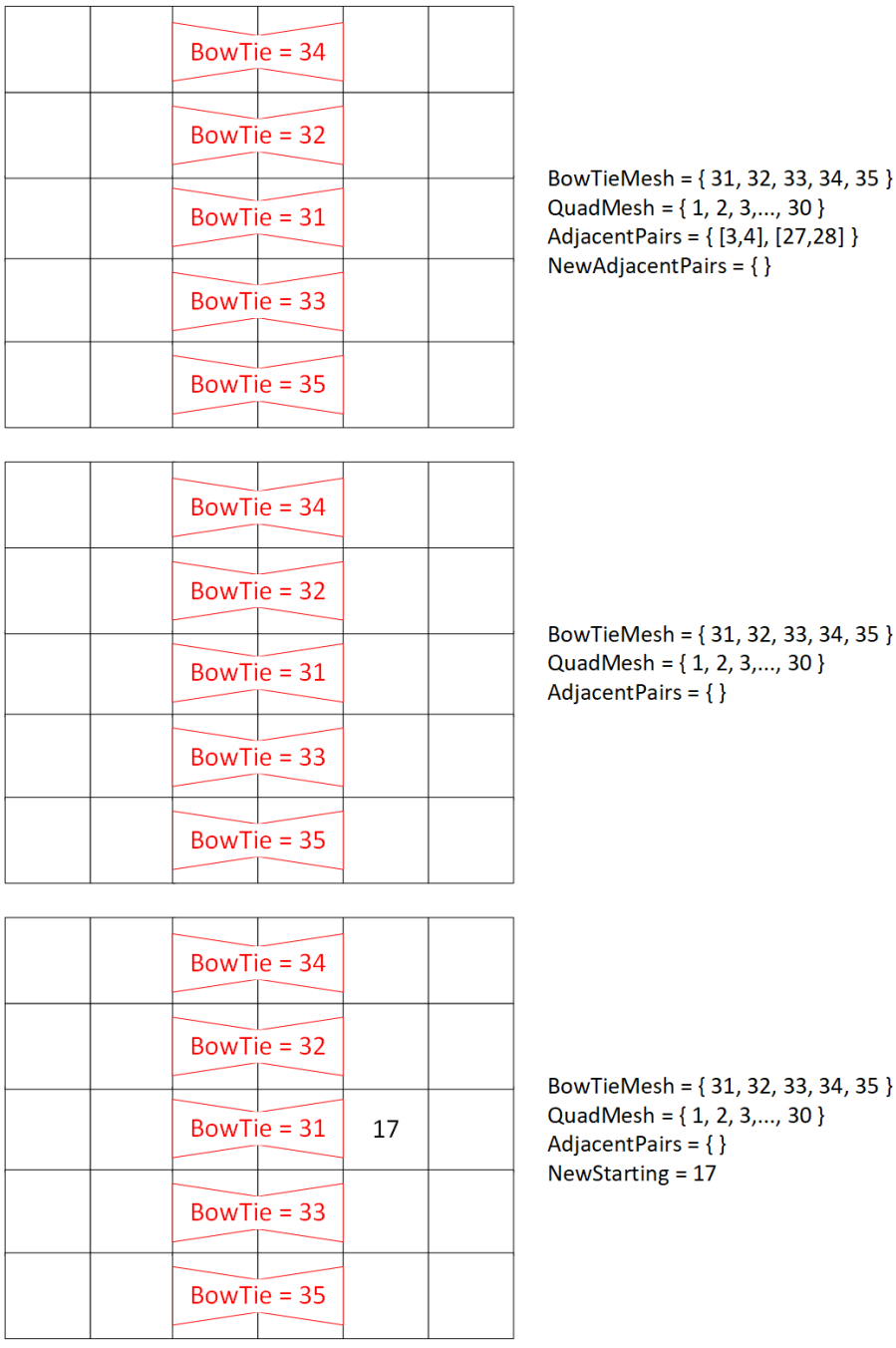
(b)

Figure 1: Execution steps of the FINDBOWTIES algorithm (cont.)



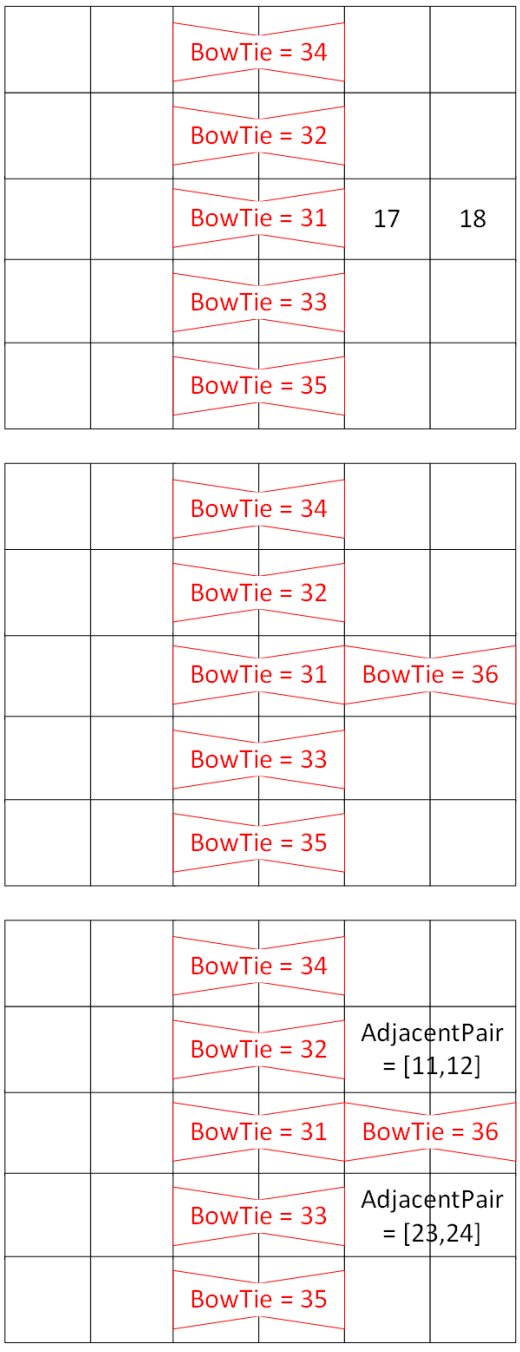
(c)

Figure 1: Execution steps of the FINDBOWTIES algorithm (cont.)



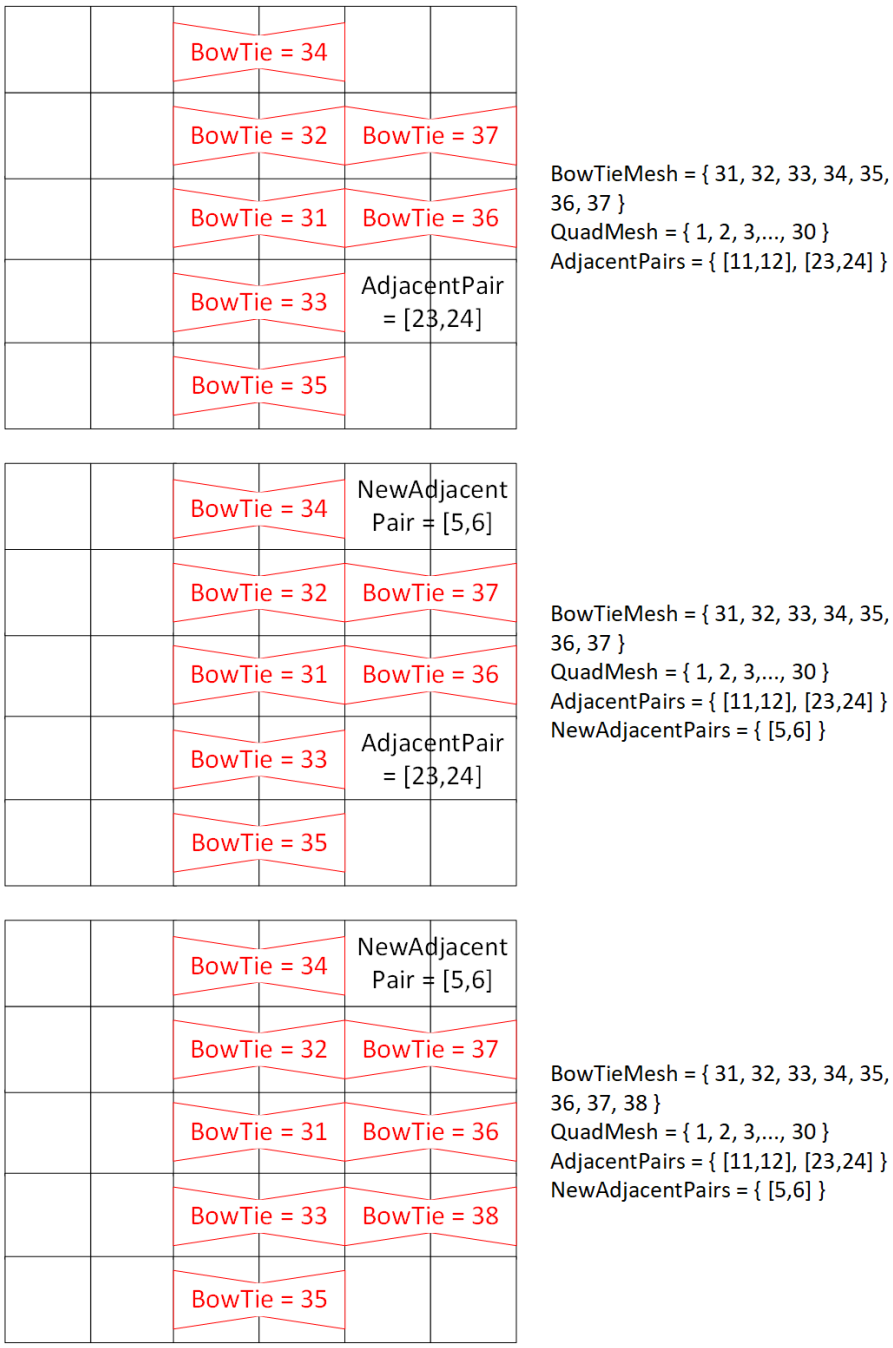
(d)

Figure 1: Execution steps of the FINDBOWTIES algorithm (cont.)



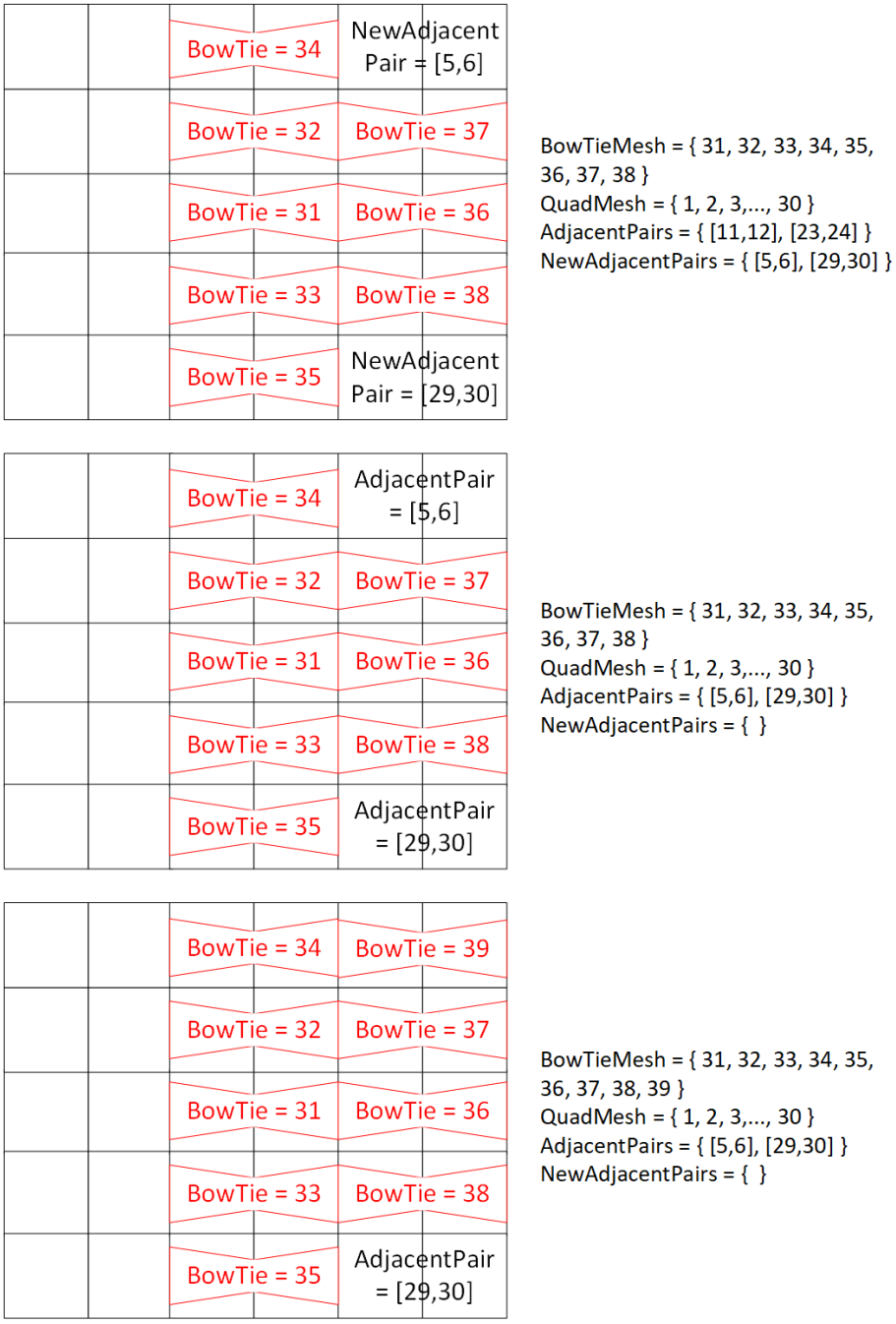
(e)

Figure 1: Execution steps of the FINDBOWTIES algorithm (cont.)



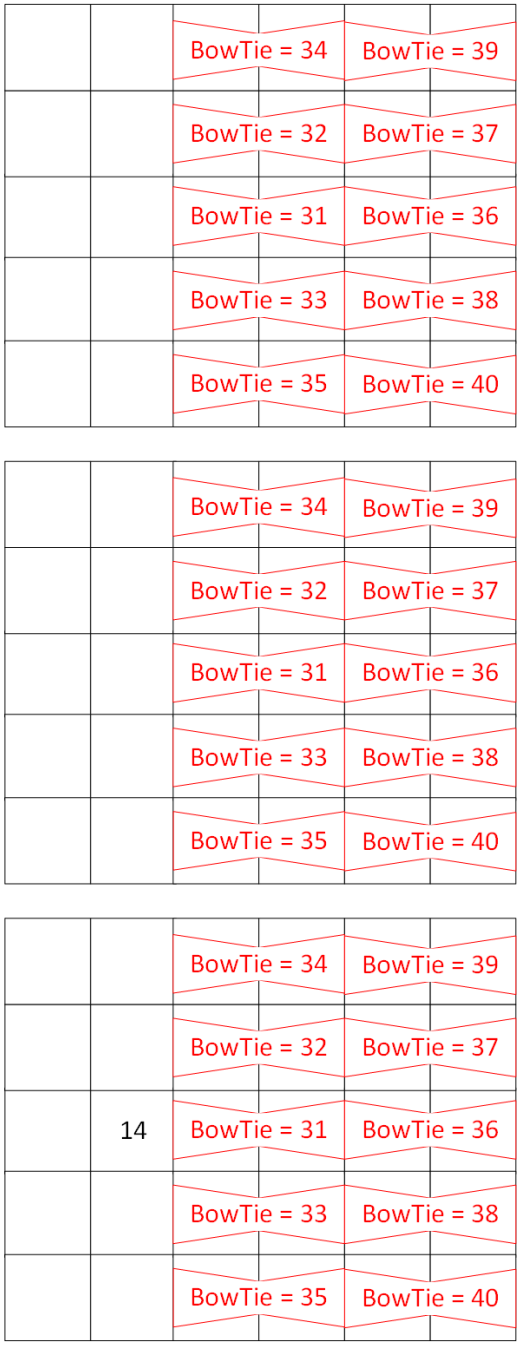
(f)

Figure 1: Execution steps of the FINDBOWTIES algorithm (cont.)



(g)

Figure 1: Execution steps of the FINDBOWTIES algorithm (cont.)



(h)

Figure 1: Execution steps of the FINDBOWTIES algorithm (cont.)

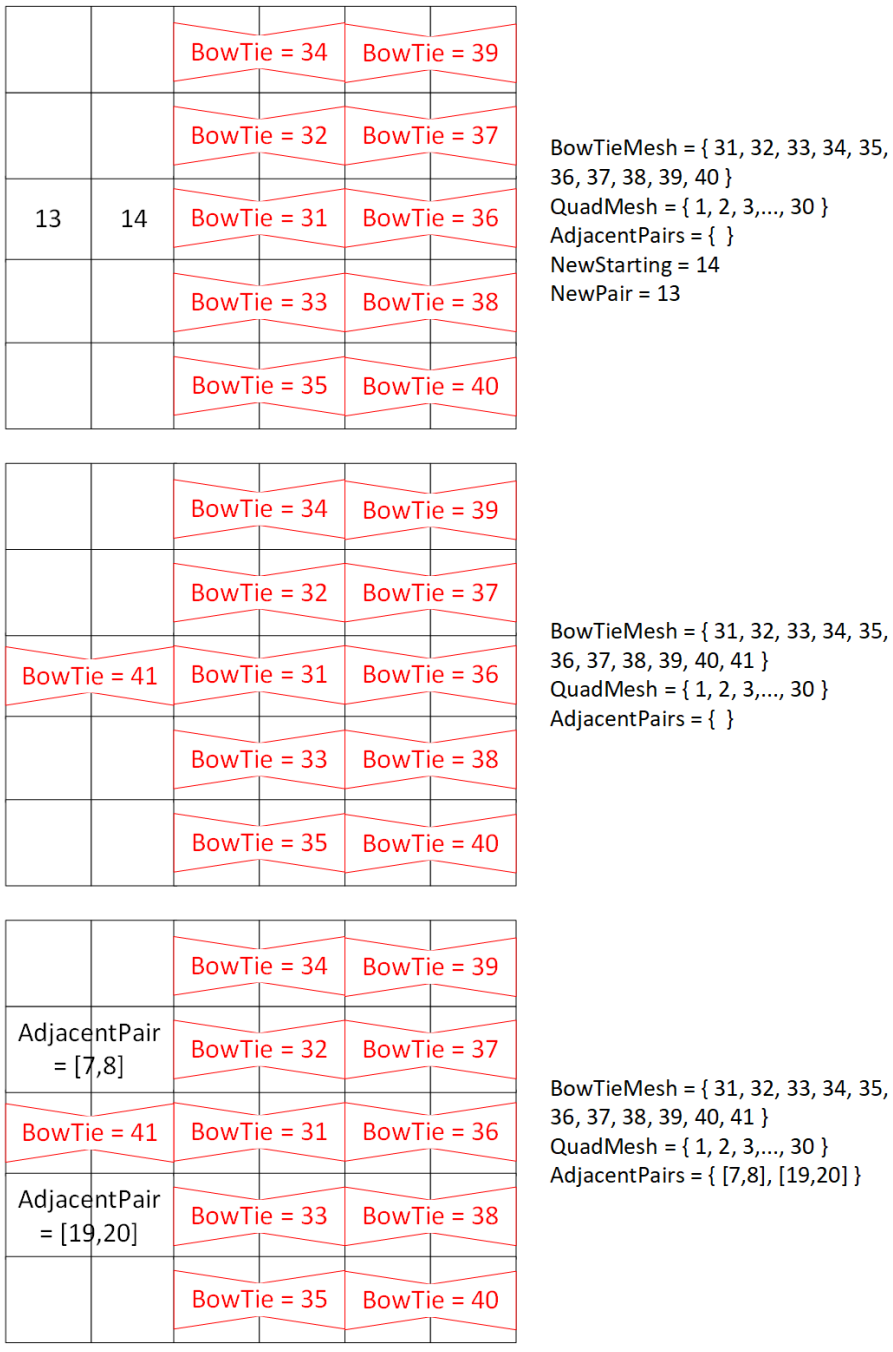


Figure 1: Execution steps of the FINDBOWTIES algorithm (cont.)

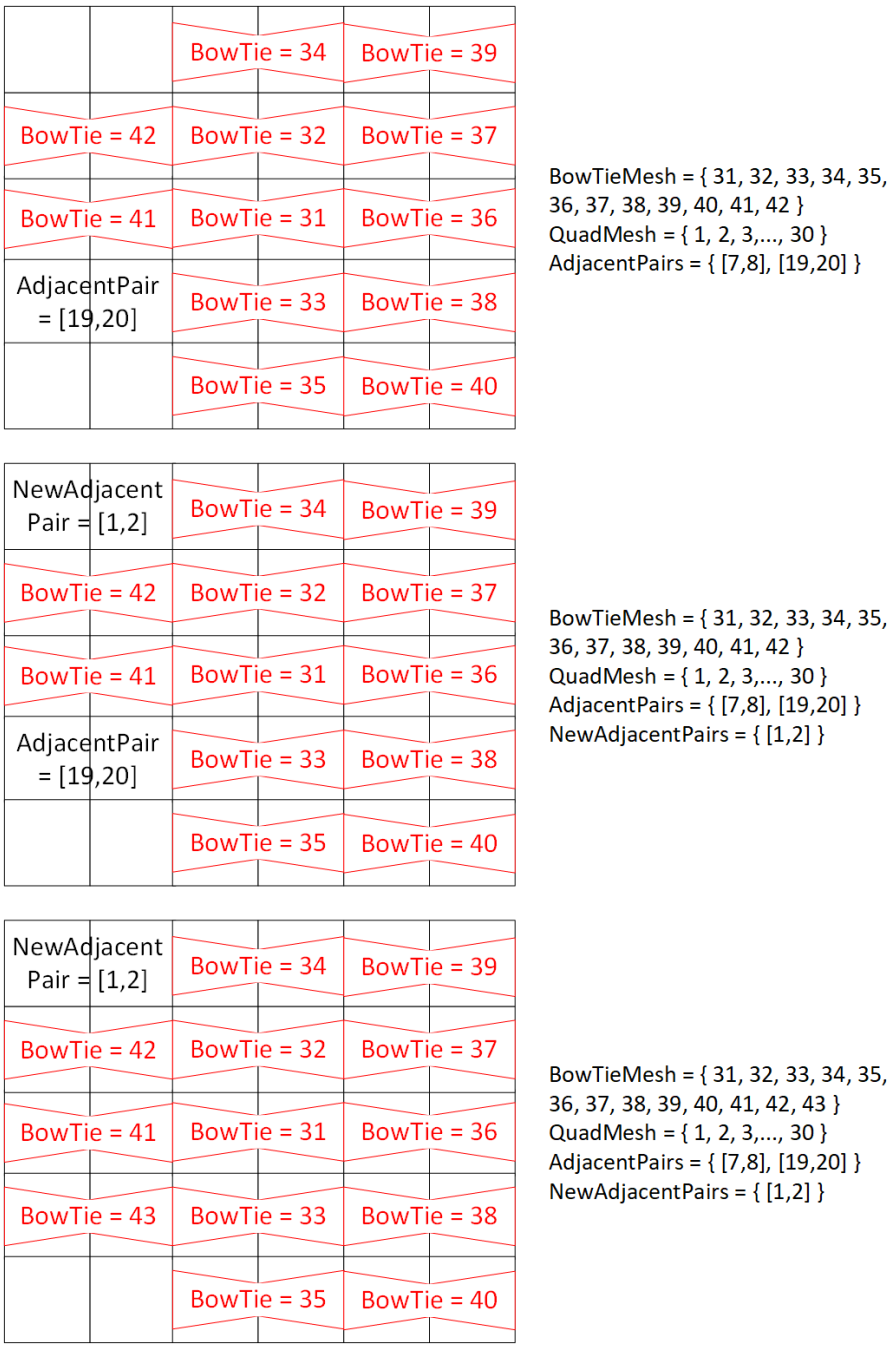
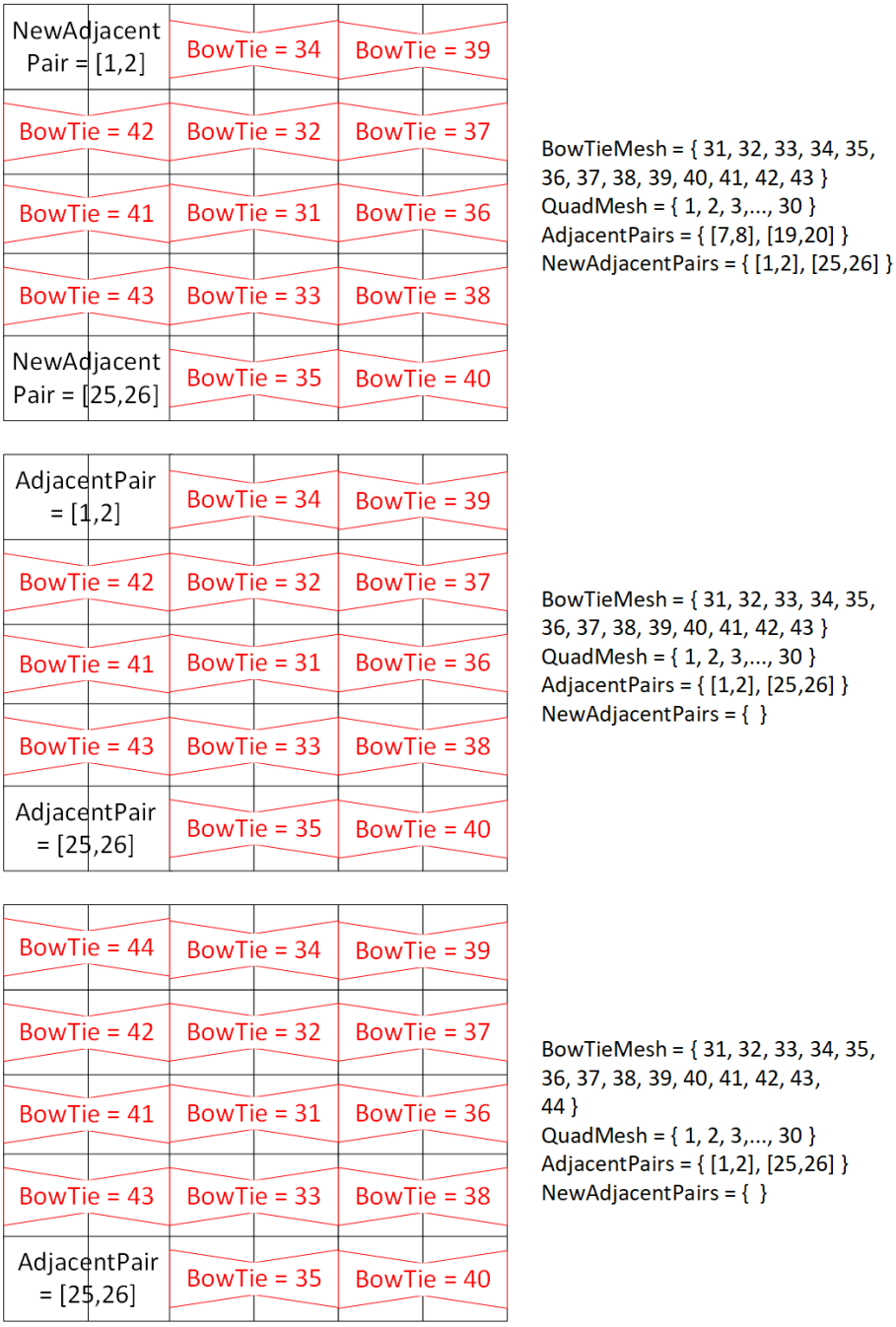
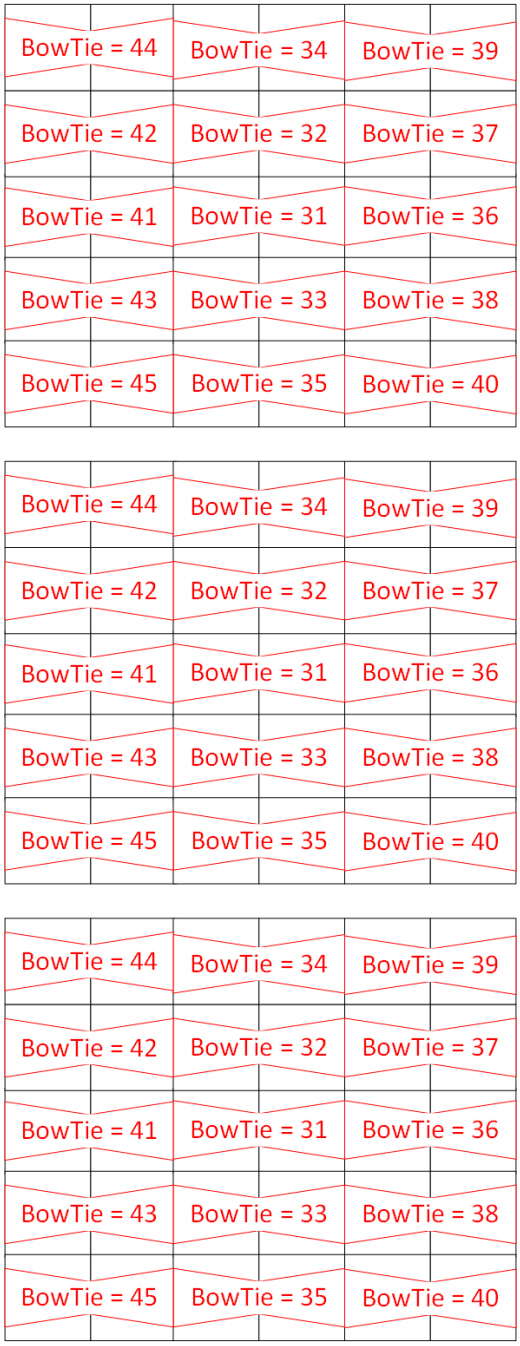


Figure 1: Execution steps of the FINDBOWTIES algorithm (cont.)



(k)

Figure 1: Execution steps of the FINDBOWTIES algorithm (cont.)



(l)

Figure 1: Execution steps of the FINDBOWTIES algorithm (cont.)

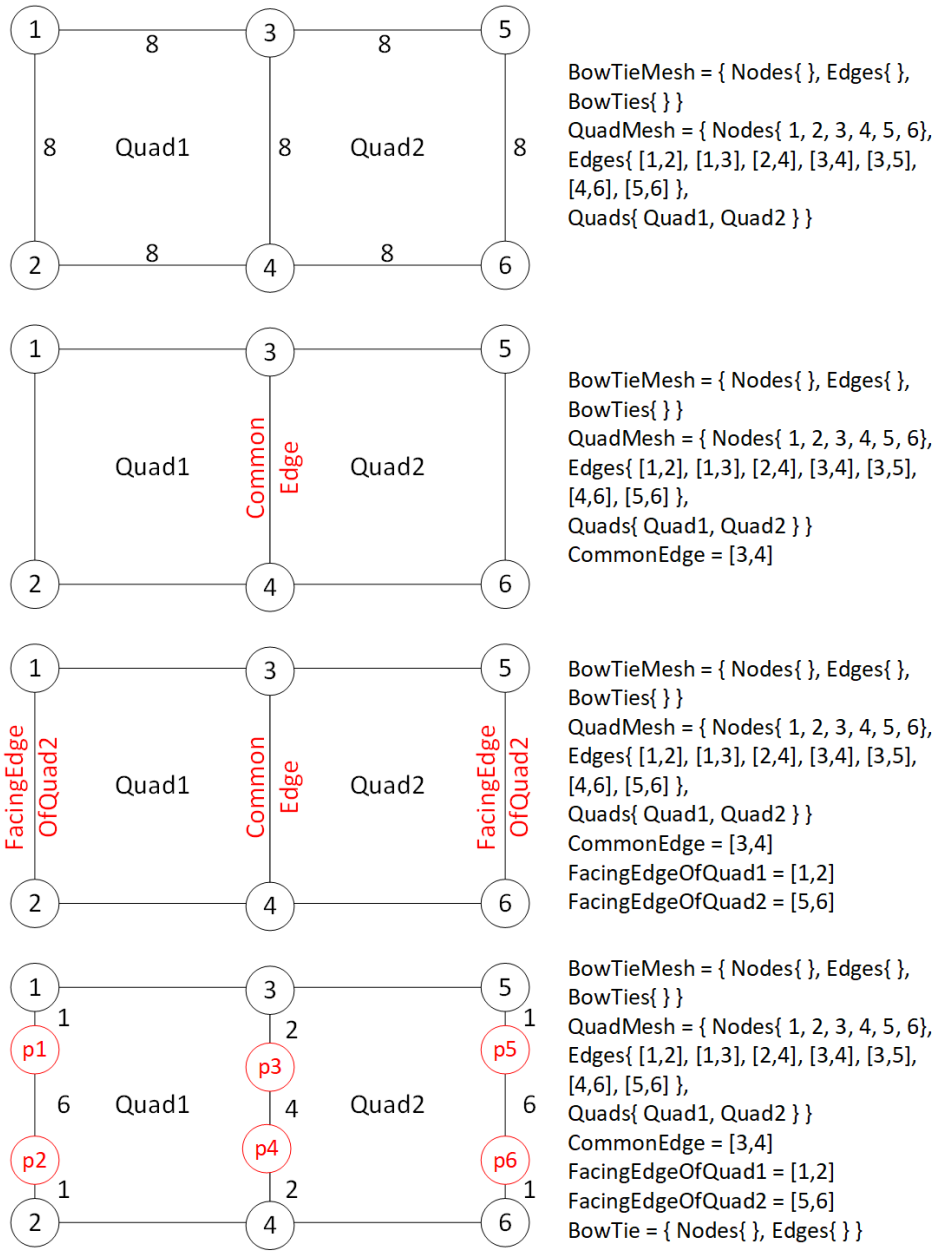
FINDBOWTIES algorithm employs INSERTBOWTIE algorithm to create and insert bow tie. Bow tie creation from a quad pair is shown in Algorithm 3. This algorithm calculates coordinates of points creating the bow tie and orders these points to create a related bow tie. Then, inserts the points and edges consisting of these points into bow tie mesh. Execution of this algorithm is shown in Figure 2.

Algorithm 3 INSERTBOWTIE(BowTieMesh, QuadMesh, Quad1, Quad2)

Input: Bow tie mesh which bow tie will be inserted, quad mesh and quads which will create bow tie together

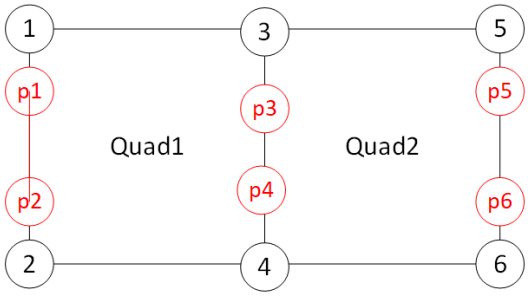
```
1:  if !Quad1.IsMatched and !Quad2.IsMatched then
2:    CommonEdge = common edge between Quad1 and Quad2
3:    if CommonEdge exists then
4:      FacingEdgeOfQuad1 = Quad1's facing edge to CommonEdge
5:      FacingEdgeOfQuad2 = Quad2's facing edge to CommonEdge
6:      if FacingEdgeOfQuad1 exists and FacingEdgeOfQuad2 exists then
7:        p1 = FacingEdgeOfQuad1.First + (lengt of FacingEdgeOfQuad1 ) * 1/8
8:        p2 = FacingEdgeOfQuad1.Second + (lengt of FacingEdgeOfQuad1 ) * 1/8

9:        p3 = CommonEdge.First + (length of CommonEdge) * 3 / 8
10:       p4 = CommonEdge.Second + (length of CommonEdge) * 3 / 8
11:       p5 = FacingEdgeOfQuad2.First + (length of FacingEdgeOfQuad2) * 1/8
12:       p6 = FacingEdgeOfQuad2.Second + (length of FacingEdgeOfQuad2) * 1/8

13:       Add points p1, p2, p3, p4, p5, p6 to BowTieMesh if they are not added
          before
14:       BowTie = empty set of ordered points creating bow tie
15:       Add point p1 to BowTie
16:       Add point p2 to BowTie
17:       if p3 is closer than p4 to p2 then
18:         Add point p3 to BowTie
19:         if p5 is closer than p6 to p3 then
20:           Add point p5 to BowTie
21:           Add point p6 to BowTie
22:         else
23:           Add point p6 to BowTie
24:           Add point p5 to BowTie
25:         end if
26:         Add point p4 to BowTie
27:       else
28:         Add point p4 to BowTie
29:         if p5 is closer than p6 to p3 then
30:           Add point p5 to BowTie
31:           Add point p6 to BowTie
32:         else
33:           Add point p6 to BowTie
34:           Add point p5 to BowTie
35:         end if
36:         Add point p3 to BowTie
```

```
37:          end if
38:          if Quad1's normal vector and BowTie's normal vector are in the opposite
             directions then
39:             Reverse points of BowTie
40:          end if
41:          Add BowTie to BowTieMesh
42:          Quad1.IsMatched = true
43:          Quad1.MatchedQuad = Quad2
44:          Quad2.IsMatched = true
45:          Quad2.MatchedQuad = Quad1
46:       end if
47:    end if
48: end if
```

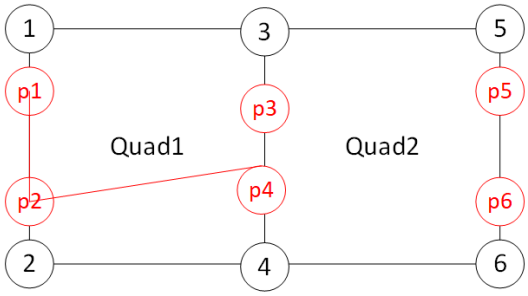


(a)

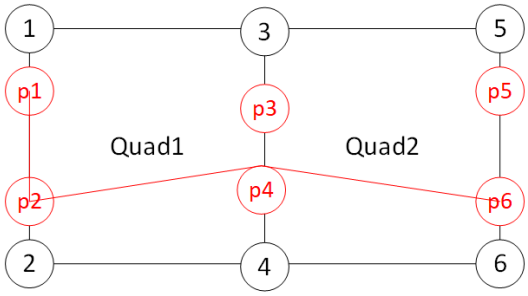
Figure 2: Execution steps of the INSERTBOWTIE algorithm



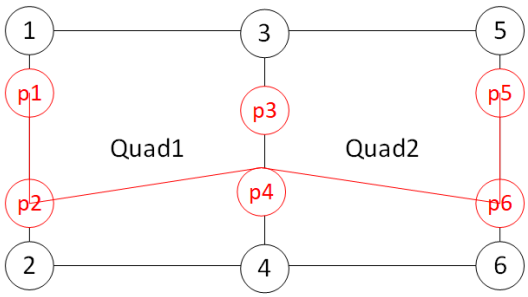
BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6}, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] },
BowTie = { Nodes{ p1, p2}, Edges{ [p1, p2] } }



BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6}, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] },
BowTie = { Nodes{ p1, p2, p4}, Edges{ [p1,p2], [p2,p4] } }



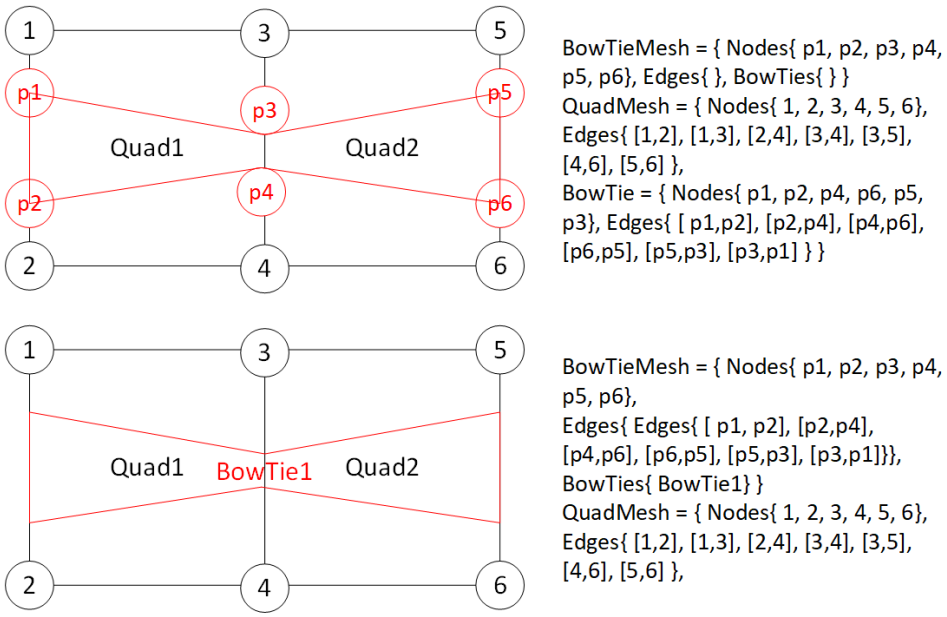
BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6}, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] },
BowTie = { Nodes{ p1, p2, p4, p6}, Edges{ [p1,p2], [p2,p4], [p4,p6] } }



BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6}, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6}, Edges{ [1,2], [1,3], [2,4], [3,4], [3,5], [4,6], [5,6] },
BowTie = { Nodes{ p1, p2, p4, p6, p5}, Edges{ [p1,p2], [p2,p4], [p4,p6], [p6,p5] } }

(b)

Figure 2: Execution steps of the INSERTBOWTIE algorithm (cont.)



(c)

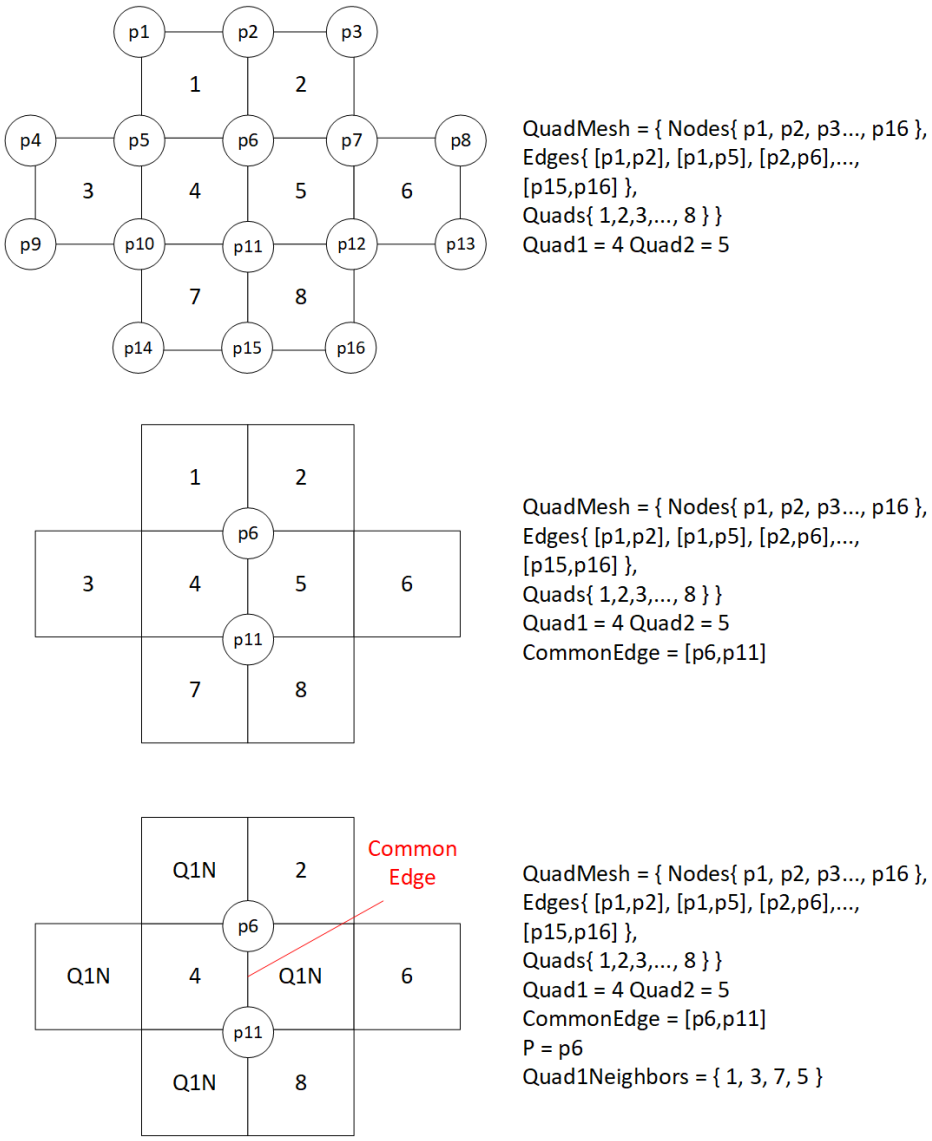
Figure 2: Execution steps of the INSERTBOWTIE algorithm (cont.)

Additionally, FINDBOWTIES algorithm uses GETADJACENTPAIRS algorithm for finding adjacent quad pairs to quad pairs created bow ties. Finding procedure is shown in Algorithm 4. This algorithm checks neighbor quads of the given quad pair and couples proper ones which can create new bow ties. Execution of this algorithm is shown in Figure 3.

Algorithm 4 GETADJACENTPAIRS(QuadMesh, Quad1, Quad2)

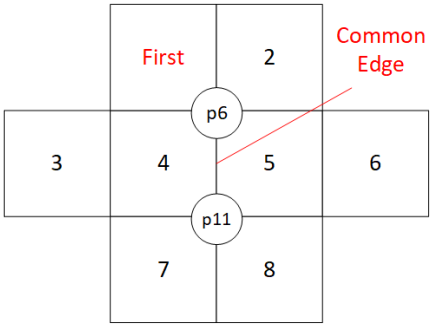
Input: Quad mesh and quads
Output: Adjacent quad pairs of input quads

```
Pairs = empty set of pairs of quads
CommonEdge = common edge between Quad1 and Quad2
if CommonEdge exists then
  for each point P ∈ CommonEdge do
    First = null
    Second = null
    Quad1Neighbors = quads having a common edge with Quad1
    for each neighbor quad NQ ∈ Quad1Neighbors do
      if NQ ≠ Quad2 and P is a point of NQ then
        First = NQ
        break
      end if
    end for
    Quad2Neighbors = quads having a common edge with Quad2
    for each neighbor quad NQ ∈ Quad2Neighbors do
      if NQ ≠ Quad1 and P is a point of NQ then
        Second = NQ
        break
      end if
    end for
    if First ≠ null and !First.IsMatched and Second ≠ null and !Second.IsMatched
    then
      Add [First, Second] to Pairs
    end if
  end for
end if
return Pairs
```

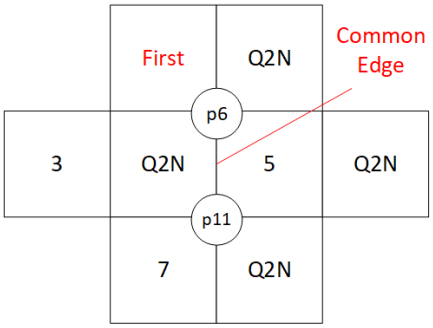


(a)

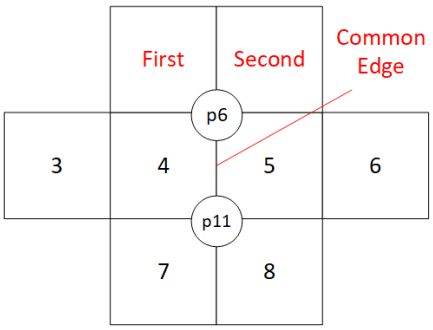
Figure 3: Execution steps of the GETADJACENTPAIRS algorithm



QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
P = p6
Quad1Neighbors = { 1, 3, 7, 5 }
First = 1



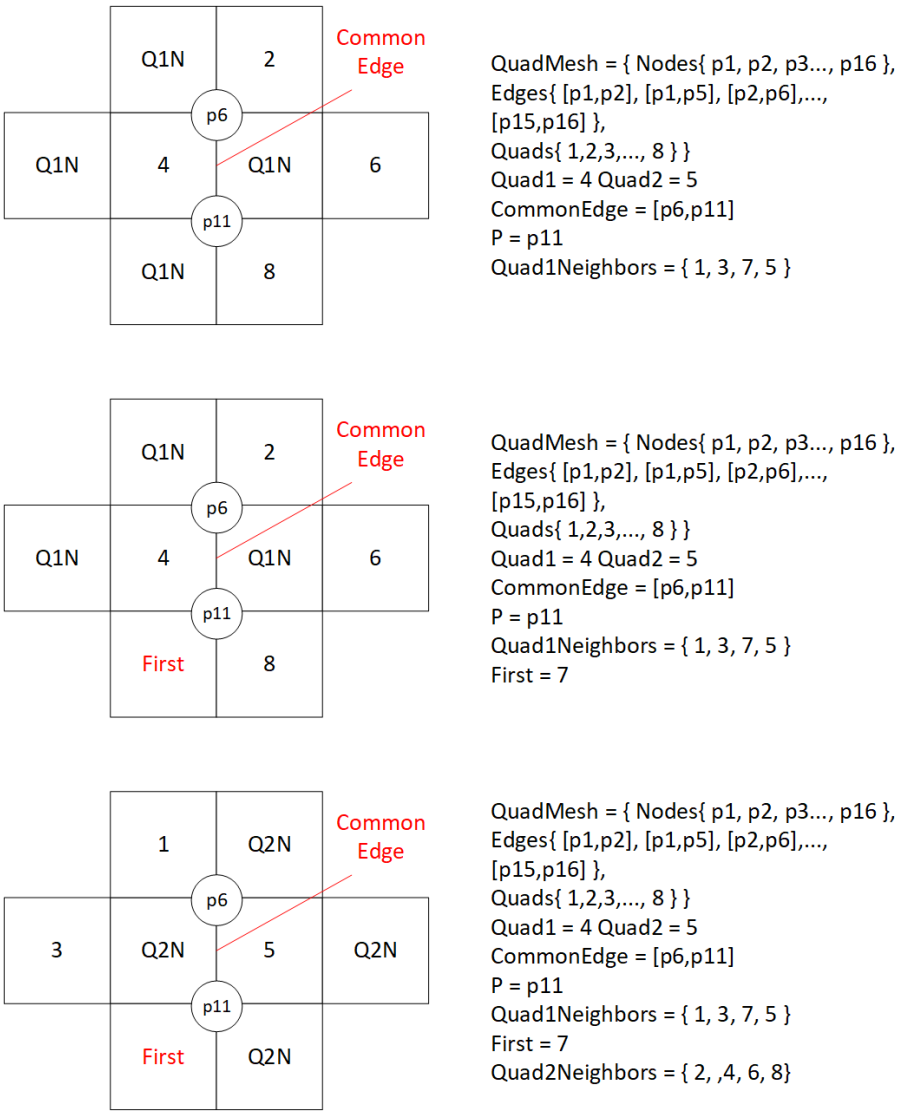
QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
P = p6
Quad1Neighbors = { 1, 3, 7, 5 }
First = 1
Quad2Neighbors = { 2, 4, 6, 8 }



QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
Quad1 = 4 Quad2 = 5
CommonEdge = [p6,p11]
Quad1Neighbors = { 1, 3, 7, 5 }
First = 1
Quad2Neighbors = { 2, 4, 6, 8 }
Second = 2
Pairs = { [1,2] }

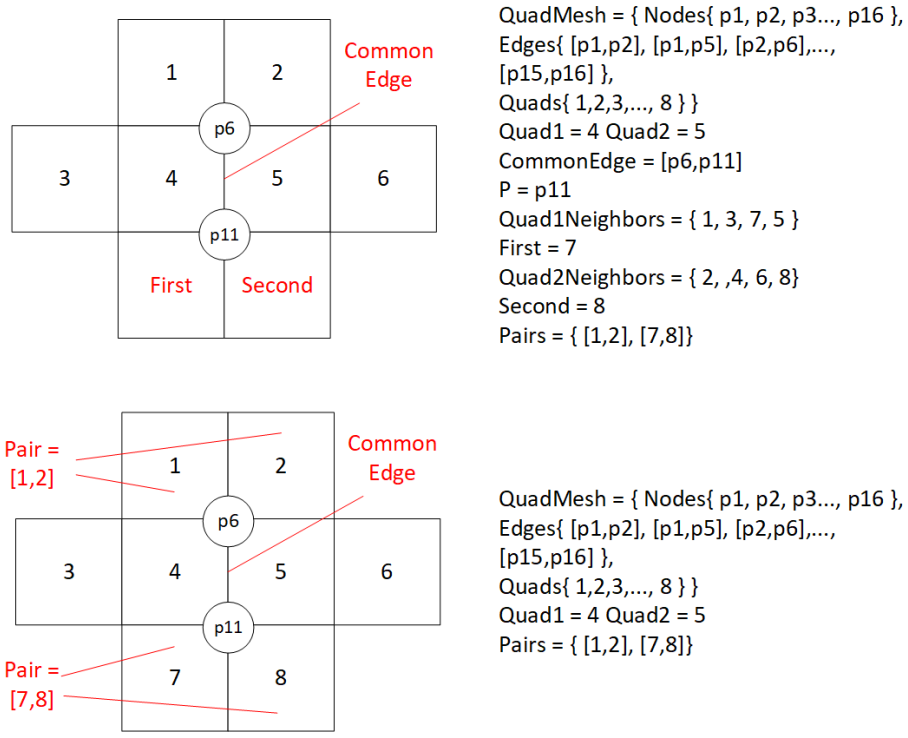
(b)

Figure 3: Execution steps of the GETADJACENTPAIRS algorithm (cont.)



(c)

Figure 3: Execution steps of the GETADJACENTPAIRS algorithm (cont.)



(d)

Figure 3: Execution steps of the GETADJACENTPAIRS algorithm (cont.)

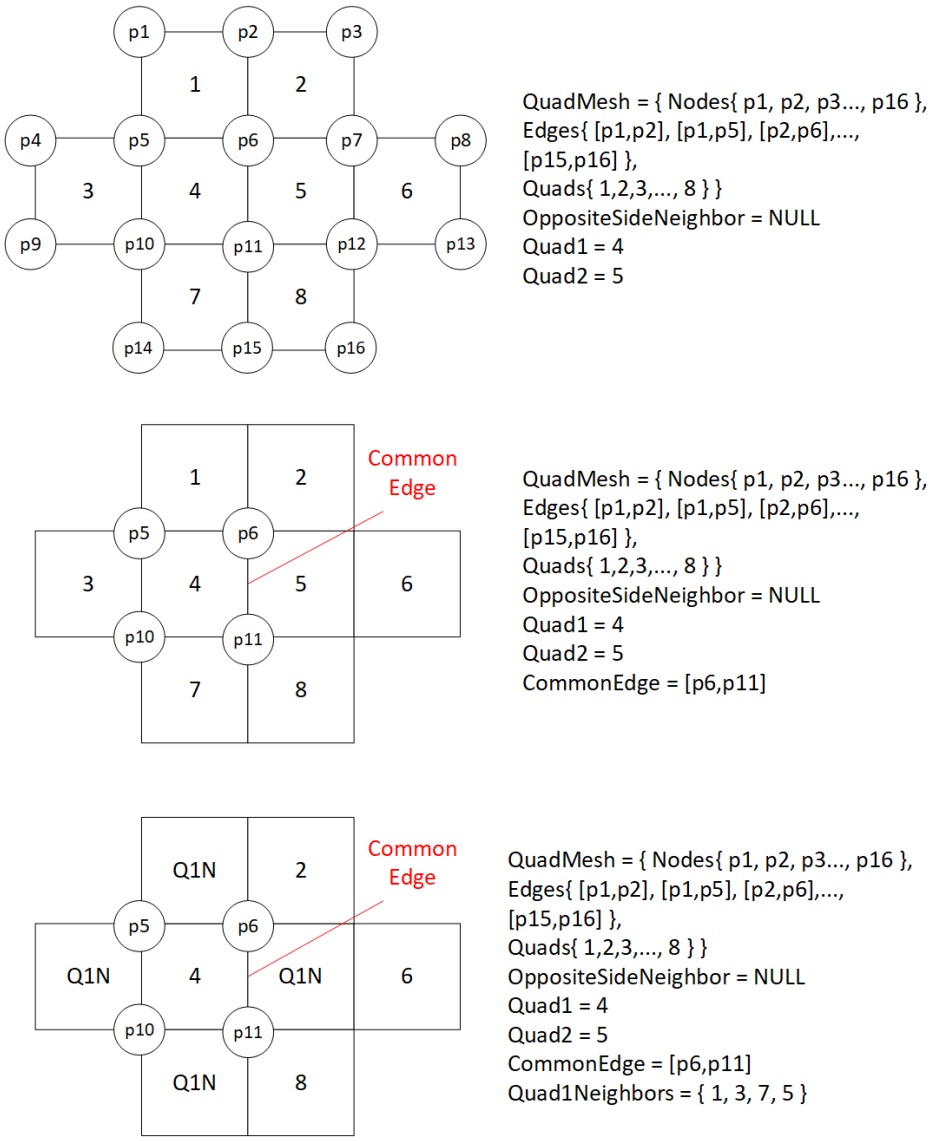
Finally, FINDBOWTIES algorithm uses GETOPPOSITESIDENEIGHBOR algorithm in order to continue finding bow ties and inserting them into bow tie mesh structure, recursively. The algorithm GETOPPOSITESIDENEIGHBOR finds and returns neighbor quad of given quad and its already founded neighbor quad. In other words, this algorithm returns the neighbor standing the opposite side of given quad according to its neighbor quad. How this algorithm works is shown in Algorithm 5. Execution of the GETOPPOSITESIDENEIGHBOR algorithm is shown in Figure 4.

Besides, the algorithm GETOPPOSITESIDENEIGHBOR is also employed by SEARCH algorithm. After finding bow ties process, SEARCH algorithm scans the quad mesh and detects the neighbor quads which does not participate in any bow tie of quads created a bow tie. When it finds such a quad, it gets this quad's neighbor quad which can create bow tie with this quad by using GETOPPOSITESIDENEIGHBOR algorithm.

Algorithm 5 GETOPPOSITESIDENEIGHBOR(QuadMesh, Quad1, Quad2)

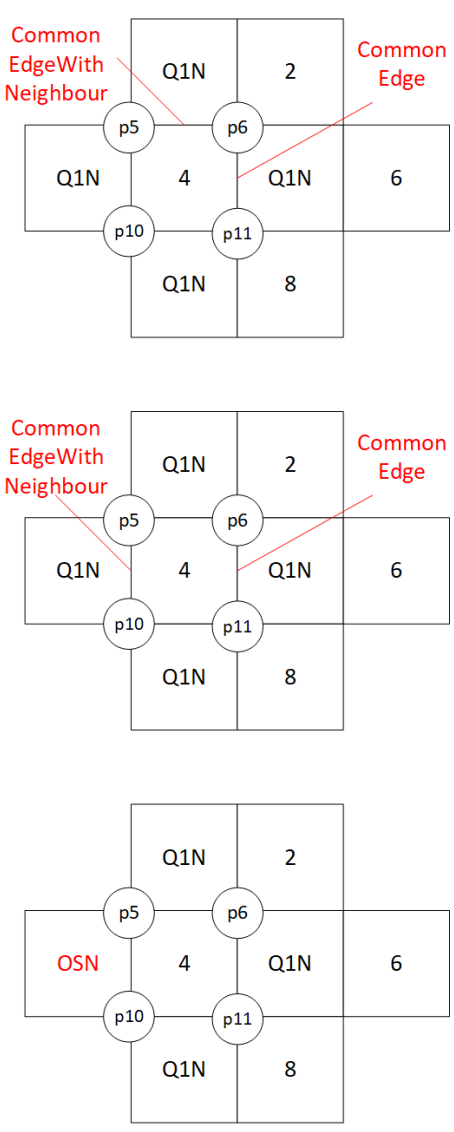
Input: Quad mesh, quad and neighbor quad of previous quad
Output: Other neighbor quad of input quad standing opposite side of quad according to input neighbor quad

```
1:  OppositeSideNeighbor = null
2:  CommonEdge = common edge between Quad1 and Quad2
3:  if CommonEdge exists then
4:    Quad1Neighbors = quads having a common edge with Quad1
5:    for each neighbor quad NQ ∈ Quad1Neighbors do
6:      if NQ ≠ Quad2 and !NQ.IsMatched then
7:        CommonEdgeWithNeighbor = common edge between Quad1 and NQ
8:        if CommonEdge and CommonEdgeWithNeighbor do not have any common point then
9:          OppositeSideNeighbor = NQ
10:         break
11:       end if
12:     end if
13:   end for
14: end if
15: return OppositeSideNeighbor
```



(a)

Figure 4: Execution steps of the GETOPPOSITESIDENEIGHBOR algorithm



QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
OppositeSideNeighbor = NULL
Quad1 = 4
Quad2 = 5
CommonEdge = [p6,p11]
Quad1Neighbors = { 1, 3, 7, 5 }
N = 1
CommonEdgeWithNeighbor = [p5, p6]
Common Point of Edges = p6

QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
OppositeSideNeighbor = NULL
Quad1 = 4
Quad2 = 5
CommonEdge = [p6,p11]
Quad1Neighbors = { 1, 3, 7, 5 }
N = 2
CommonEdgeWithNeighbor = [p5, p10]
Common Point of Edges = NULL

QuadMesh = { Nodes{ p1, p2, p3..., p16 },
Edges{ [p1,p2], [p1,p5], [p2,p6],...,
[p15,p16] },
Quads{ 1,2,3,..., 8 } }
OppositeSideNeighbor = 3
Quad1 = 4
Quad2 = 5
CommonEdge = [p6,p11]
Quad1Neighbors = { 1, 3, 7, 5 }
N = 2
CommonEdgeWithNeighbour = [p5, p10]
Common Point of Edges = NULL

(b)

Figure 4: Execution steps of the GETOPPOSITESIDENEIGHBOR algorithm (cont.)

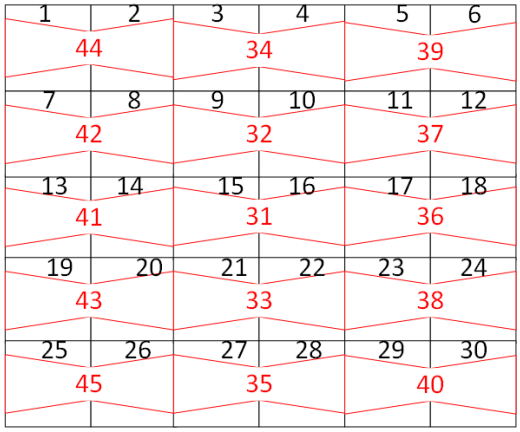
After the algorithm SEARCH completes the finding bow ties, then it starts a new finding operation to insert bow ties standing between founded bow ties, called internal bow ties. Internal bow ties cannot be inserted directly because founded bow ties generate them. If internal bow ties cannot be added into the bow tie mesh, gaps occur at the resulted bow tie mesh and these gaps cause disconnections. FINDINTERNALBOWTIES explained in Algorithm 6 is used for finding and inserting internal bow ties. This algorithm gets quad, checks bow ties around it for internal bow ties and inserts internal bow ties. After that, this algorithm jumps the neighbor quads of given quad and continues recursively. Execution of the FINDINTERNALBOWTIES algorithm is shown in Figure 5.

Algorithm 6 FINDINTERNALBOWTIES(BowTieMesh, QuadMesh, StartingQuad)

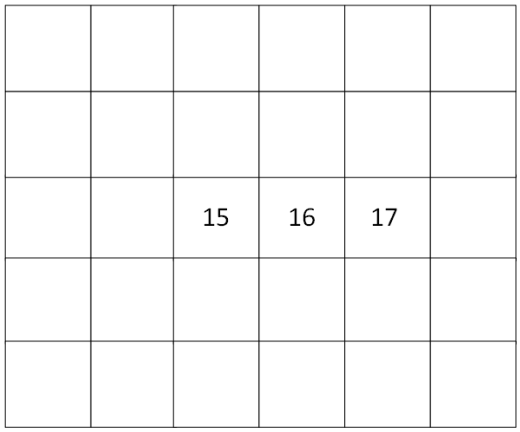
Input: Bow tie mesh for inserting founded internal bow ties, quad mesh and starting quad for searching internal bow ties

```
1: if StartingQuad.IsMatched then
2:   PairQuad = StartingQuad.MatchedQuad
3:   NeighborQuad = GETOPPOSITESIDENEIGHBOR(QuadMesh, StartingQuad,
     PairQuad)
4:   if NeighborQuad ≠ null and !NeighborQuad.IsInternalMatched and !Start-
     ingQuad.IsInternalMatched then
5:     PossibleInternalBowTies = empty set of [Quad1, Quad2, QuadPair] creating
       possible internal bow tie
6:     AdjacentInternalPairs = GETADJACENTINTERNALPAIRS(QuadMesh, Start-
       ingQuad, NeighbourQuad)
7:     for each adjacent internal pair AIP ∈ AdjacentInternalPairs do
8:       Add [StartingQuad, NeighborQuad, AIP] to PossibleInternalBowTies
9:     end for
10:    while PossibleInternalBowTies ≠ empty do
11:      NewPossibleInternalBowTies = empty set of [quad, quad, pair of quads]
         creating possible internal bow tie
12:      for each possible internal bow tie PIB ∈ AdjacentInternalPairs do
13:        INSERTINTERNALBOWTIE(BowTieMesh, QuadMesh, PIB.Quad1,
           PIB.Quad2, PIB.QuadPair)
14:        if bow tie was inserted from PIB then
15:          NewAdjacentInternalPairs = GETADJACENTINTERNALPAIRS(QuadMesh,
             PIB.QuadPair.First, PIB.QuadPair.Second)
16:          for each adjacent internal pair AIP ∈ NewAdjacentInternalPairs
             do
17:            Add [PIB.QuadPair.First, PIB.QuadPair.Second, AIP] to New-
               PossibleInternalBowTies
18:          end for
19:        end if
20:      end forPossibleInternalBowTies = NewPossibleInternalBowTies
21:    end while
22:    if NeighborQuad.IsMatched then
23:      NewStartingQuad = NeighborQuad.MatchedQuad
24:      FINDINTERNALBOWTIES(BowTieMesh, QuadMesh, NewStartingQuad)
```

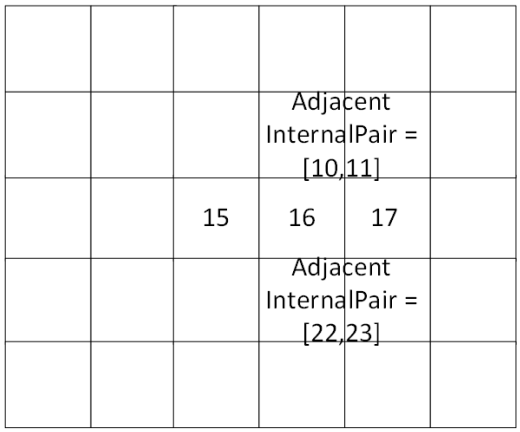
```
25:       end if
26:       NewStartingQuad = PairQuad
27:       FINDINTERNALBOWTIES(BowTieMesh, QuadMesh, NewStartingQuad)
28:     end if
29:  end if
```



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16



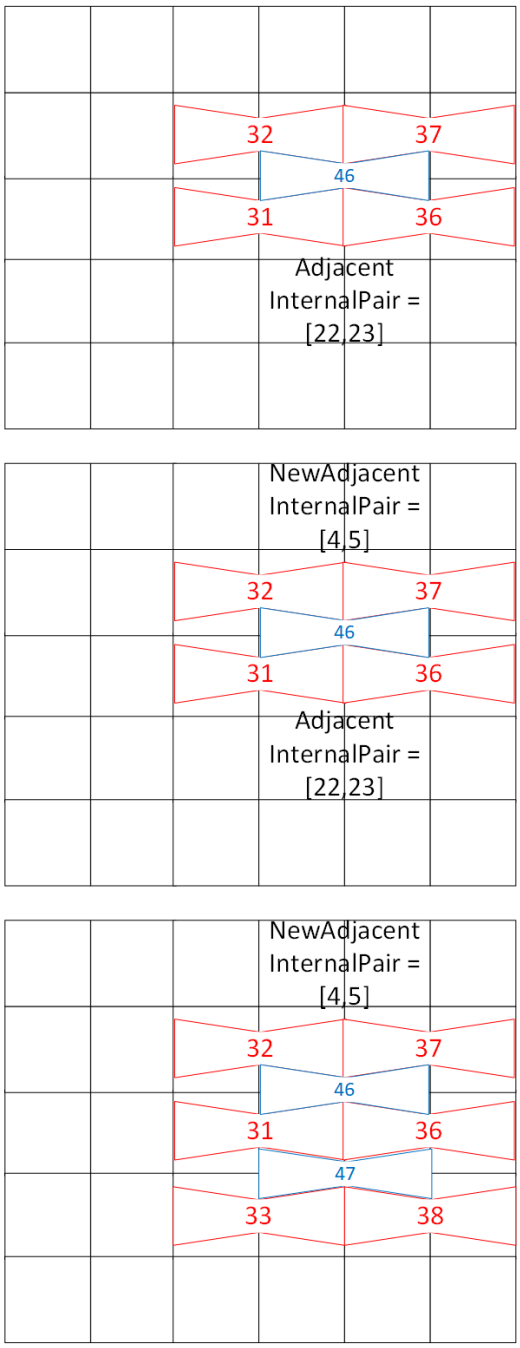
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { }



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {16, 17, [10,11]}, {16, 17, [22,23]} }
NewPossibleInternalBowTies = { }

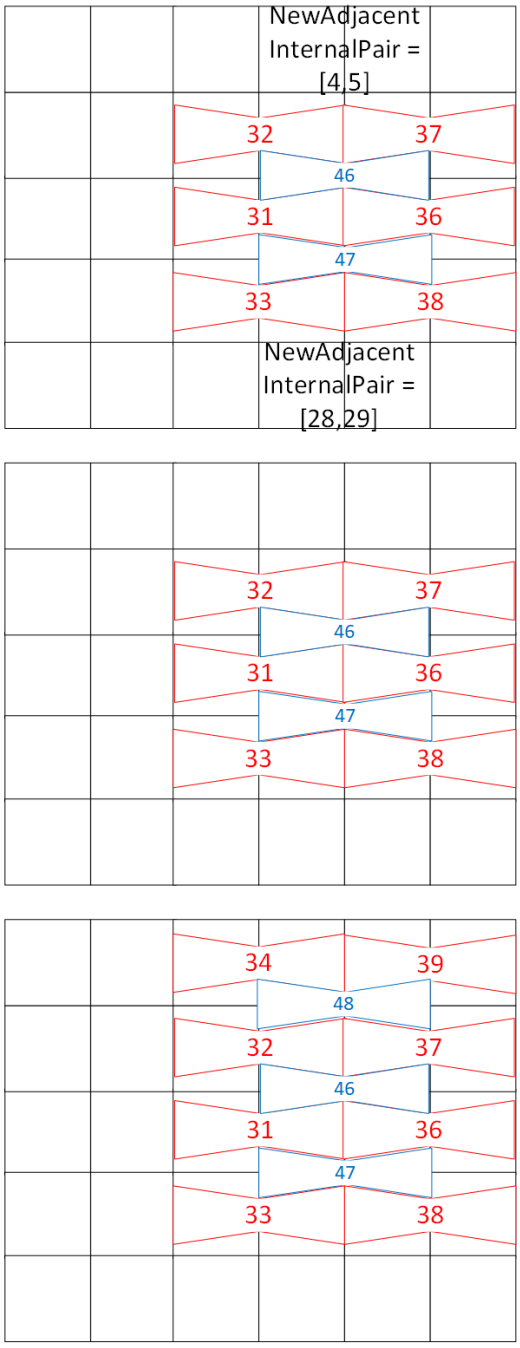
(a)

Figure 5: Execution steps of the FINDINTERNALBOWTIES algorithm



(b)

Figure 5: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)



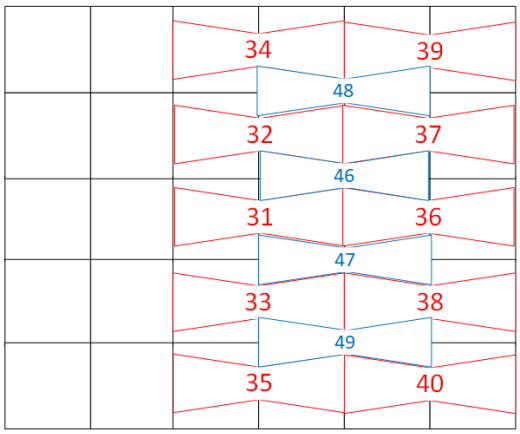
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {16, 17, [10,11]}, {16, 17, [22,23]} }
NewPossibleInternalBowTies = { {10, 11, [4,5]}, {22, 23, [28, 29]} }

BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {10, 11, [4,5]}, {22, 23, [28, 29]} }
NewPossibleInternalBowTies = { }

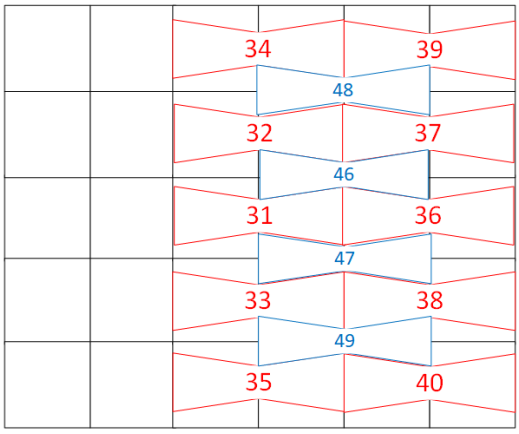
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {10, 11, [4,5]}, {22, 23, [28, 29]} }
NewPossibleInternalBowTies = { }

(c)

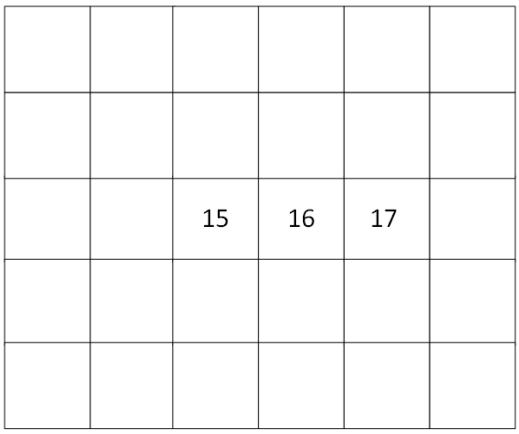
Figure 5: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)



BowTieMesh = { 31, 32, 33, 34, 35,
36, 37, 38, 39, 40, 41, 42, 43,
44, 45, 46, 47, 48, 49 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { {10, 11,
[4,5]}, {22, 23, [28, 29]} }
NewPossibleInternalBowTies = { }



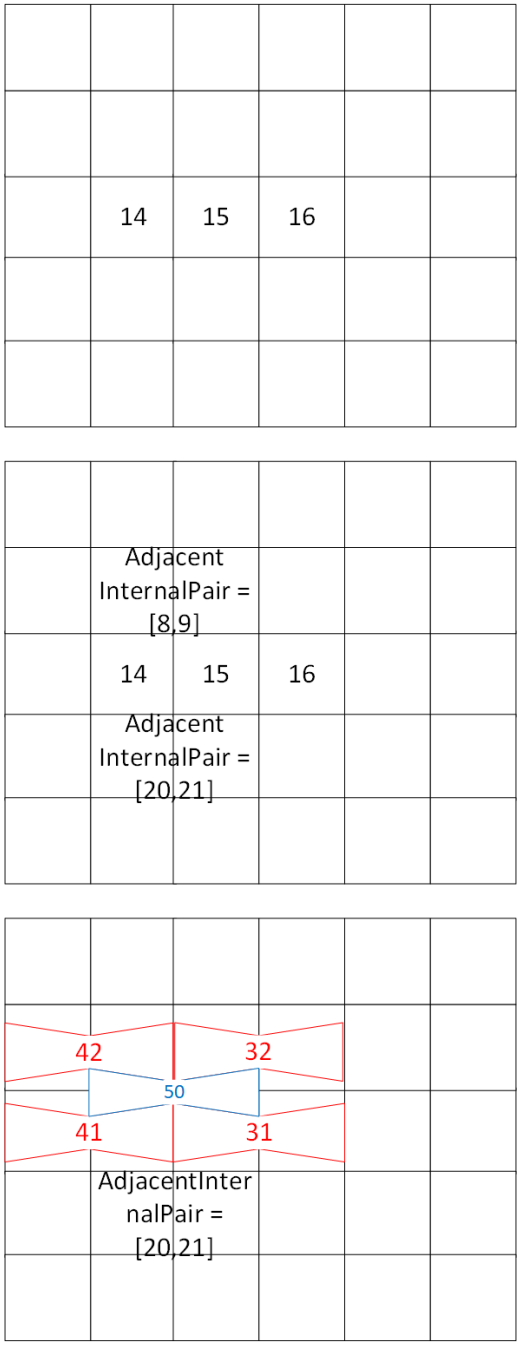
BowTieMesh = { 31, 32, 33, 34, 35,
36, 37, 38, 39, 40, 41, 42, 43,
44, 45, 46, 47, 48, 49 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { }



BowTieMesh = { 31, 32, 33, 34, 35,
36, 37, 38, 39, 40, 41, 42, 43,
44, 45, 46, 47, 48, 49 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 16
Pair = 15
Neighbor = 17
PossibleInternalBowTies = { }
NewStarting = Pair

(d)

Figure 5: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)



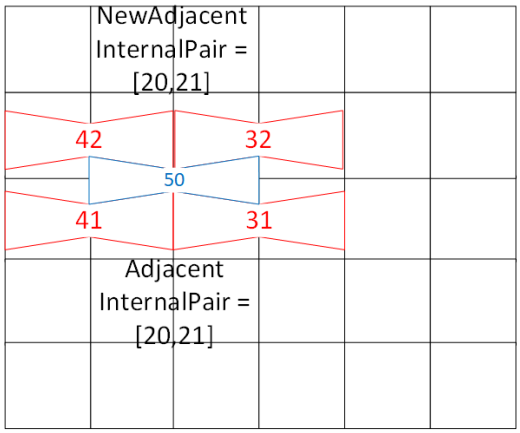
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { }

BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {15, 14, [8,9]}, {15, 14, [20,21]} }
NewPossibleInternalBowTies = { }

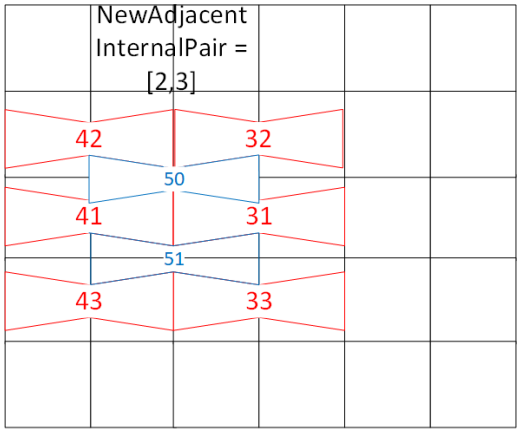
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {15, 14, [8,9]}, {15, 14, [20,21]} }
NewPossibleInternalBowTies = { }

(e)

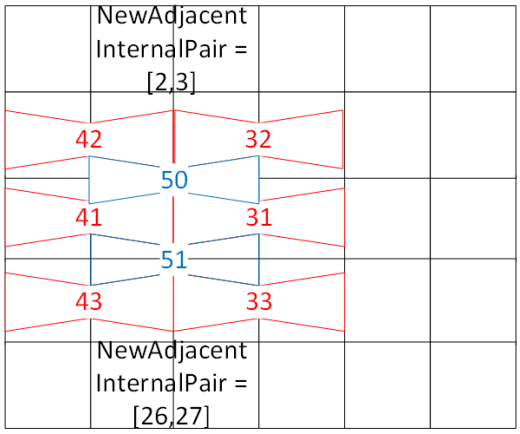
Figure 5: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {15, 14, [8,9]}, {15, 14, [20,21]} }
NewPossibleInternalBowTies = { {8, 9, [20,21]} }



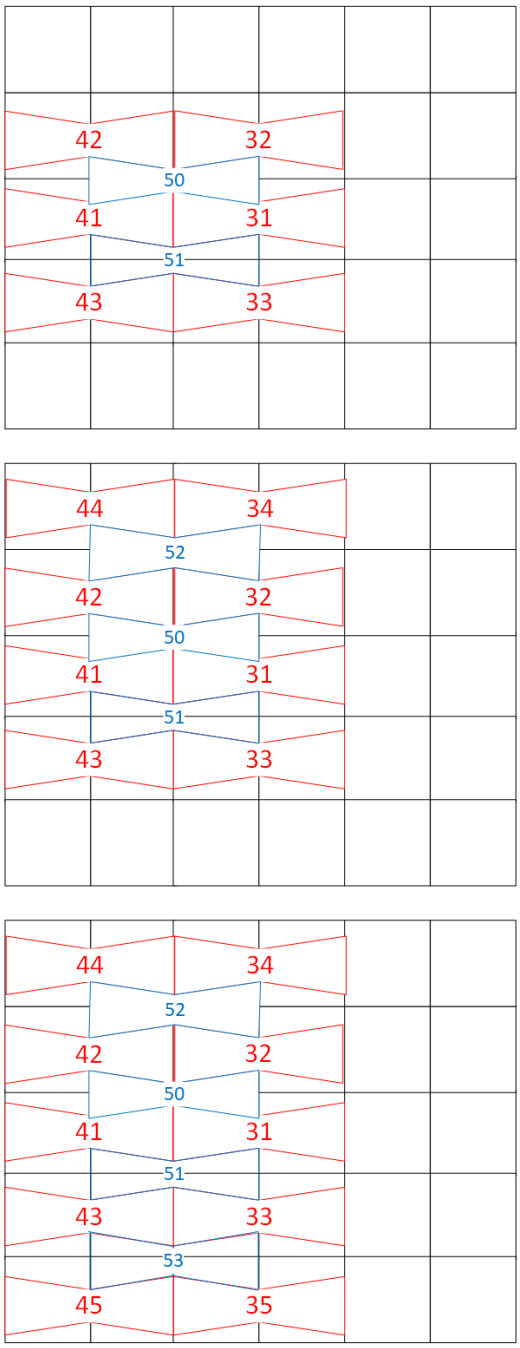
BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {15, 14, [8,9]}, {15, 14, [20,21]} }
NewPossibleInternalBowTies = { {8, 9, [2,3]} }



BowTieMesh = { 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { {15, 14, [8,9]}, {15, 14, [20,21]} }
NewPossibleInternalBowTies = { {8, 9, [2,3]}, {20, 21, [26, 27]} }

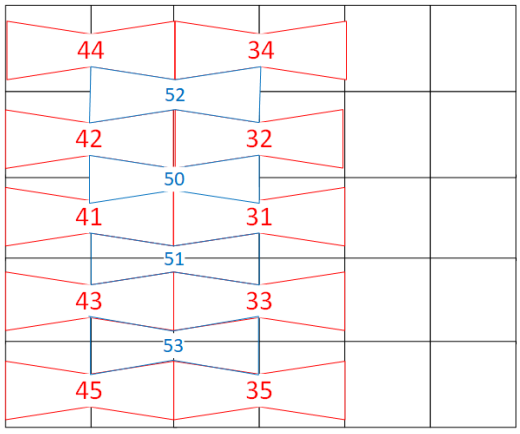
(f)

Figure 5: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)

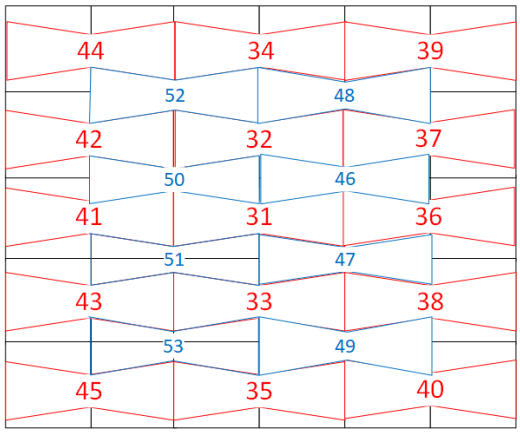


(g)

Figure 5: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)



BowTieMesh = { 31, 32, 33, 34, 35,
36, 37, 38, 39, 40, 41, 42, 43,
44, 45, 46, 47, 48, 49, 50, 51, 52, 53 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { }



BowTieMesh = { 31, 32, 33, 34, 35,
36, 37, 38, 39, 40, 41, 42, 43,
44, 45, 46, 47, 48, 49, 50, 51, 52, 53 }
QuadMesh = { 1, 2, 3,..., 30 }
Starting = 15
Pair = 16
Neighbor = 14
PossibleInternalBowTies = { }
NewStarting = NULL

(h)

Figure 5: Execution steps of the FINDINTERNALBOWTIES algorithm (cont.)

Similar to algorithm FINDBOWTIES described in Algorithm 2, FINDINTERNALBOWTIES algorithm is also used GETOPPOSITESIDENEIGHBOR algorithm described in Algorithm 5 in order to find opposite side neighbor of the given quad.

The algorithm FINDINTERNALBOWTIES needs procedure getting adjacent internal pairs of quads of the given quad pair in order to create new internal bow ties. This procedure is very close the Algorithm 4 used by the Algorithm 2. Detail of this procedure is given in Algorithm 7. This algorithm, GETADJACENTINTERNALPAIRS, takes a pair of quads, checks their neighbors which created bow tie before and returns the suitable combination of them to create an internal bow tie with the given pair of quads. GETADJACENTINTERNALPAIRS algorithm is almost the same with the Algorithm 4 except being pair criteria therefore the same Figure 3 is valid for its execution.

Algorithm 7 GETADJACENTINTERNALPAIRS(QuadMesh, Quad1, Quad2)

Input: Quad mesh and quads
Output: Adjacent internal quad pairs of input quads

```
 1: Pairs = empty set of pairs of quads
 2: CommonEdge = common edge between Quad1 and Quad2
 3: if CommonEdge exists then
 4:    for each point P ∈ CommonEdge do
 5:       First = null
 6:       Second = null
 7:       Quad1Neighbors = quads having a common edge with Quad1
 8:       for each neighbor quad NQ ∈ Quad1Neighbors do
 9:          if NQ ≠ Quad2 and P is a point of NQ then
10:             First = NQ
11:             break
12:          end if
13:       end for
14:       Quad2Neighbors = quads having a common edge with Quad2
15:       for each neighbor quad NQ ∈ Quad2Neighbors do
16:          if NQ ≠ Quad1 and P is a point of NQ then
17:             Second = NQ
18:             break
19:          end if
20:       end for
21:       if First ≠ null and Second ≠ null and First ≠ Second and !First.IsInternalMatched
          and !Second.IsInternalMatched and First.IsMatched and Second.IsMatched
          and First.MatchedQuad ≠ Second and Second.MatchedQuad ≠ First then
22:          Add [First, Second] to Pairs
23:       end if
24:    end for
25: end if
26: return Pairs
```

FINDINTERNALBOWTIES algorithm needs a different method from the INSERTBOWTIE described in Algorithm 3 to create and insert internal bow

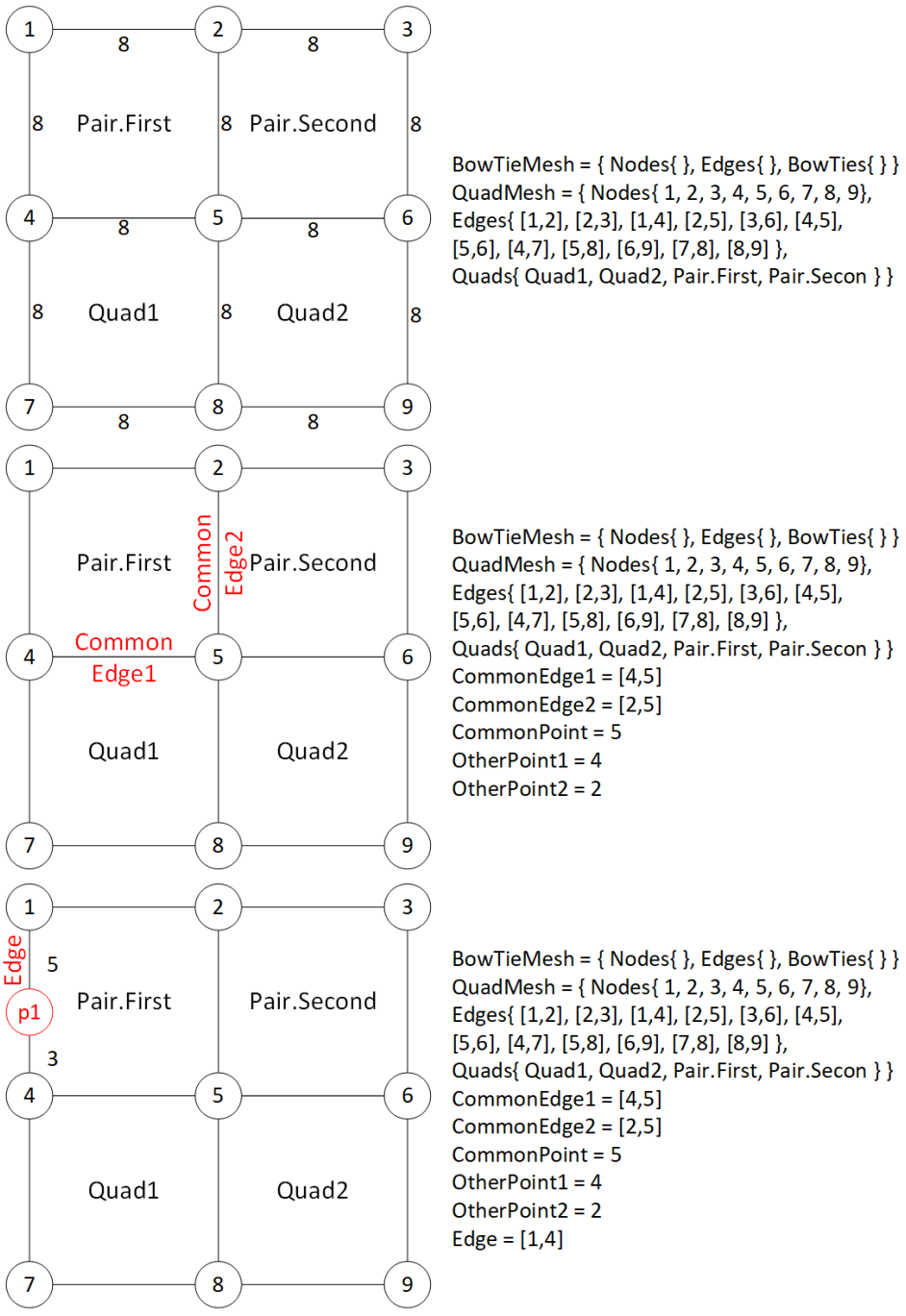
ties because bow ties consist of quads while internal bow ties consist of the other bow ties. This method is named INSERTINTERNALBOWTIE and described in Algorithm 8. INSERTINTERNALBOWTIE algorithm takes four quads which created bow tie before in doubles. This algorithm calculates coordinates of points creating the internal bow tie and orders these points to create related internal bow tie. Execution of this algorithm is shown in Figure 6.

Algorithm 8 INSERTINTERNALBOWTIE(BowTieMesh, QuadMesh, Quad1, Quad2, QuadPair)

Input: Bow tie mesh which internal bow tie will be inserted, quad mesh, quads and adjacent internal pair quads of quads which will create internal bow tie together

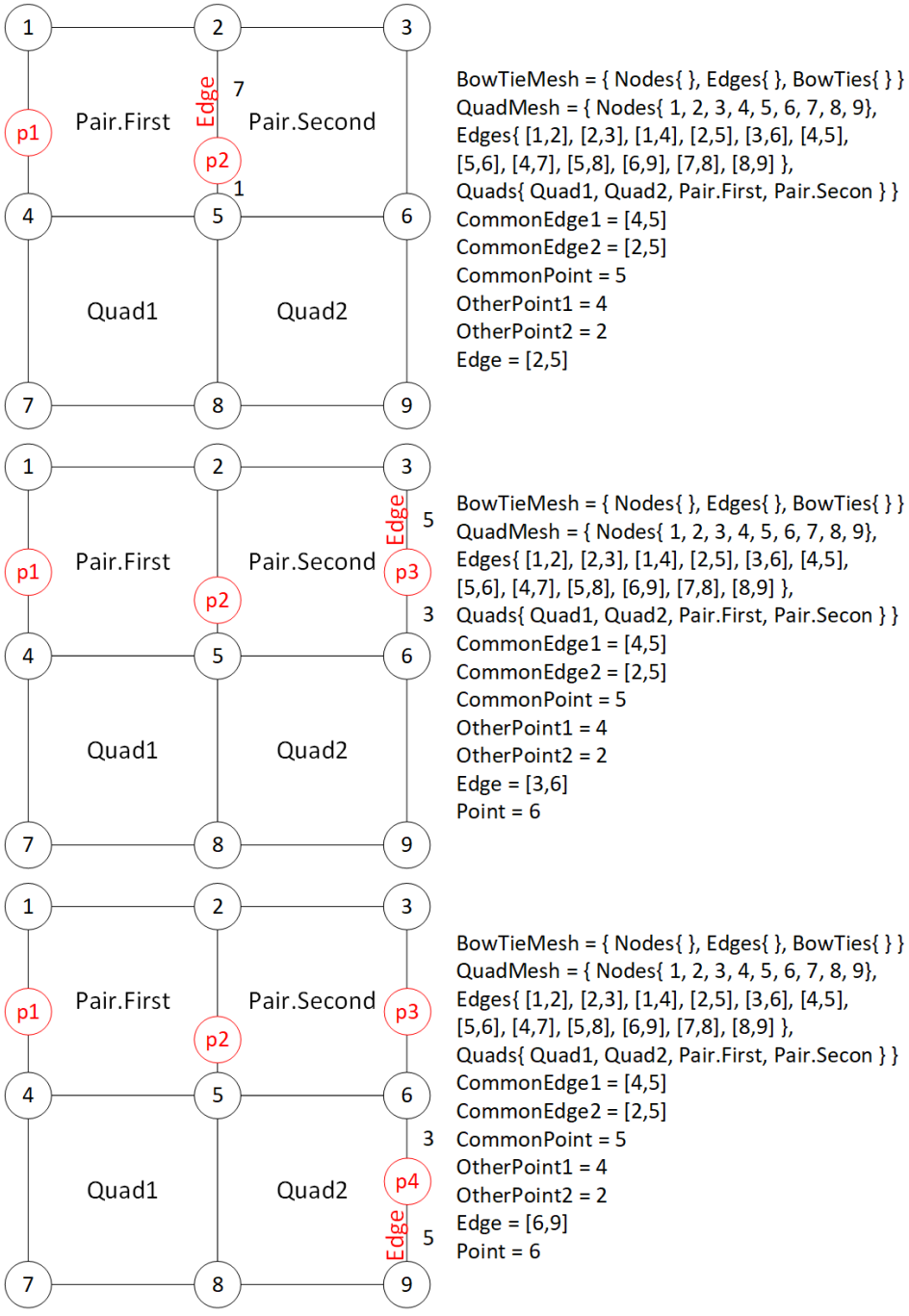
```
1: CommonEdge1 = common edge between Quad1 and QuadPair.First
2: CommonEdge2 = common edge between QuadPair.First and QuadPair.Second
3: if CommonEdge1 exists and CommonEdge2 exists then
4:    CommonPoint = common point of CommonEdge1, CommonEdge2
5:    if CommonPoint exists then
6:       OtherPoint1 = CommonEdge1's other point than CommonPoint
7:       OtherPoint2 = CommonEdge2's other point than CommonPoint
8:       Edge = QuadPair.Firsts edge containing OtherPoint1 and not containing
         CommonPoint
9:       p1 = OtherPoint1 + (length of Edge) * 3/8
10:      Edge = QuadPair.Firsts edge containing CommonPoint and not containing
         OtherPoint1
11:      p2 = CommonPoint + (length of Edge) * 1/8
12:      Edge = Pair.Seconds edge not containing CommonPoint and not containing
         OtherPoint2
13:      Point = Edges point creating an edge with CommonPoint
14:      p3 = Point + (length of Edge) * 1/8
15:      Edge = Quad2s edge containing Point and not containing CommonPoint
16:      p4 = Point + (length of Edge) * 1/8
17:      Edge = Quad2s edge containing CommonPoint and not containing Point
18:      p5 = CommonPoint + (length of Edge) * 3/8
19:      Edge = Quad1s edge containing OtherPoint1 and not containing Common-
         Point
20:      p6 = OtherPoint1 + (length of Edge) * 3/8
21:      Add points p1, p2, p3, p4, p5, p6 to BowTieMesh if they are not added before

22:      BowTie = empty set of ordered points creating bow tie
23:      Add ordered points p1, p2, p3, p4, p5, p6 to BowTie
24:      if Quad1's normal vector and BowTie's normal vector are in the opposite
         directions then
25:         Reverse points of BowTie
26:      end if
27:      Add BowTie to BowTieMesh
28:      Quad1.IsInternalMatched = true
29:      Quad2.IsInternalMatched = true
30:      QuadPair.First.IsInternalMatched = true
31:      QuadPair.Second.IsInternalMatched = true
32:   end if
```

33: **else**
34: INSERTHALFBOWTIE(BowTieMesh, QuadMesh, Quad1, Quad2, QuadPair)
35: **end if**



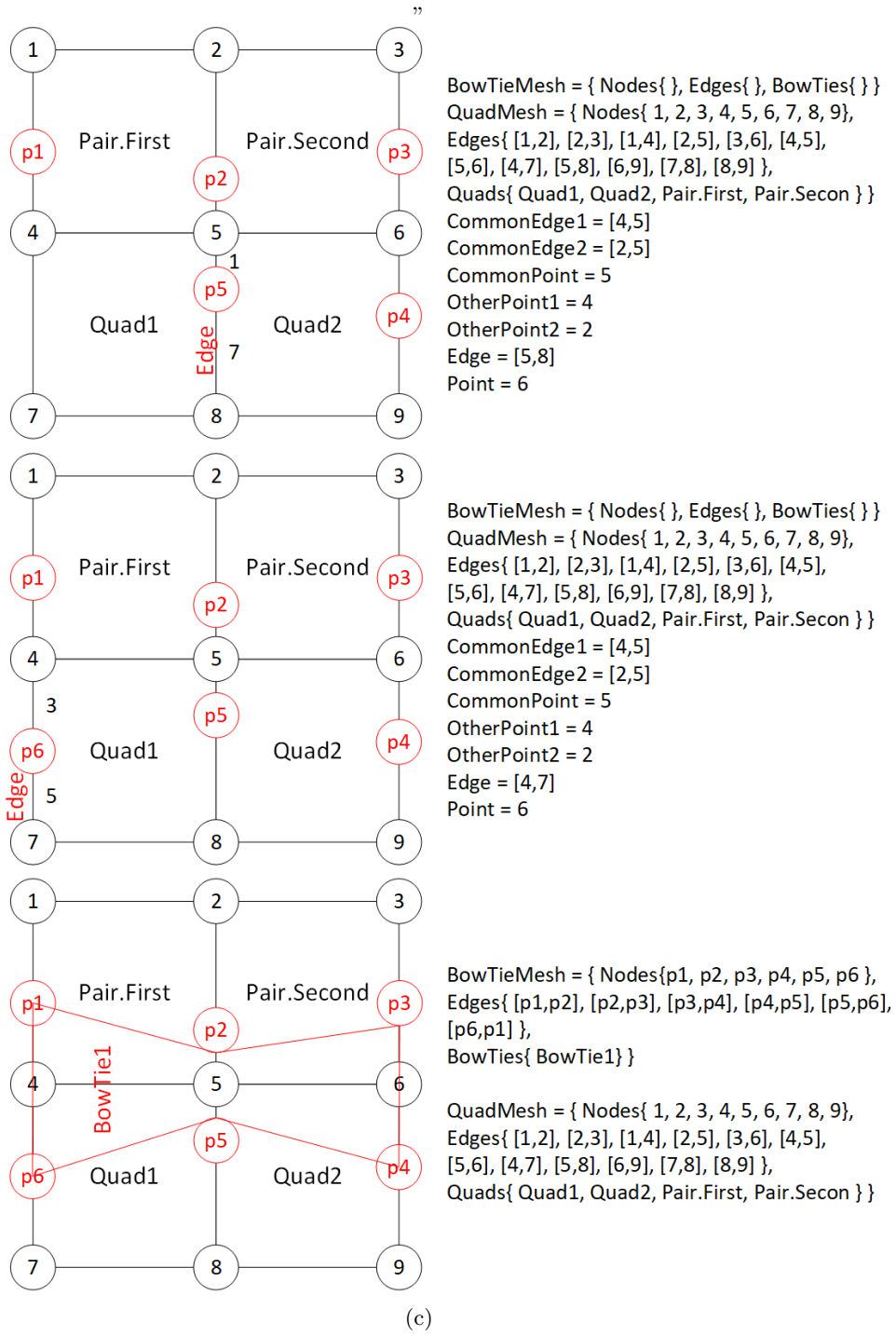
(a)

Figure 6: Execution steps of the INSERTINTERNALBOWTIE algorithm



(b)

Figure 6: Execution steps of the INSERTINTERNALBOWTIE algorithm (cont.)



(c)

Figure 6: Execution steps of the INSERTINTERNALBOWTIE algorithm (cont.)

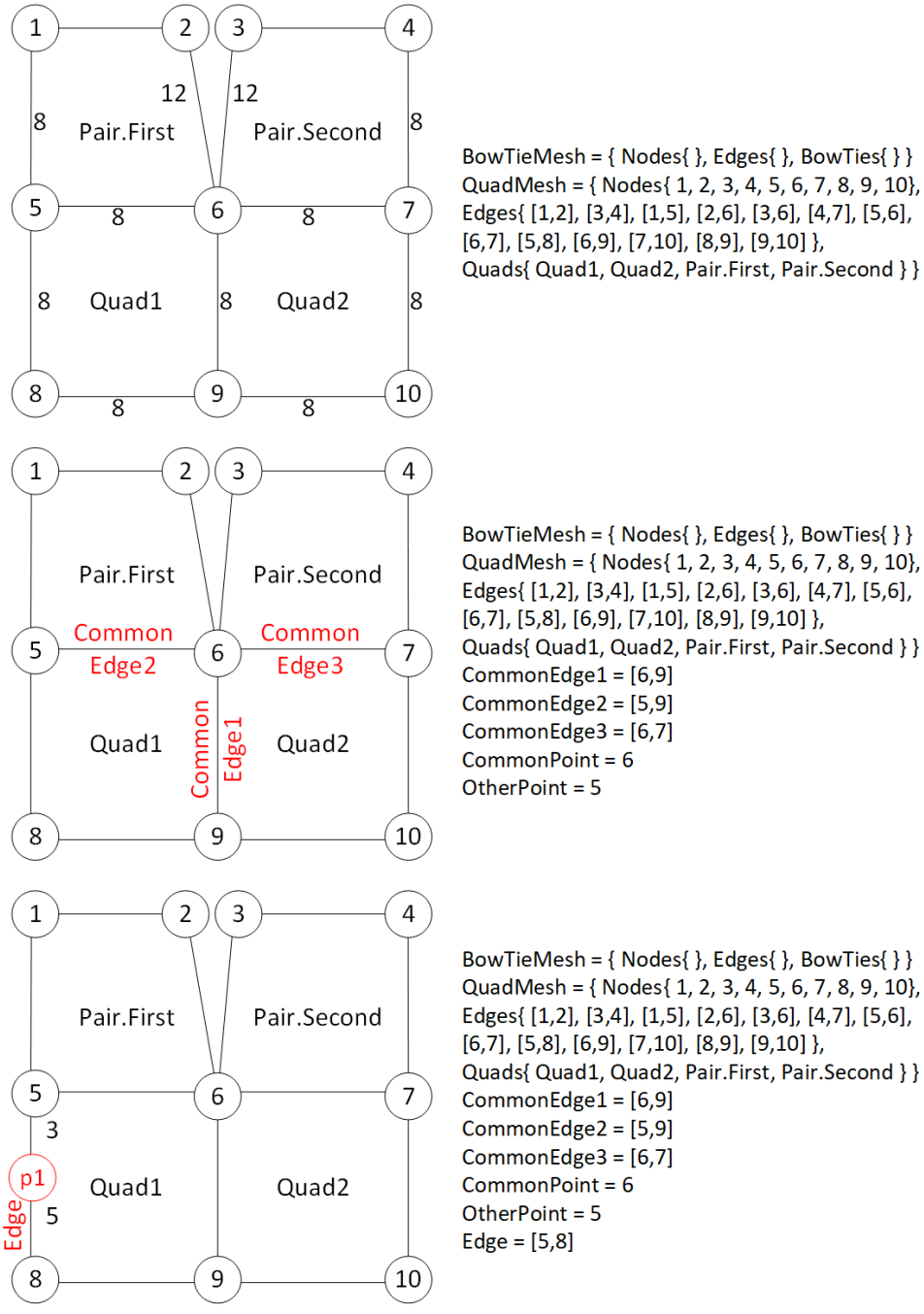
If the algorithm INSERTINTERNALBOWTIE cannot achieve to create complete bow tie due to lack of common edges, it applies to INSERTHALFBOWTIE described in Algorithm 9. This algorithm tries to create one or two half bow tie with the given quads and inserts what it founded in the bow tie mesh. Half bow ties are very important to keep resulted bow tie mesh together. Insertion steps of half bow ties are shown in Figure 7.

Algorithm 9 INSERTHALFBOWTIE(BowTieMesh, QuadMesh, Quad1, Quad2, QuadPair)

Input: Bow tie mesh which half bow tie will be inserted, quad mesh, quads and adjacent internal pair quads of quads which will create half bow tie together

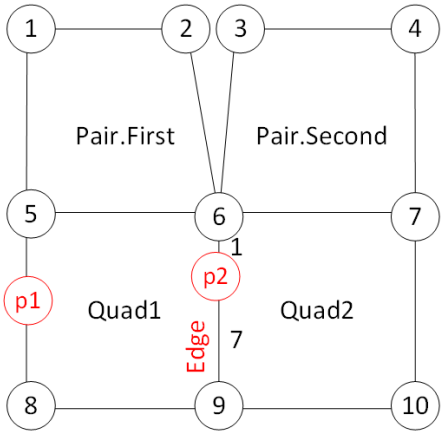
```
CommonEdge1 = common edge between Quad1 and Quad2
CommonEdge1 = common edge between Quad1 and QuadPair.First
CommonEdge2 = common edge between Quad2 and QuadPair.Second
if CommonEdge1 exists and CommonEdge2 exists then
    CommonPoint = common point of CommonEdge1, CommonEdge2
    if CommonPoint exists then
        OtherPoint = CommonEdge2's other point than CommonPoint
        Edge = Quad1s edge containing OtherPoint and not containing CommonPoint
        p1 = OtherPoint + (length of Edge) * 3/8
        Edge = Quad1's edge containing CommonPoint and not containing OtherPoint
        p2 = CommonPoint + (length of Edge) * 1/8
        Edge = Pair.Firsts edge containing CommonPoint and not containing OtherPoint
        p3 = CommonPoint + (length of Edge) * 1/8
        Edge = Pair.Firsts edge containing OtherPoint and not containin CommonPoint
        p4 = OtherPoint + (length of Edge) * 3/8
        Add points p1, p2, p3, p4 to BowTieMesh if they are not added before
        HalfBowTie = empty set of ordered points creating bow tie
        Add ordered points p1, p2, p3, p4 to HalfBowTie
        if Quad1's normal vector and HalfBowTie's normal vector are in the opposite directions then
            Reverse points of HalfBowTie
        end if
        Add HalfBowTie to BowTieMesh
    end if
end if
if CommonEdge1 exists and CommonEdge3 exists then
    CommonPoint = common point of CommonEdge1, CommonEdge3
    if CommonPoint exists then
        OtherPoint = CommonEdge3's other point than CommonPoint
        Edge = Quad2s edge containing OtherPoint and not containing CommonPoint
        p5 = OtherPoint + (length of Edge) * 3/8
        Edge = Quad2's edge containing CommonPoint and not containing OtherPoint
```

```
32:       p6 = CommonPoint + (length of Edge) * 1/8
33:       Edge = Pair.Secondss edge containing CommonPoint and not containing
          OtherPoint
34:       p7 = CommonPoint + (length of Edge) * 1/8
35:       Edge = Pair.Secondss edge containing OtherPoint and not containin Com-
          monPoint
36:       p8 = OtherPoint + (length of Edge) * 3/8
37:       Add points p5, p6, p7, p8 to BowTieMesh if they are not added before
38:       HalfBowTie = empty set of ordered points creating bow tie
39:       Add ordered points p5, p6, p7, p8 to HalfBowTie
40:       if Quad2's normal vector and HalfBowTie's normal vector are in the opposite
          directions then
41:          Reverse points of HalfBowTie
42:       end if
43:       Add HalfBowTie to BowTieMesh
44:    end if
45: end if
```

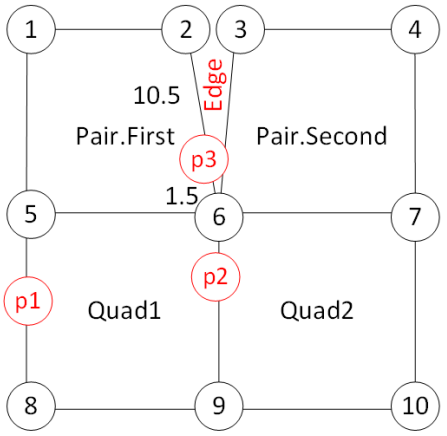


(a)

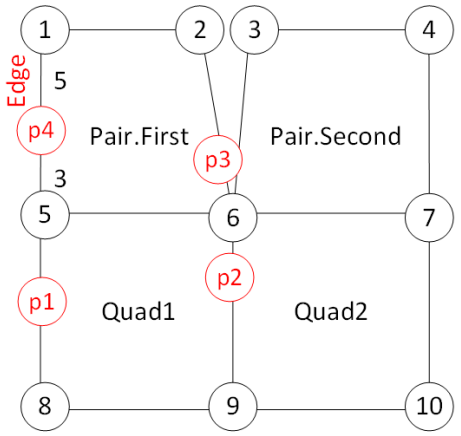
Figure 7: Execution steps of the INSERTHALFBOWTIE algorithm



BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 5
Edge = [6,9]



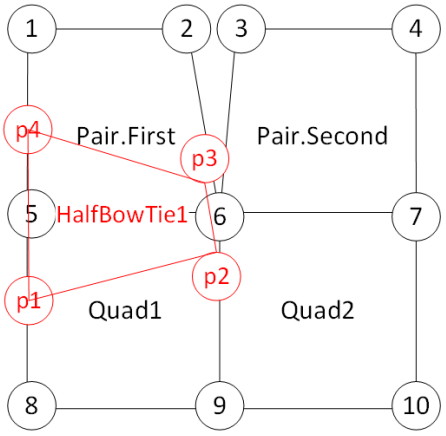
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 5
Edge = [2,6]



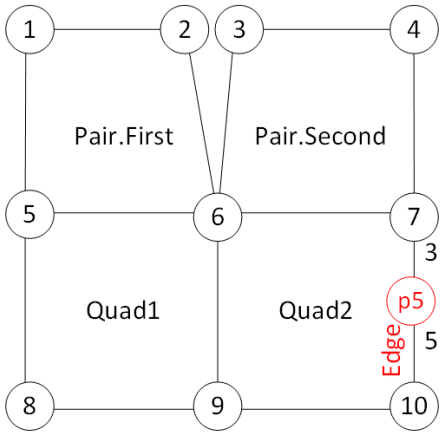
BowTieMesh = { Nodes{ }, Edges{ }, BowTies{ } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 5
Edge = [1,5]

(b)

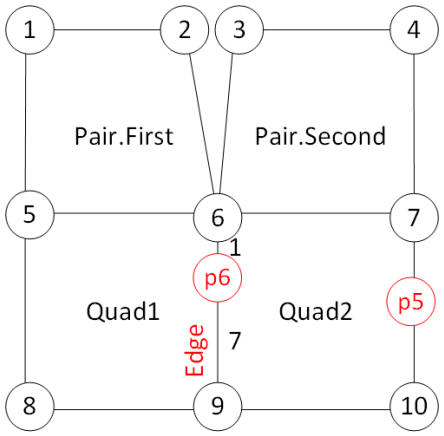
Figure 7: Execution steps of the INSERTHALFBOWTIE algorithm (cont.)



BowTieMesh = { Nodes{ p1, p2, p3, p4 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1] },
BowTies{ HalfBowTie1 } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 5



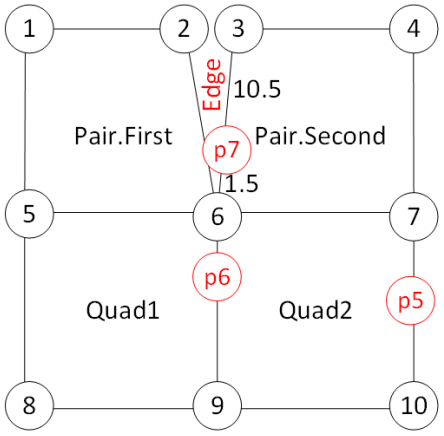
BowTieMesh = { Nodes{ p1, p2, p3, p4 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1] },
BowTies{ HalfBowTie1 } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 7
Edge = [7,10]



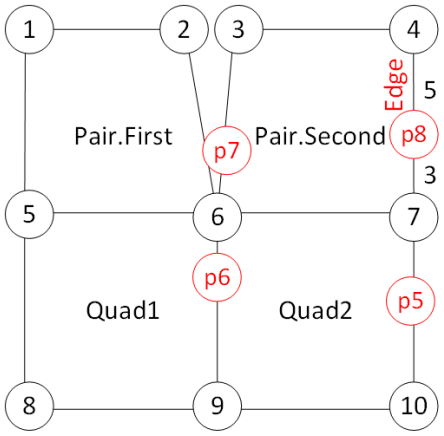
BowTieMesh = { Nodes{ p1, p2, p3, p4 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1] },
BowTies{ HalfBowTie1 } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 7
Edge = [6,9]

(c)

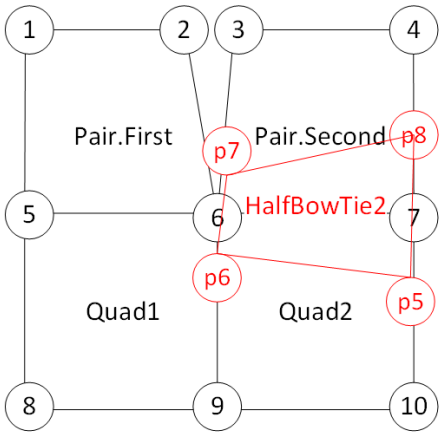
Figure 7: Execution steps of the INSERTHALFBOWTIE algorithm (cont.)



BowTieMesh = { Nodes{ p1, p2, p3, p4 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1] },
BowTies{ HalfBowTie1 } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 7
Edge = [3,6]



BowTieMesh = { Nodes{ p1, p2, p3, p4 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1] },
BowTies{ HalfBowTie1 } }
QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }
CommonEdge1 = [6,9]
CommonEdge2 = [5,9]
CommonEdge3 = [6,7]
CommonPoint = 6
OtherPoint = 7
Edge = [4,7]



BowTieMesh = { Nodes{ p1, p2, p3, p4, p5, p6,
p7, p8 },
Edges{ [p1,p2], [p2,p3], [p3,p4], [p4,p1],
[p5,p6], [p6,p7], [p7,p8], [p8,p4], },
BowTies{ HalfBowTie1, HalfBowTie2 } }

QuadMesh = { Nodes{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
Edges{ [1,2], [3,4], [1,5], [2,6], [3,6], [4,7], [5,6],
[6,7], [5,8], [6,9], [7,10], [8,9], [9,10] },
Quads{ Quad1, Quad2, Pair.First, Pair.Second } }

(d)

Figure 7: Execution steps of the INSERTHALFBOWTIE algorithm (cont.)