This is a post-print version of the article
ERRATA CORRIGE ON "MODELING AND COMPUTING TERNARY PROJECTIVE RELATIONS BETWEEN REGIONS"
Eliseo Clementini, Roland Billen and Marco Santic
IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING
NOVEMBER 2011
PP 7-15
ORIGINAL PUBLICATION CAN BE FOUND AT THIS ADDRESS : HTTP://DOI.IEEECOMPUTERSOCIETY.ORG/10.1109/TKDE.2011.239

Errata Corrige on "Modeling and Computing Ternary Projective Relations Between Regions"

Eliseo Clementini, Roland Billen, and Marco Santic

We report a corrected version of the algorithms to compute ternary projective relations between regions appeared in [1]. Not all the algorithms were affected by errors, but only some special cases that were treated by particular functions (on pages 810-811).The affected functions were "NN_Case_Before_After", "Treat_Between_Zone", "BT_Case_Before_After", and "BT_Case_Leftside_Rightside". The function "NN_Case_Before_After" and "Treat_Between_Zone" should be changed by the functions with the same name as listed afterwards. The functions "BT_Case_Before_After" and "BT_Case_Leftside_Rightside" are instead to be replaced by new functions "Case_Between_Before", "Case_Between_After", "Case_Between_Leftside", and "Case_Between_Rightside". The computational complexity of the overall algorithm is not affected by these changes, which are merely a rearrangement of the conditions to be checked. The errors were discovered thanks to a new implementation and experiments performed on polygons of various shapes, while the previous implementation was tested on a limited number of simplified shapes.
The corrected version of the algorithm has been checked against all possible significant configurations and therefore we can be sure that all errors have been found out. Providing a full proof of the correctness of the algorithms would be out of the scope of this errata corrige. Nonetheless, we discuss the basic strategy that has been used. By possible significant configurations we mean the geometric configurations that produce a change in the projective relation. There is a finite number of such geometric configurations: consider the case of a segment a_1a_2 with an endpoint in *Between* zone and an endpoint in *Leftside* zone (Fig.1). The algorithms in this case need to assess whether the segment intersects *After* and *Before* zones as well. Let us divide the *Between* zone in four parts as determined by the internal tangents: considering the position of endpoint a_1 in each of these four parts, we enumerate the possible positions (*leftside* or *rightside*) of the segment with respect to the four points *r*,*s*,*u*,*v* (see Fig.1). Once obtained the possible configurations of a segment, it suffices to check whether the algorithm is correct. The same procedure can be applied to identify the significant positions of segments for other combinations of the positions of endpoints in the five zones. The corrected functions are following.

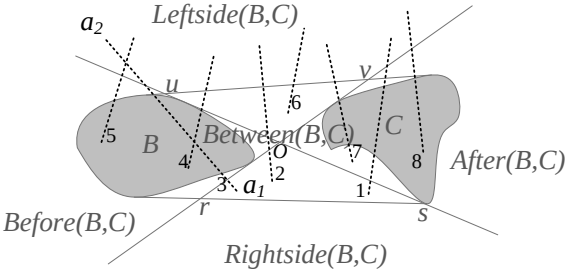


Fig.1. The possible configurations (dotted lines) of segment a_1a_2 bridging *Between*(*B*,*C*) and *Leftside*(*B*,*C*) zones. The *Between*(*B*,*C*) zone is divided in four parts by the internal tangents, identified by the angles *rOs*, *uOr*, *vOu*, *sOv*. If the endpoint a_1 is inside the angle *rOs*, there are three possible configurations of the segment (labels 1,2,3): for configuration 1, *ls*(*v*,a_1,a_2) and *ls*(*u*,a_1,a_2) hold; for configuration 2, *rs*(*v*,a_1,a_2) and *ls*(*u*,a_1,a_2) hold; for configuration 3, *rs*(*v*,a_1,a_2) and *rs*(*u*,a_1,a_2) hold. Analogously, there are two configurations (labels 4,5) for angle *uOr*, one configuration (label 6) for angle *vOu*, and two configurations (labels 7,8) for angle *sOv*.

```
function NN_Case_Before_After
begin
    if pos = bf then {firstvertex= a_{i-1}; secondvertex= a_i }
    else /* pos = af */
        {firstvertex= a_i ; secondvertex= a_{i-1}};
    if Check_Intersect(firstvertex, secondvertex,
       CH(B∪C))
    then Update_5int(bt);
    if ls(r, firstvertex, secondvertex) or
        ls(s, firstvertex, secondvertex)
    then Update_5int(rs)
    else if rs(u, firstvertex, secondvertex)
        or rs(v, firstvertex, secondvertex)
        then Update_5int(ls)
end;

function Treat_Between_Zone
begin
    if (pos = bf) or (posnext = bf) then
        if not Check_Matrix(ls, rs, af)
        then Case_Between_Before else;
    if (pos = af) or (posnext = af) then
        if not Check_Matrix(ls, rs, bf)
        then Case_Between_After else;
    if (pos = ls) or (posnext = ls)
        if not Check_Matrix(bf, af)
        then Case_Between_Leftside else;
    if (pos = rs) or (posnext = rs)
```

- *E. Clementini is with the Dept. of Electrical and Information Eng., University of L'Aquila, L'Aquila, Italy. E-mail: eliseo.clementini@univaq.it.*
- *R. Billen is with the Dept. of Geography, University of Liege, Liege, Belgium. E-mail: rbillen@ulg.ac.be.*
- *M. Santic is with the Dept. of Electrical and Information Eng., University of L'Aquila, L'Aquila, Italy. E-mail: marco.santic@westaquila.com.*

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.

```
        if not Check_Matrix(bf, af)
        then Case_Between_Rightside else;
end;

function Case_Between_Before
begin
    if pos = bf then {firstvertex= a_{i-1}; secondvertex= a_i }
    else /* posnext = bf */
        {firstvertex= a_i ; secondvertex= a_{i-1}};
    if rs(secondvertex, r, v) then
        if ls(r, firstvertex, secondvertex)
        then
        {  Update_5int(rs);
           if  ls(s, firstvertex, secondvertex)
           then Update_5int(af);
        }
    if ls(secondvertex, u, s) then
        if rs(u, firstvertex, secondvertex)
        then
        {   Update_5int(ls);
            if rs(v, firstvertex, secondvertex)
            then Update_5int(af);
        }
end;

function Case_Between_After
begin
    if posnext = af then
        {firstvertex= a_{i-1}; secondvertex= a_i }
    else /* pos = af */
        {firstvertex= a_i ; secondvertex= a_{i-1}};
    if rs(firstvertex, u, s) then
        if ls(s, firstvertex, secondvertex)
        then
        {  Update_5int(rs);
           if  ls(r, firstvertex, secondvertex)
           then Update_5int(bf);
        }
    if ls(firstvertex, r, v) then
        if rs(v, firstvertex, secondvertex)
        then
        {  Update_5int(ls);
           if rs(u, firstvertex, secondvertex)
           then Update_5int(bf);
        }
end;

function Case_Between_Leftside
begin
    if posnext = ls then
        {firstvertex= a_{i-1}; secondvertex= a_i }
    else /* pos = ls */
        {firstvertex= a_i ; secondvertex= a_{i-1}};
    if rs(u, firstvertex, secondvertex)
    then Update_5int(bf);
    if ls(v, firstvertex, secondvertex)
    then Update_5int(af);
end;

function Case_Between_Rightside
begin
 if pos = rs then {firstvertex= a_{i-1}; secondvertex= a_i }
 else /* posnext = rs */
        {firstvertex= a_i ; secondvertex= a_{i-1}};
 if rs(r, firstvertex, secondvertex)
 then Update_5int(bf);
 if ls(s, firstvertex, secondvertex)
 then Update_5int(af);
end;
```

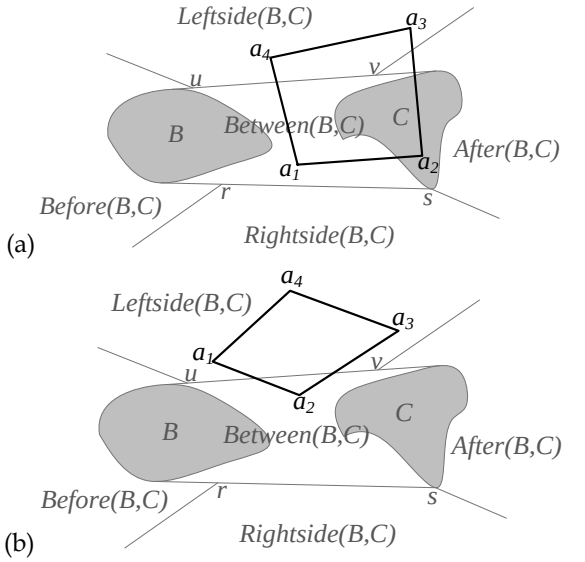


Fig.2. Geometric configurations illustrating the special case *Between* and *Leftside*.

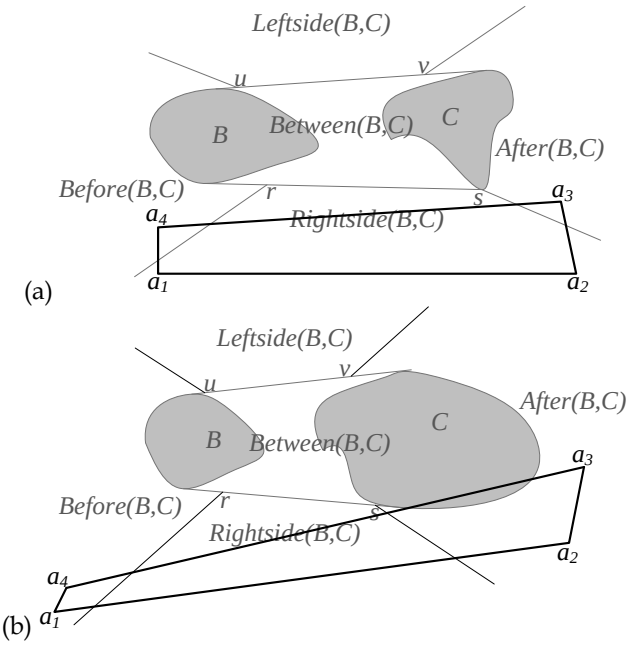


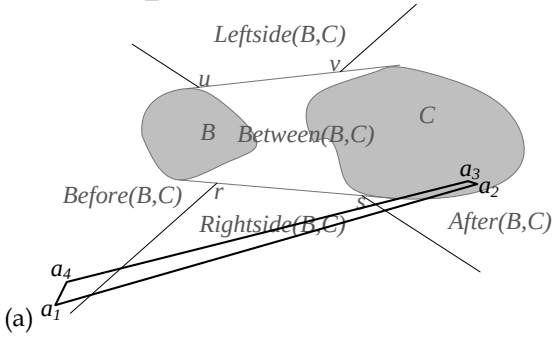
Fig. 3. Geometric configurations illustrating the special case *Before* and *After*.

Regarding the old function BT_Case_Leftside_Rightside, it wrongly included the relations *before* and *after* in some configurations. To illustrate this case, both in Fig.2(a) and Fig.2(b), relations *between* and *leftside* hold because there

This is a post-print version of the article
ERRATA CORRIGE ON "MODELING AND COMPUTING TERNARY PROJECTIVE RELATIONS BETWEEN REGIONS"
Eliseo Clementini, Roland Billen and Marco Santic
IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING
NOVEMBER 2011
PP 7-15
ORIGINAL PUBLICATION CAN BE FOUND AT THIS ADDRESS : HTTP://DOI.IEEECOMPUTERSOCIETY.ORG/10.1109/TKDE.2011.239

are some vertices falling in both *Between* and *Leftside* zones, as it is assessed by Algorithm 2. Also, Algorithm 4 is called (Treat_Special_Cases): one of the special cases is when one of the vertices falls inside the *Between* zone. Therefore, the function Treat_Between_zone is called: among other situations, this function checks whether, if there are consecutive vertices falling in zones *Between* and *Leftside* (e.g., in Fig.2(a) and (b), vertices a_2 and a_3), there is an intersection of the corresponding segment with *After* or *Before* zones. In Fig.2(a), such an intersection exists, while in Fig.2(b) it does not. The old algorithm could not correctly distinguish the conditions that apply when the segment crosses the *Between* and *Leftside* zones from the conditions that apply when the segment crosses the *Between* and *Rightside* zones. Dealing with the conditions in two new separate functions Case_Between_Leftside and Case_Between_Rightside allowed us to solve the problem. In the old function, the result in the case of Fig.2(b) was *bt:bf:ls:af(A,B,C)* instead of *bt:ls(A,B,C)*, due to the fact that the condition *rs(s,a1,a2)* was verified and, therefore, the relation *after* was added; also, the condition *ls(r,a3,a2)* was verified and, therefore, the relation *before* was added.

The old function NN_Case_Before_After failed to include in the result the Between zone in a few configurations. In Fig. 3, we show two configurations related to the case where two consecutive vertices of polygon *A*, e.g., a_3 and a_4, fall inside the *Before* and *After* zones. In this case, Algorithm 4 makes a call to the function Treat_Non_Neighbor_Zone, which in turn makes a call to the function NN_Case_Before_After. This latter function in the original version correctly found the intersection of polygon *A* with the *Rightside* zone (Fig.3(a)), since both points *r* and *s* are *leftside* of points a_4 and a_3. Unfortunately, the function did not recognize the intersection with the *Between* zone in a similar situation (Fig.3(b)), giving the wrong result *rs:bf:af(A,B,C)*. The corrected NN_Case_Before_After function finds the result *bt:rs:bf:af(A,B,C)* for the configuration in Fig.3(b) with an additional Check_Intersect.



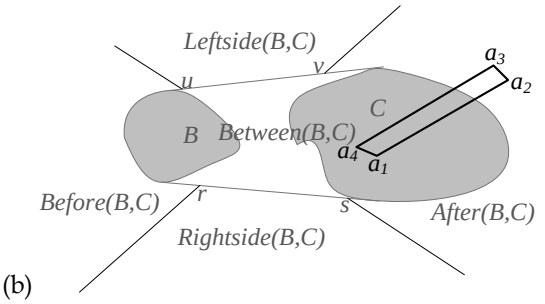


Fig. 4. Geometric configurations illustrating the special cases *Between* and *Before* (a) and *Between* and *After* (b).

The old function BT_Case_Before_After did not recognize the *before* and *after* relations in some cases and wrongly recognized the *rightside* and *leftside* relations in other cases. For example, in Fig. 4(a) we show a configuration where the function fails to add the relation *after* to the result. Only the relation *rightside* was added giving the result *bt:rs:bf(A,B,C)*. The new function Case_Between_Before adds the relation *after* as well, returning the result *bt:rs:bf:af(A,B,C)* for the configuration in Fig.4(a). Analogously, the function Case_Between_After solves the case where the old function BT_Case_Before_After failed to include the *before* relation. Another error of old function BT_Case_Before_After was a false recognition of the *Rightside* zone like in Fig.4(b) and of the *Leftside* zone as well in similar cases. The new functions Case_Between_After and Case_Between_Before give the correct result.

For the sake of completeness, we also update Algorithm 2 of [1] with a last check taking into consideration the case when the zone *Between*(*B*,*C*) is properly contained inside the region *A*. This case requires a point-in-polygon test between an arbitrary point belonging to $CH(B \cup C)$ and region *A* itself. A java implementation of the complete algorithms is available in [2].

Algorithm 2: Build 5-intersection.
Input: region *A*; $CH(B \cup C)$; internal tangents; intersections *r*,*s*,*u*,*v*;
Output: 5-intersection matrix;
begin
  $i \leftarrow 1$;
  pos $\leftarrow$ Check_Position(a_i, $CH(B \cup C)$, internal tangents);
  Update_5int(pos);
  $i \leftarrow i+1$;
  while $a_i \neq a_1$ **do**
    posnext $\leftarrow$ Check_Position(a_i, $CH(B \cup C)$, internal tangents);
    Update_5int(posnext);
    Treat_Special_Cases(a_{i-1}, a_i, pos, posnext, $CH(B \cup C)$, *r*,*s*,*u*,*v*);
    pos $\leftarrow$ posnext;
    $i \leftarrow i+1$;

endwhile
if 5-intersection matrix = (1 1 0 1 1 | 0 0) **then**
if Point_In_Polygon(Any_Point_In($CH(B \cup C)$), A)
then Update_5int(bt);
end

ACKNOWLEDGMENT

The authors are grateful to the anonymous referees for their helpful comments.

REFERENCES

[1] E. Clementini and R. Billen, "Modeling and computing ternary projective relations between regions," IEEE Transactions on Knowledge and Data Engineering, vol. 18, pp. 799-814, 2006.
[2] Java Projective Suite, "http://www.x-placer.com/kb/JavaProjectiveSuite/," 2011.