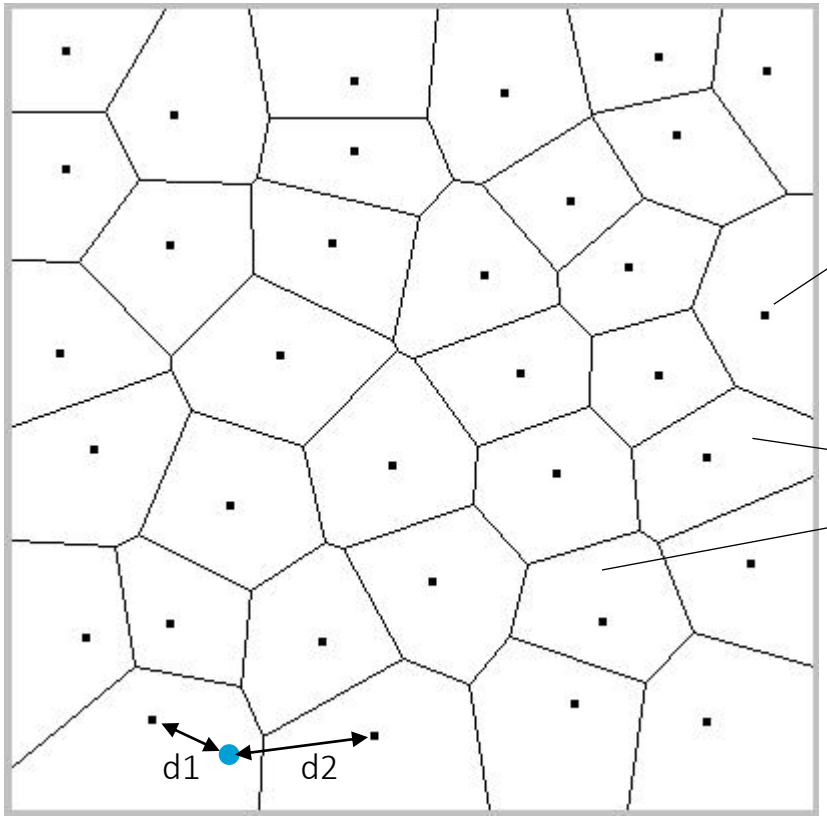
Implantation stratégique d'une école avec un diagramme de Voronoï

Matéo CHARLES-MENNIER

Filière MPI, années 2022-2023, SCEI : 48162

Sommaire

I) Implémentation du diagramme de Voronoï



Germe

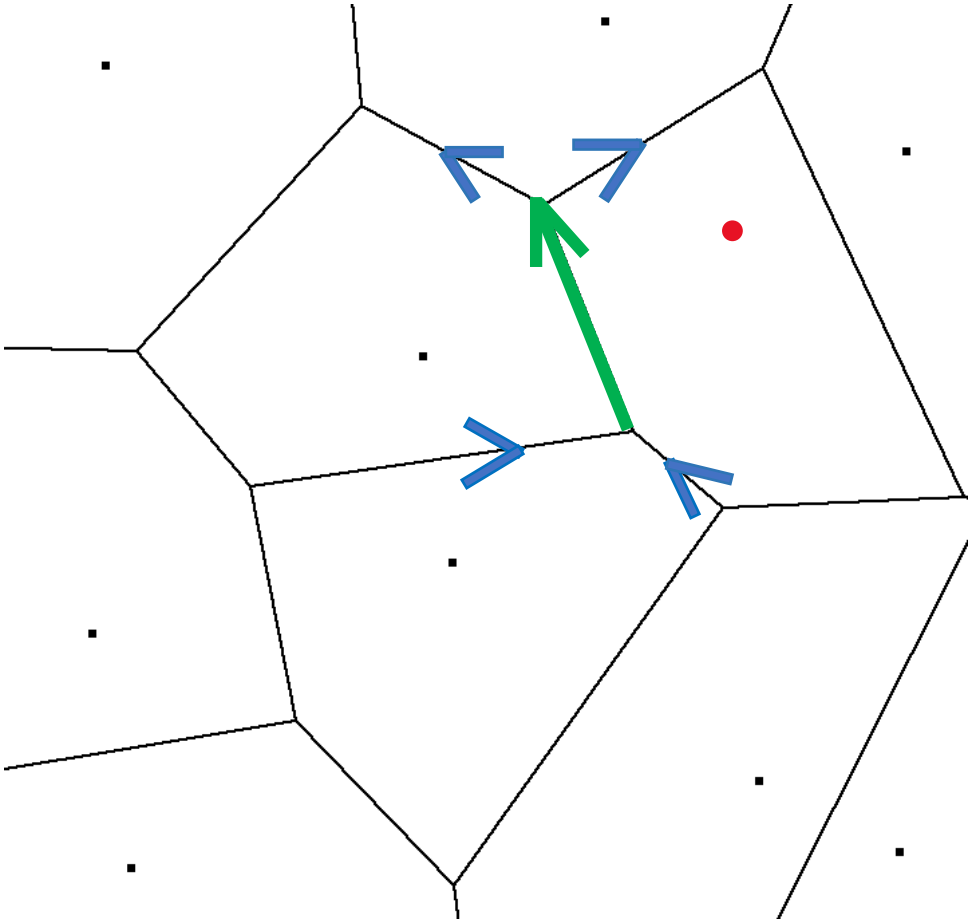
Cellules

Gueorgui Voronoï 1908

d1 < d2

I) Implémentation du diagramme de Voronoï

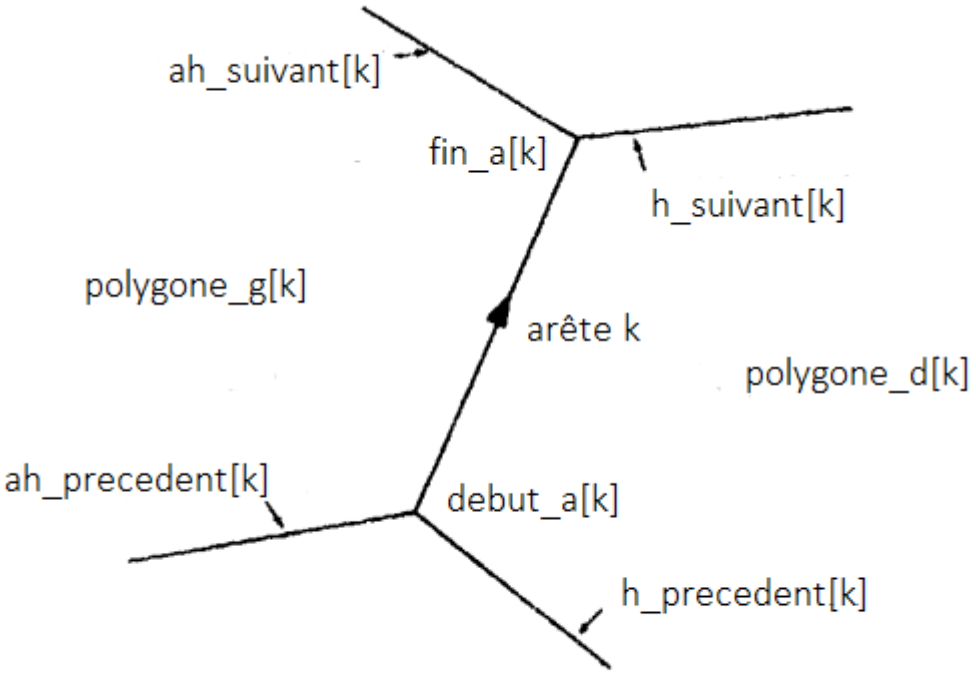
1. Structure de donnée « Winged-Edge »



- Germe
- Arête
- Sens de parcours

I) Implémentation du diagramme de Voronoï

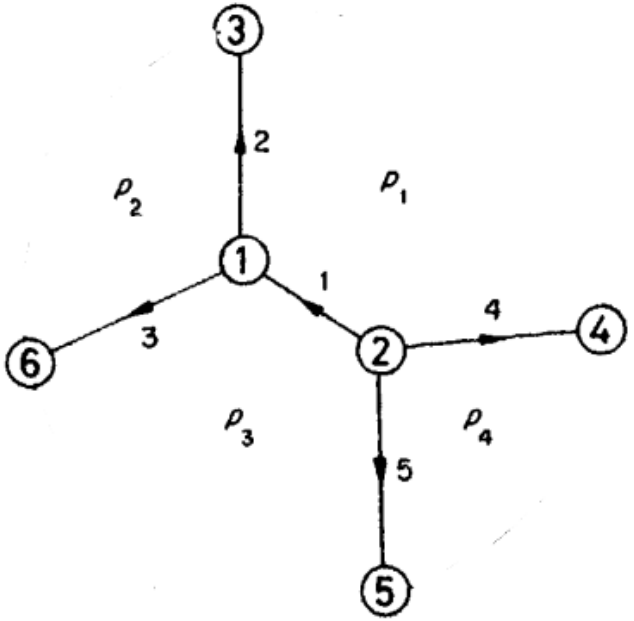
1. Structure de donnée « Winged-Edge »



Baumgart 1975

I) Implémentation du diagramme de Voronoï

1. Structure de donnée « Winged-Edge »



k (numéro d'arête)	1	2	3	4	5
polygone_d[k]	1	1	2	4	3
polygone_g[k]	3	2	3	1	4
debut_a[k]	2	1	1	2	2
fin_a[k]	1	3	6	4	5
h_precedent[k]	4	1	2	5	1
ah_precedent[k]	5	3	1	1	4
h_suivant[k]	3				
ah_suivant[k]	2				

i (numéro de polygone)	1	2	3	4
arete_autour_p[i]	[1,2,4]	[2,3]	[1,3,5]	[4,5]

Figure 1

I) Implémentation du diagramme de Voronoï

2.Méthode incrémentale pour le diagramme de Voronoï

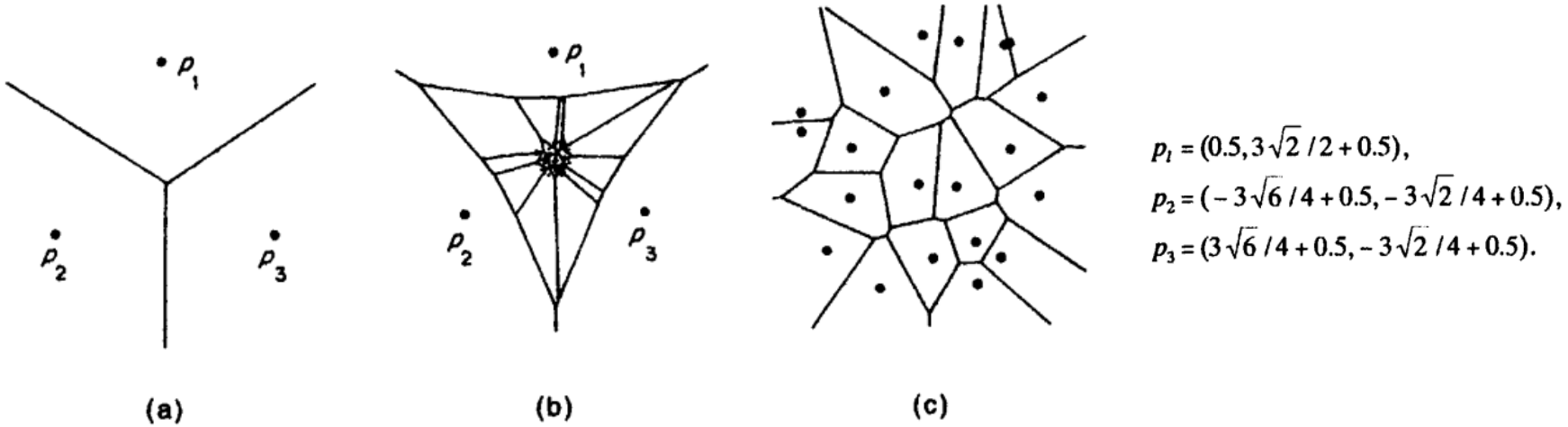


Figure 2

I) Implémentation du diagramme de Voronoï

2.Méthode incrémentale pour le diagramme de Voronoï

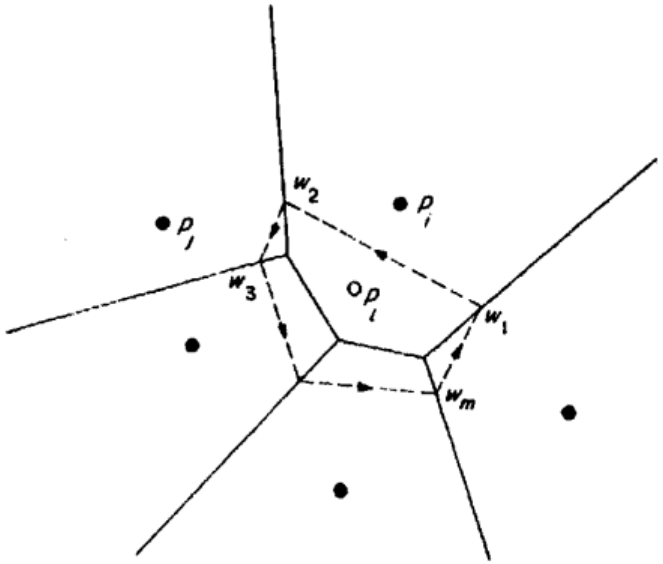
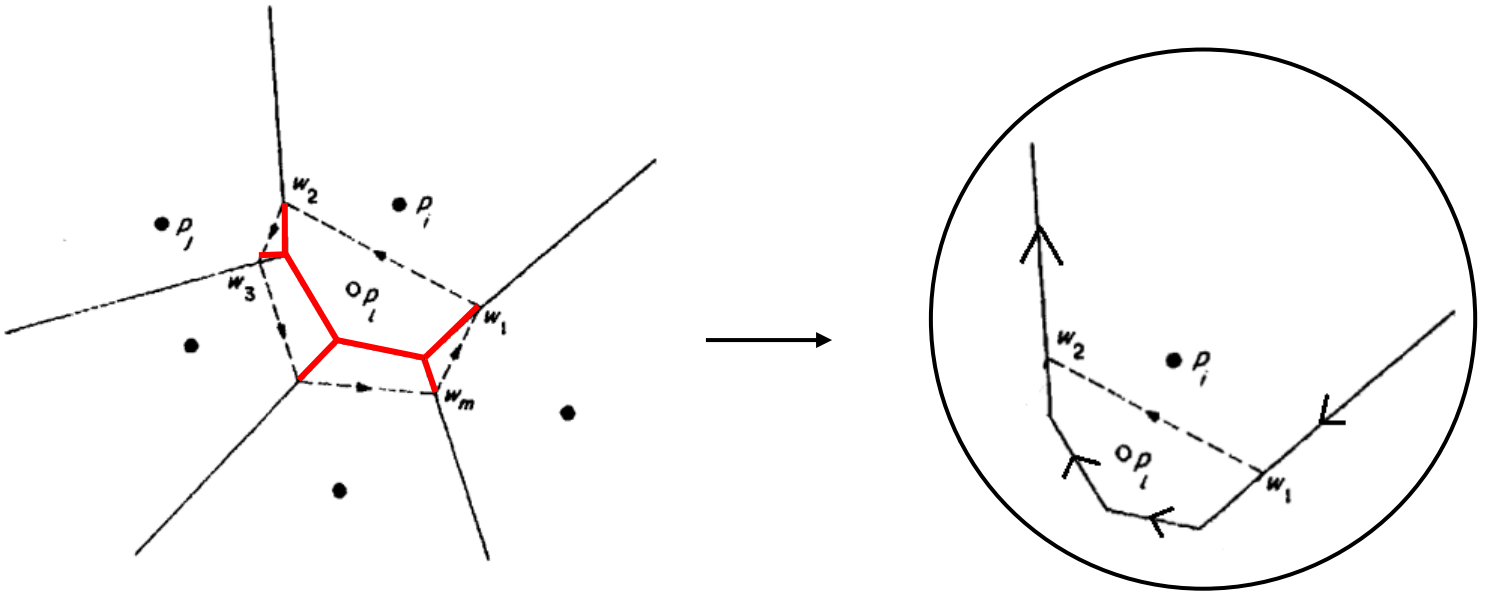


Figure 3

- Placer le nouveau germe
- Trouver dans quelle cellule il se trouve O(n)
- Tracer la médiatrice entre notre point (Pl) et celui de la cellule dans laquelle il est (Pi)
- Itérer les médiatrices avec les autres points (Pj) dans le sens anti-horaire O(n)

I) Implémentation du diagramme de Voronoï

2.Méthode incrémentale pour le diagramme de Voronoï

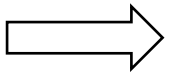


II) Optimisation de la complexité

$O(n^2)$

Recherche de cellules naïve

Placement des points dans un ordre quelconque



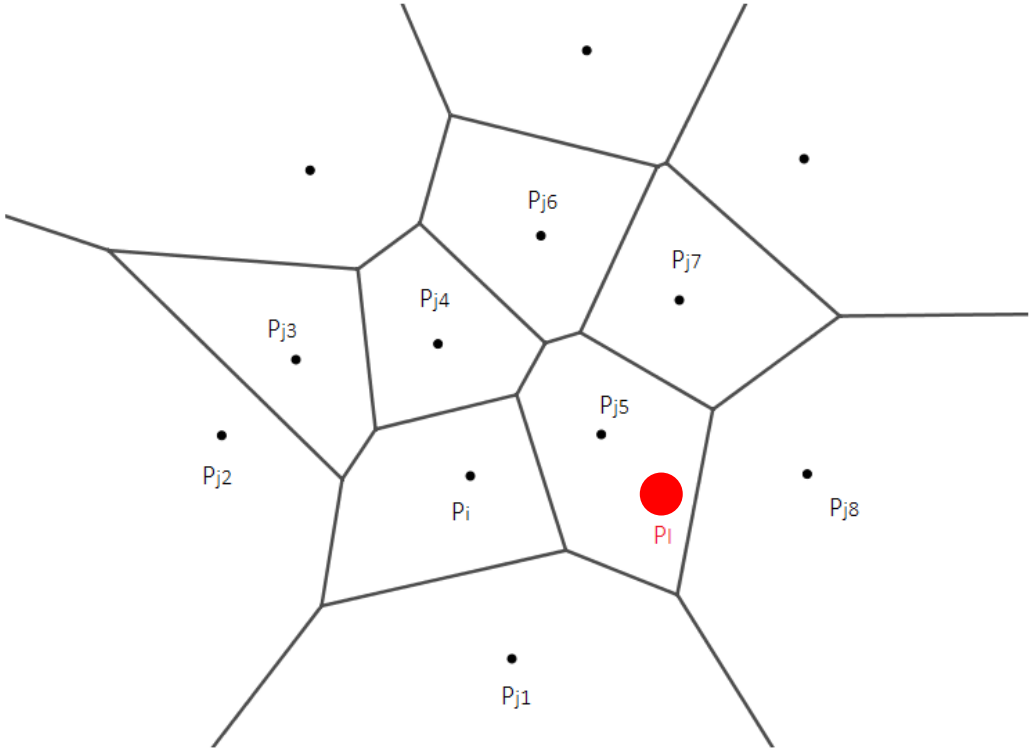
O(n) en moyenne

Recherche des plus proches voisins

Uniformisation du placement avec un arbre quaternaire

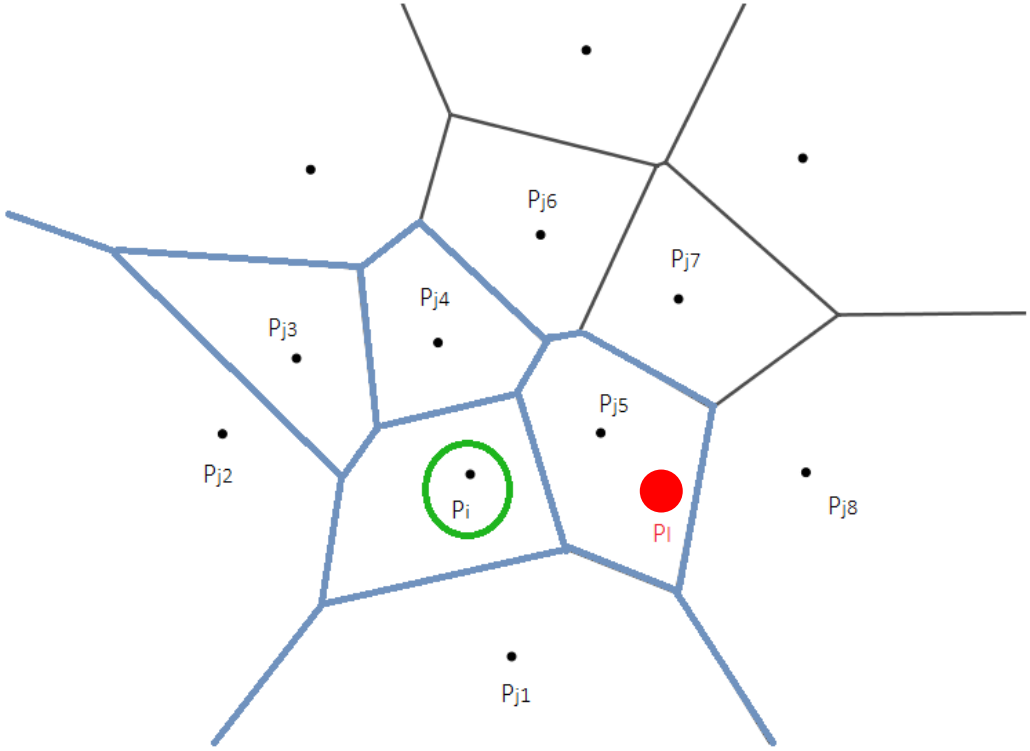
II) Optimisation de la complexité

1. Recherche des plus proches voisins



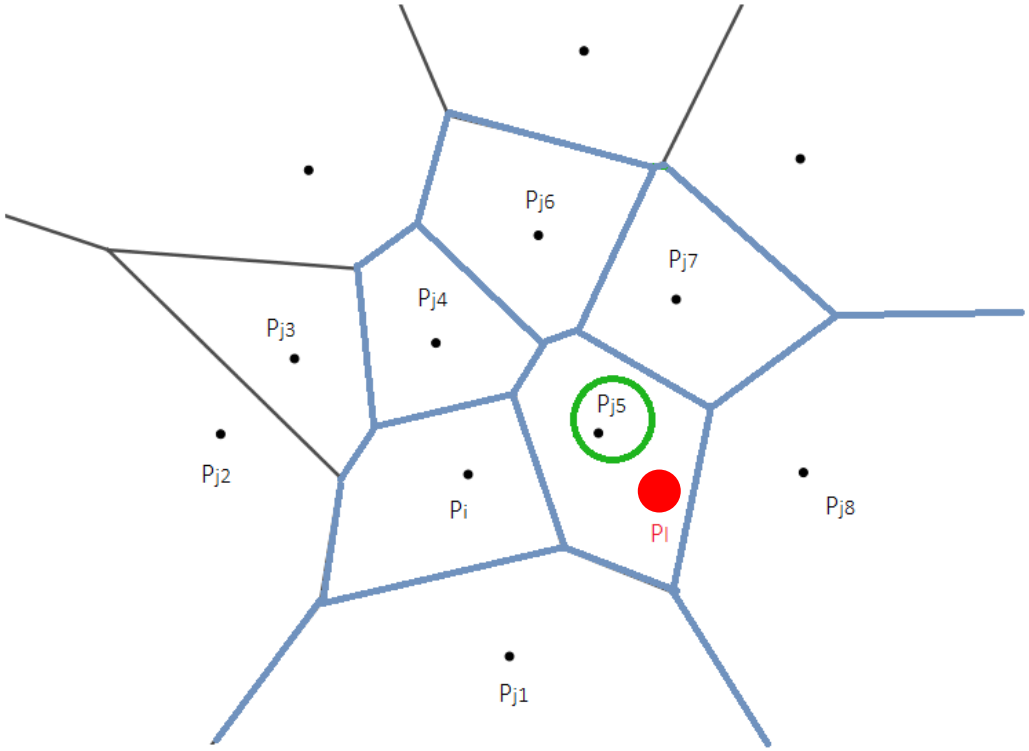
II) Optimisation de la complexité

1. Recherche des plus proches voisins



II) Optimisation de la complexité

1. Recherche des plus proches voisins



II) Optimisation de la complexité

2. Arbre quaternaire (quad-tree) et technique de « bucketing »

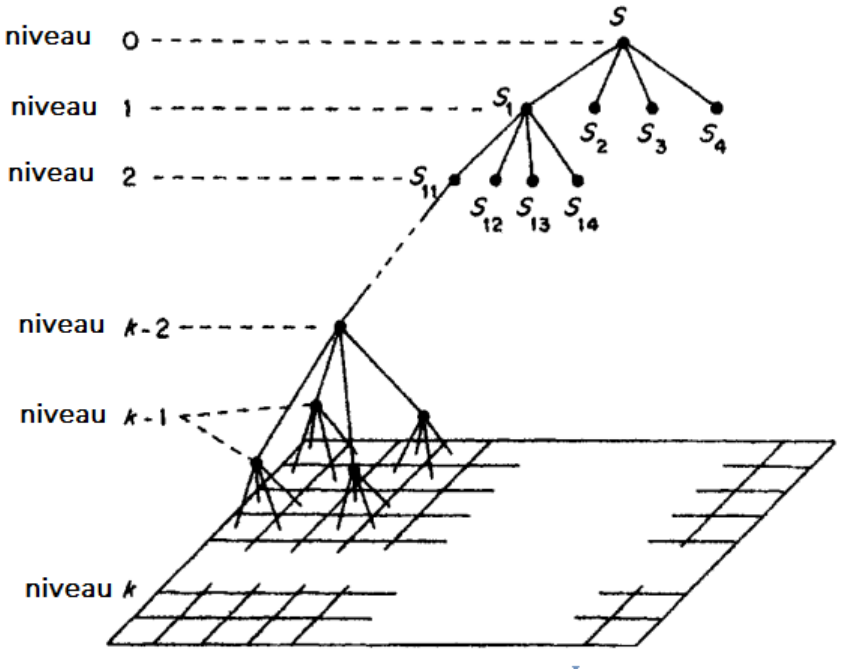
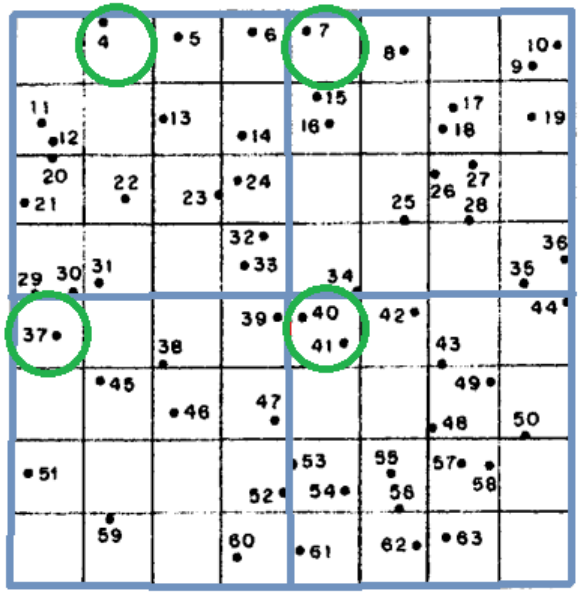
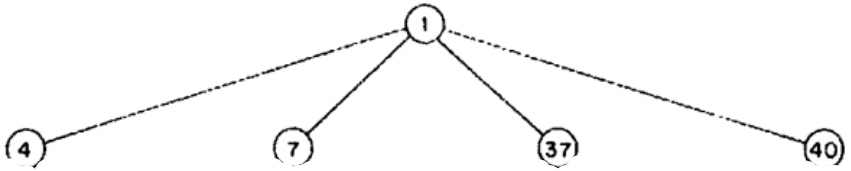


Figure 4

II) Optimisation de la complexité

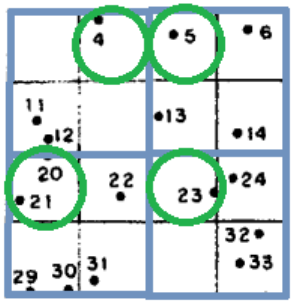
2. Arbre quaternaire (quad-tree) et technique de « bucketing »

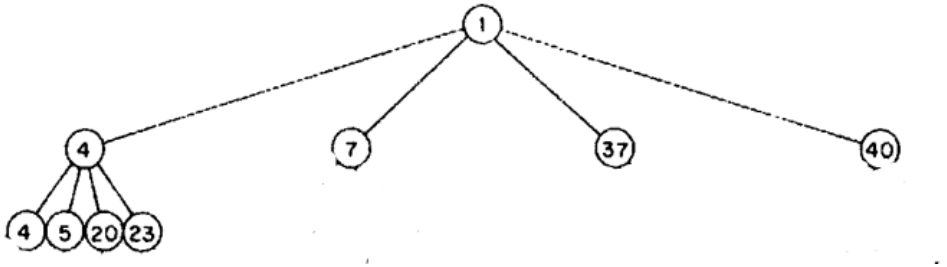




II) Optimisation de la complexité

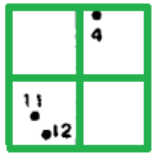
2. Arbre quaternaire (quad-tree) et technique de « bucketing »

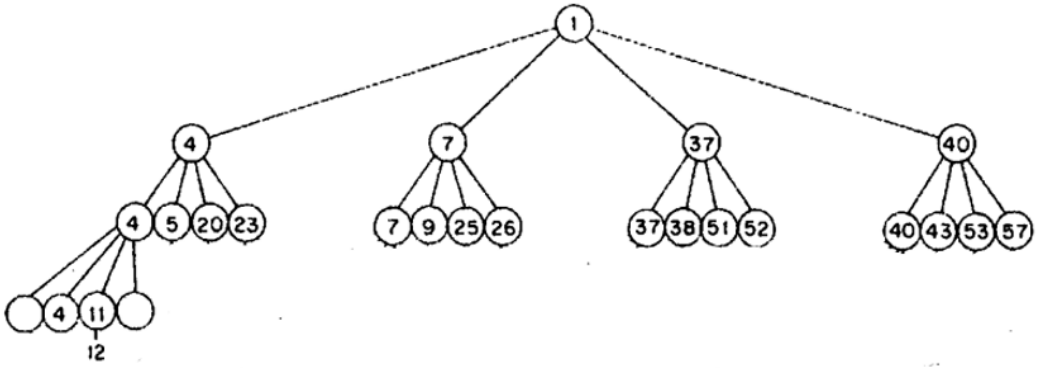




II) Optimisation de la complexité

2. Arbre quaternaire (quad-tree) et technique de « bucketing »





II) Optimisation de la complexité

2. Arbre quaternaire (quad-tree) et technique de « bucketing »

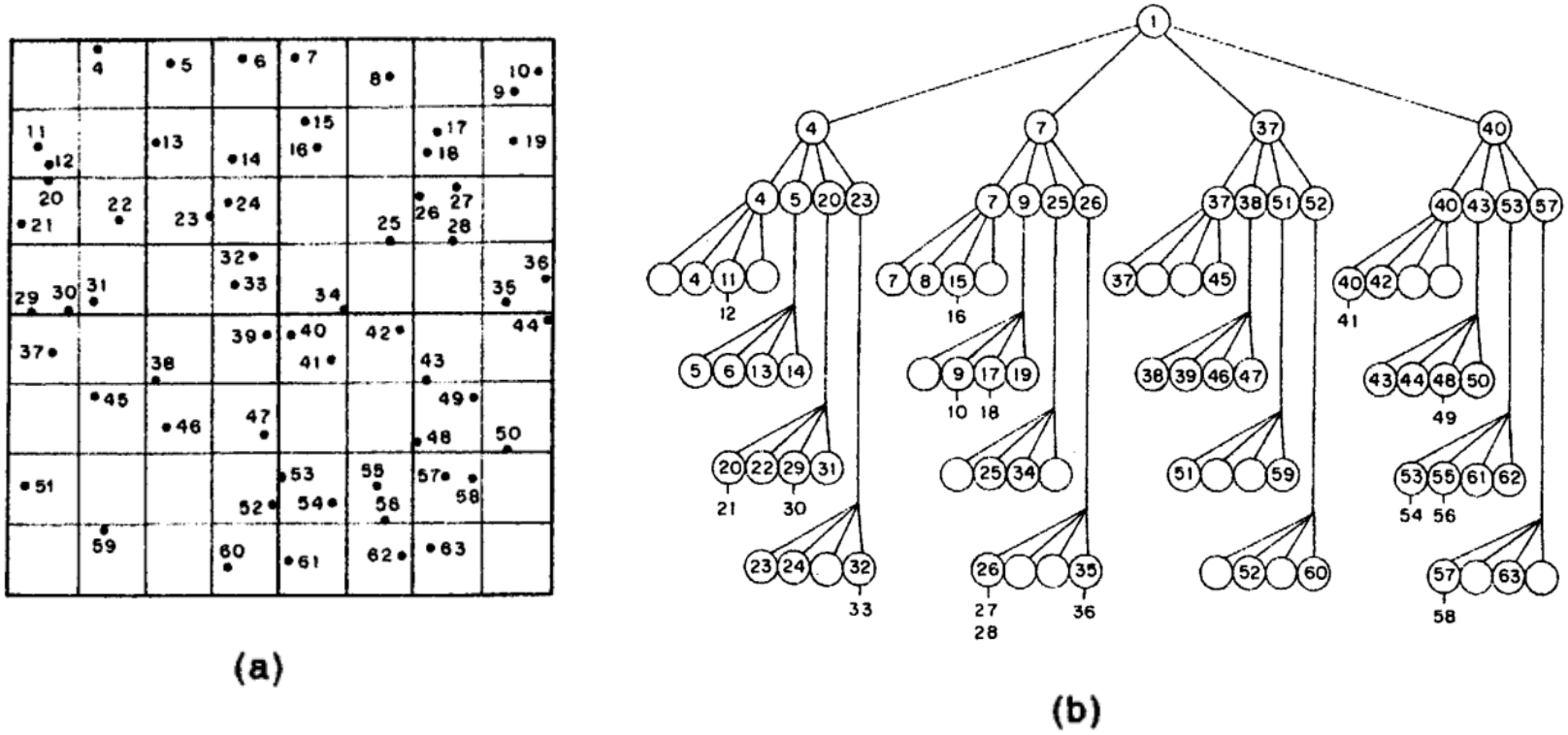
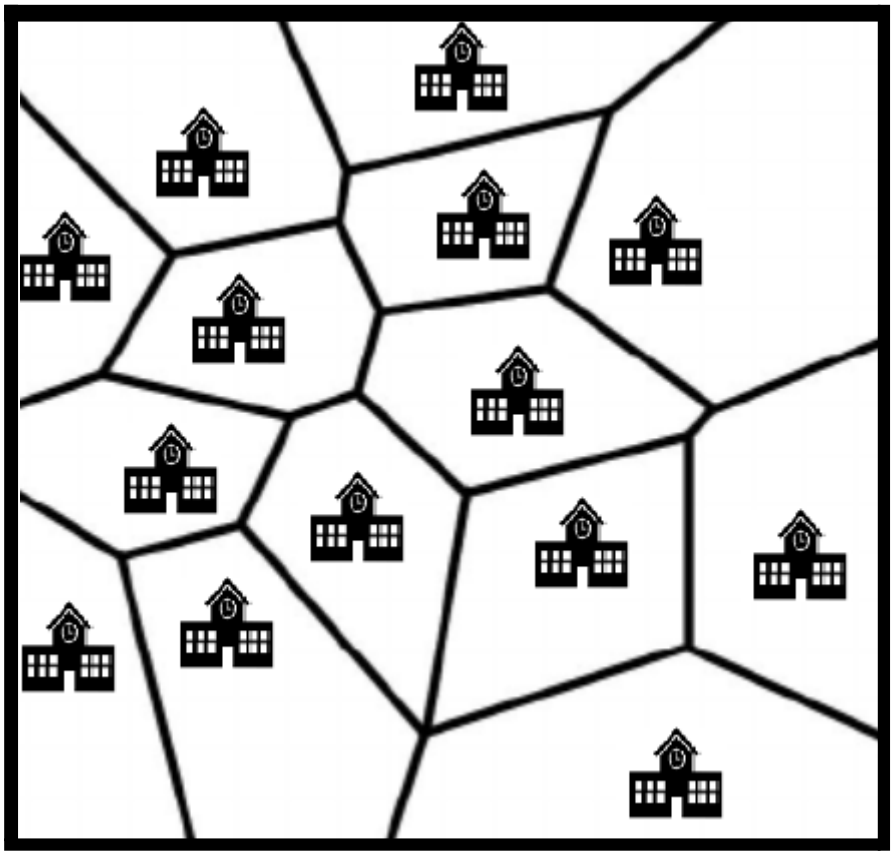


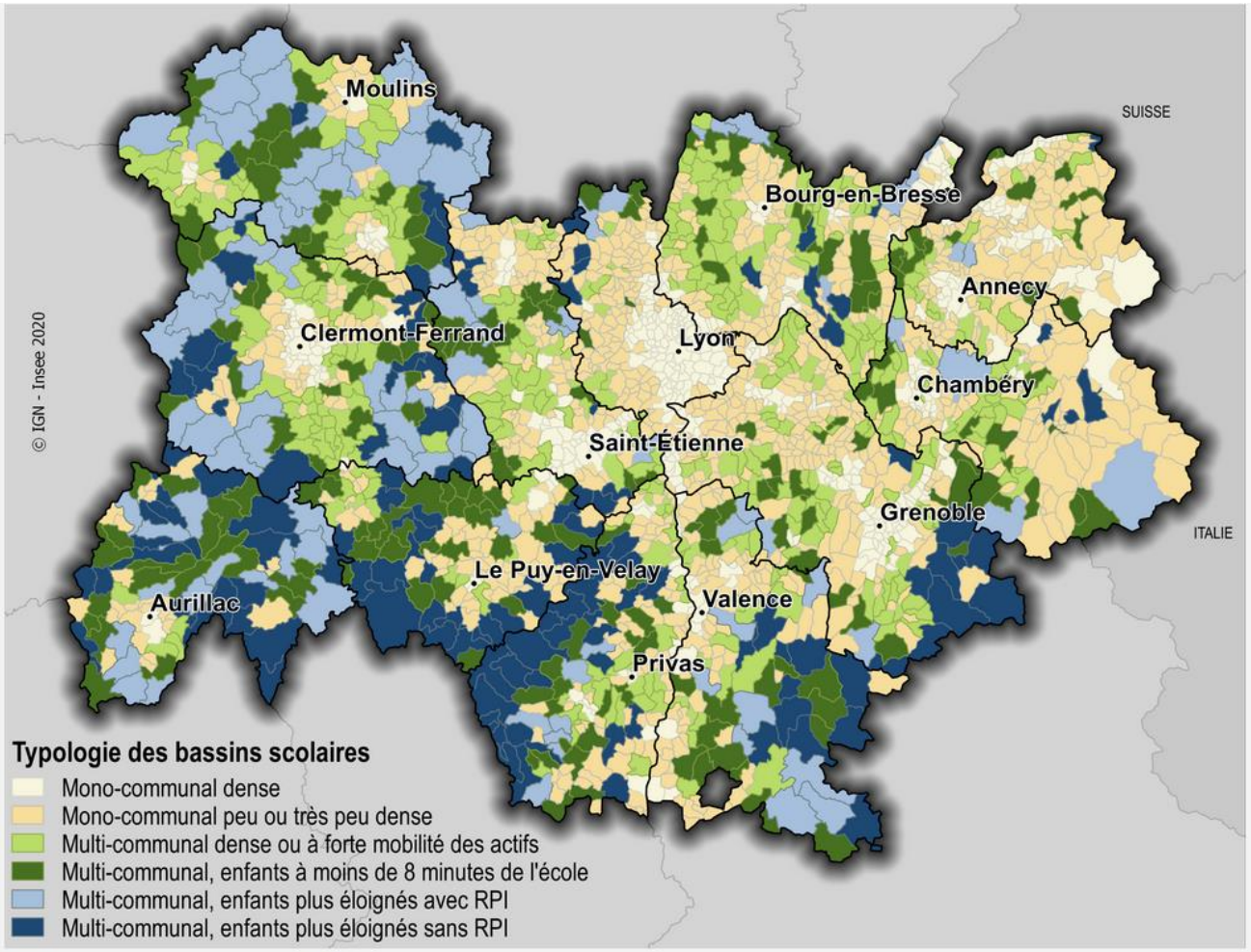
Figure 5

III) Application dans l'implémentation d'une école



III) Application dans l'implémentation d'une école

1. Application à l'Auvergne Rhône-Alpes



Sources : Insee, Recensement de la population 2015, DEPP

III) Application dans l'implémentation d'une école

1. Application à l'Auvergne Rhône-Alpes

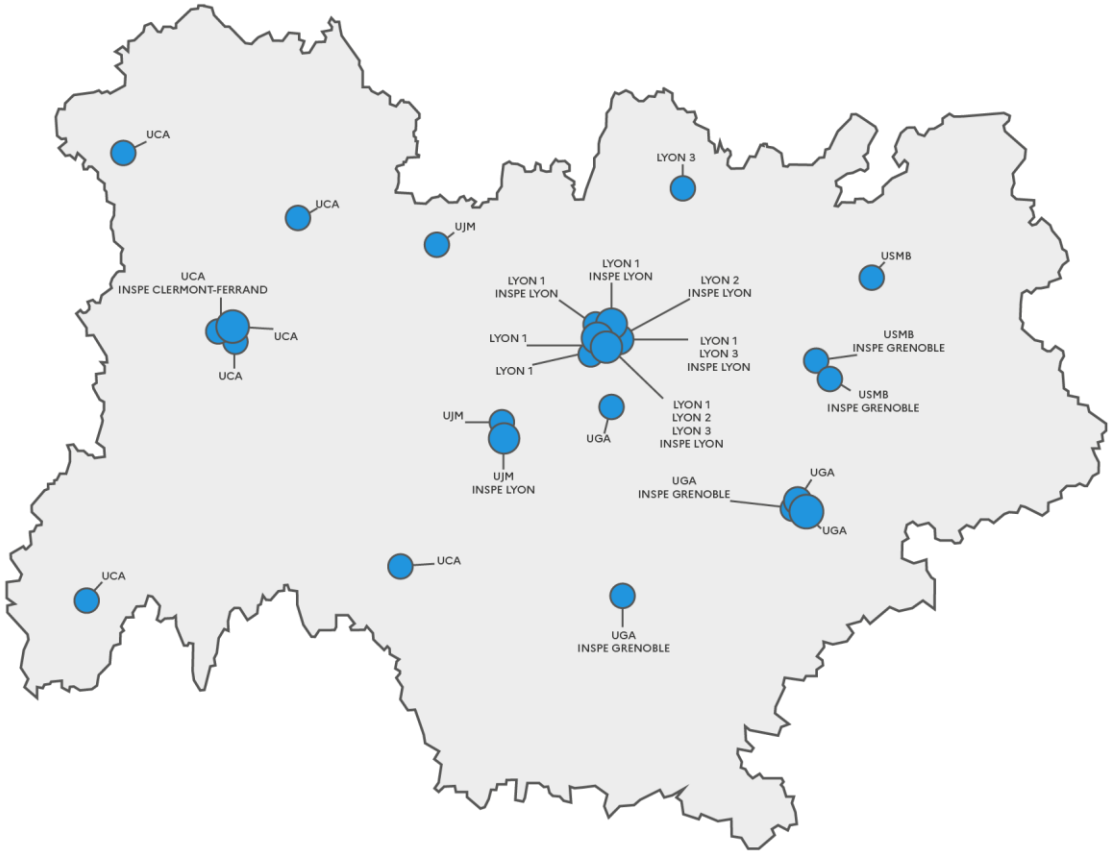
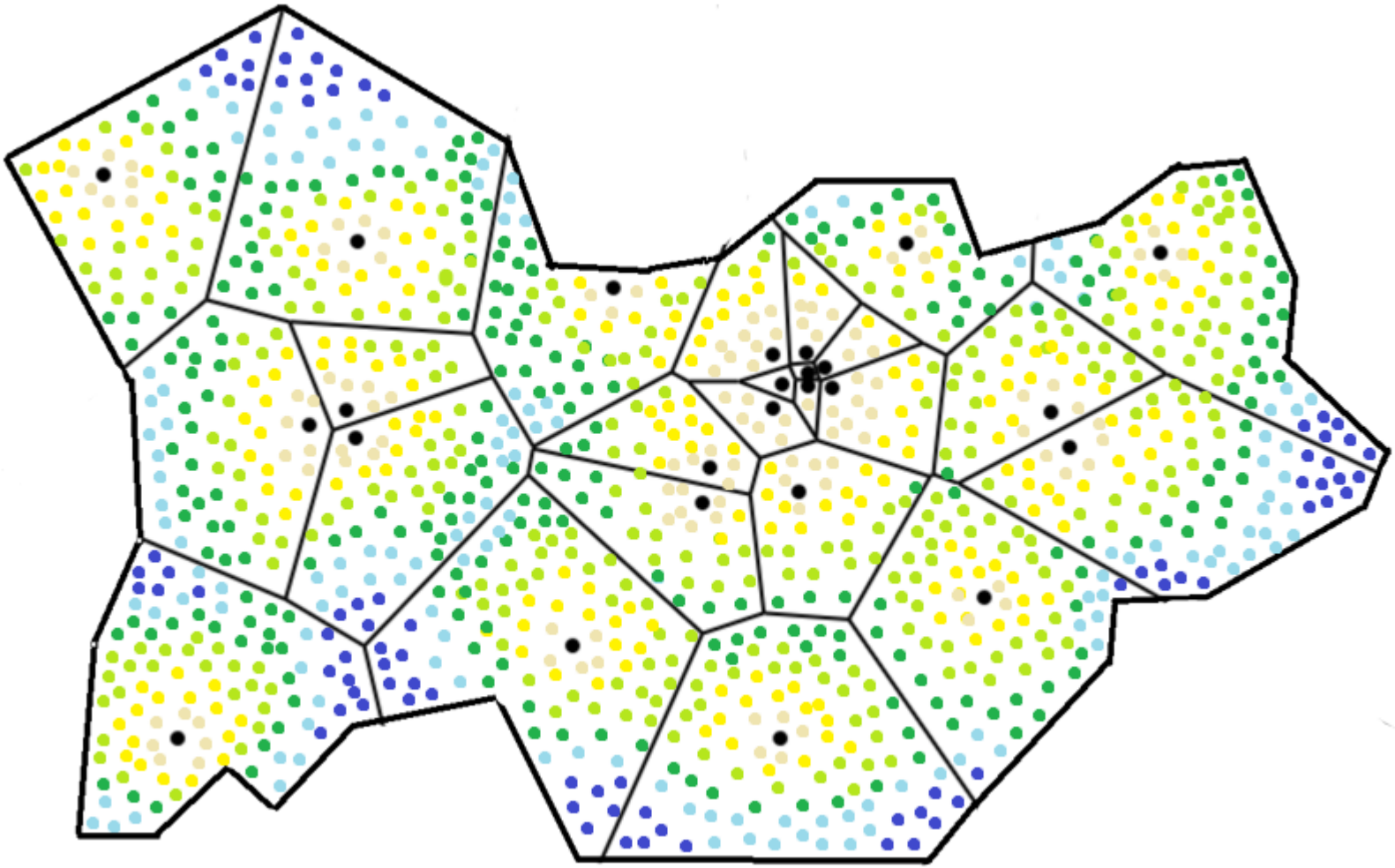


Figure 6

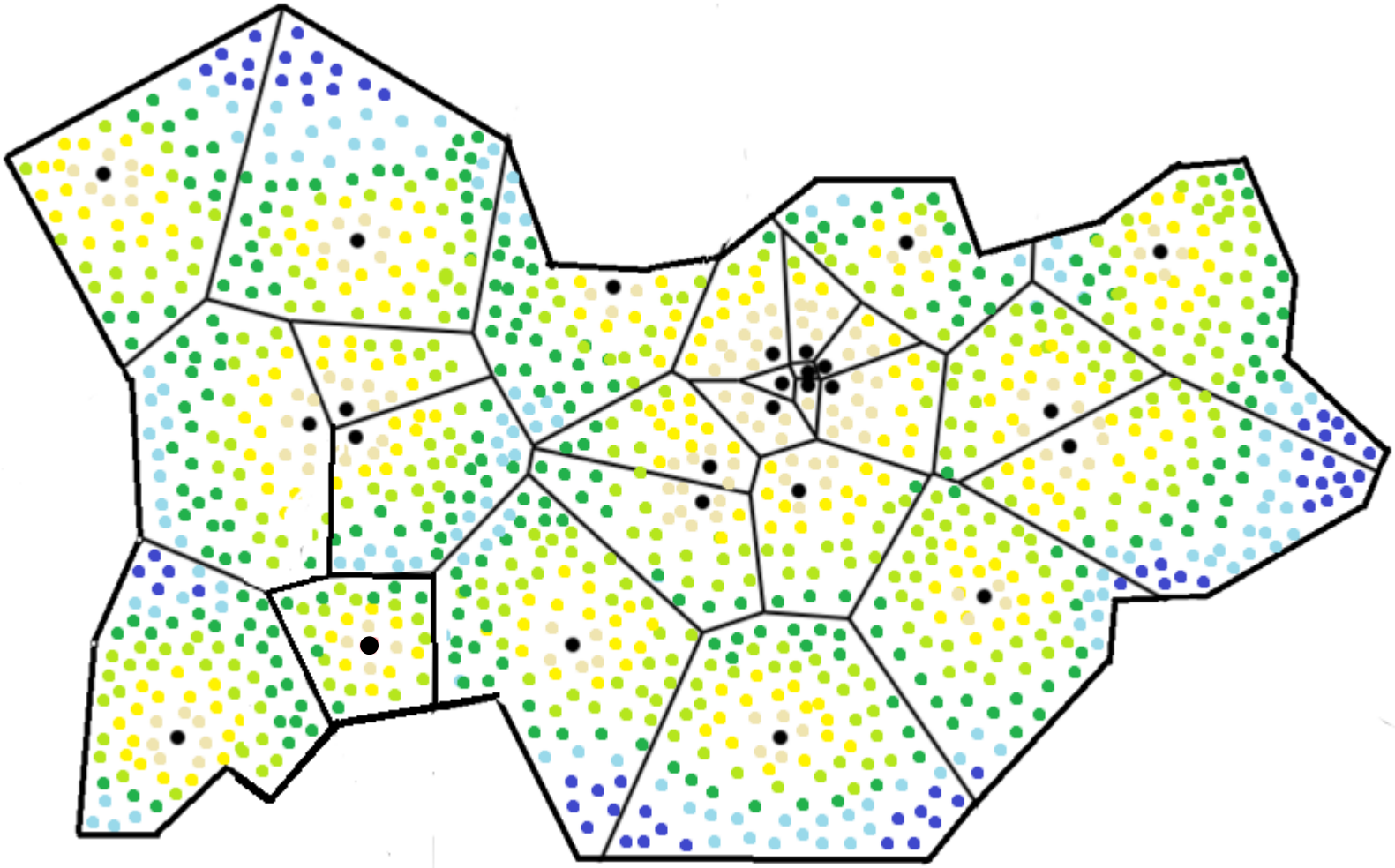
III) Application dans l'implémentation d'une école

1. Application à l'Auvergne Rhône-Alpes



III) Application dans l'implémentation d'une école

1. Application à l'Auvergne Rhône-Alpes



III) Application dans l'implémentation d'une école

2. Faiblesses du modèle

- Prise en compte de la densité de population.
- Prise en compte de la topologie.
- Distance à vol d'oiseau.

Conclusion

- Objectif : proposer une stratégie permettant l'implantation d'une école à l'emplacement idéal.
- Modèle : diagramme de Voronoï.
- Démarche : se servir du diagramme pour déterminer les zones les plus éloignées des centres d'enseignements.
- Résultat : modèle fonctionnel seulement si la proximité est le seul problème.
- Améliorations : ajouter les informations concernant la topologie et la densité de population.

Annexes

Contre-exemple de polygones adjacents

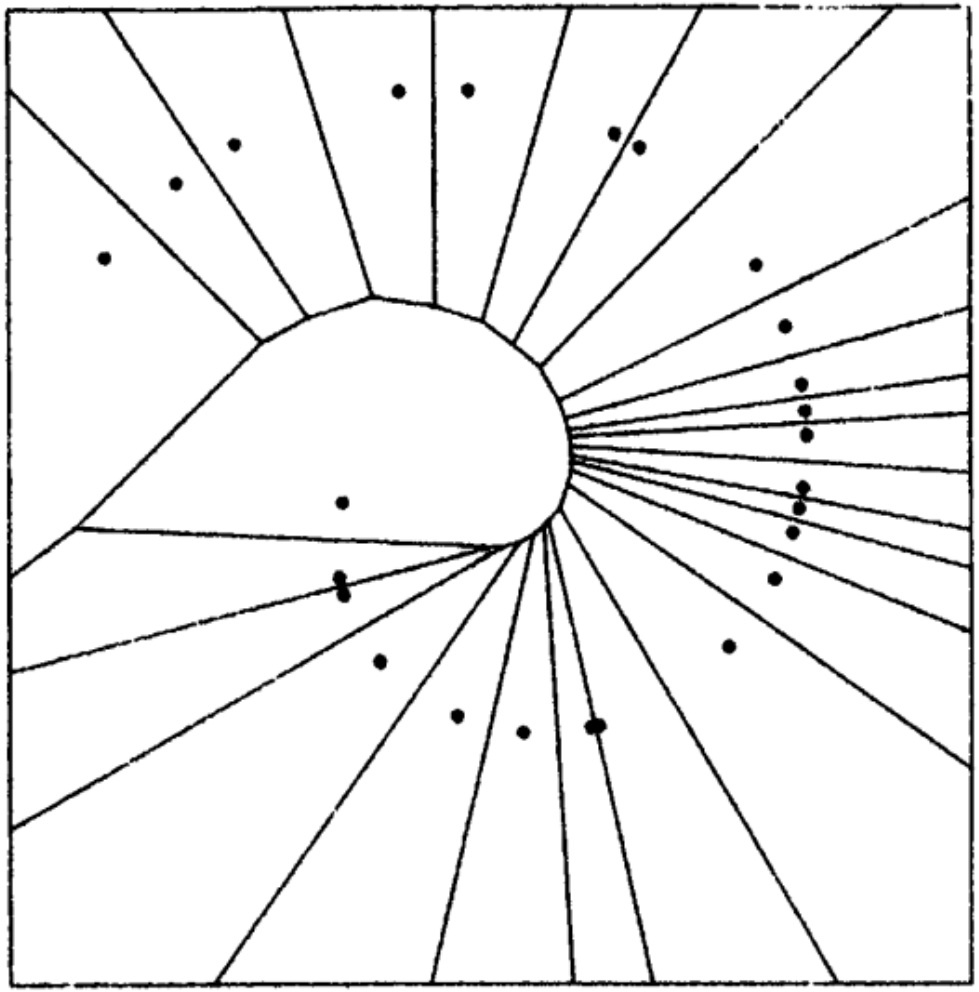
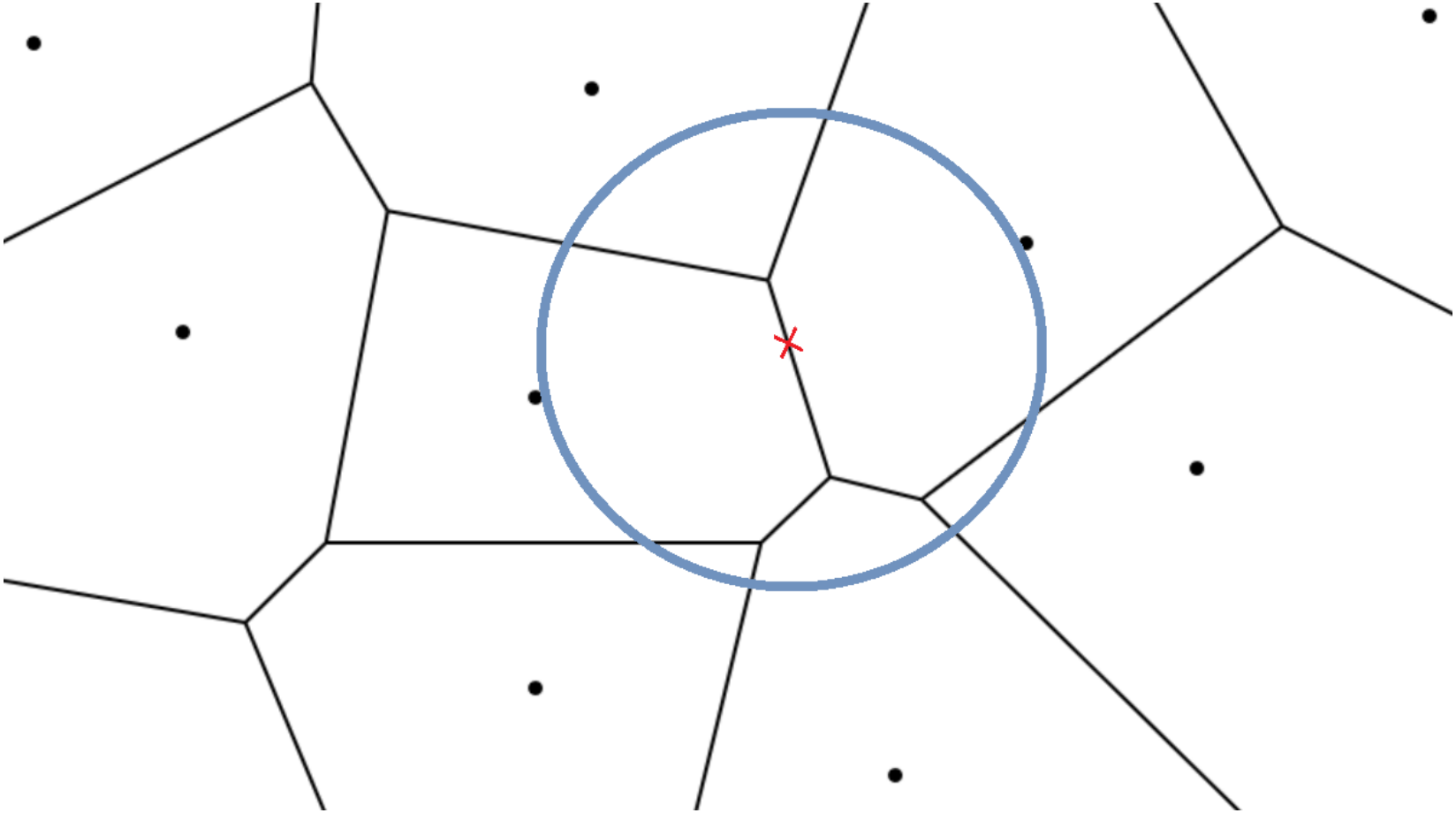


Figure 7

Annexes

Repérage d’un nouveau point



Annexes

Méthode de Shamos & Hoey

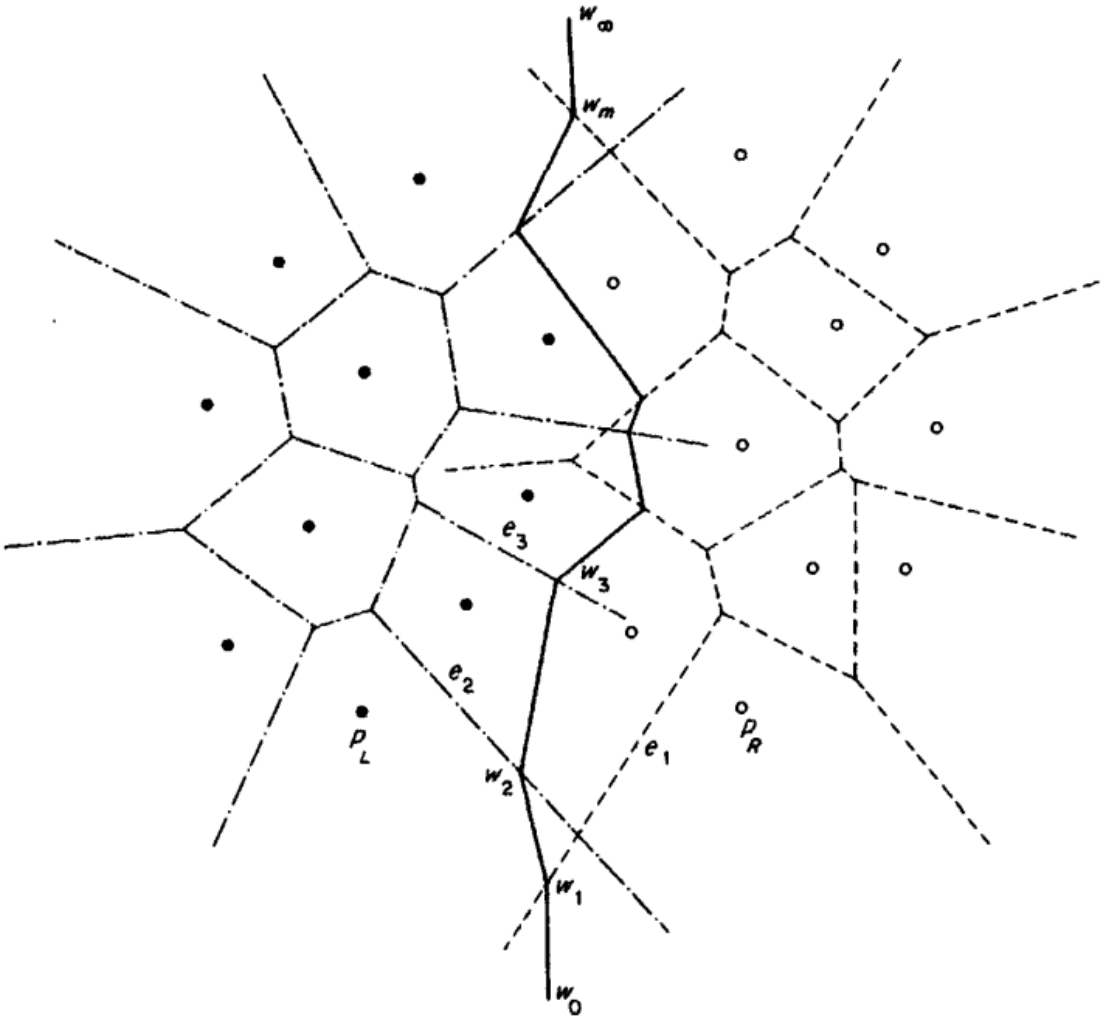


Figure 8

Annexes

Utilisation du diagramme sur des pompes à eau

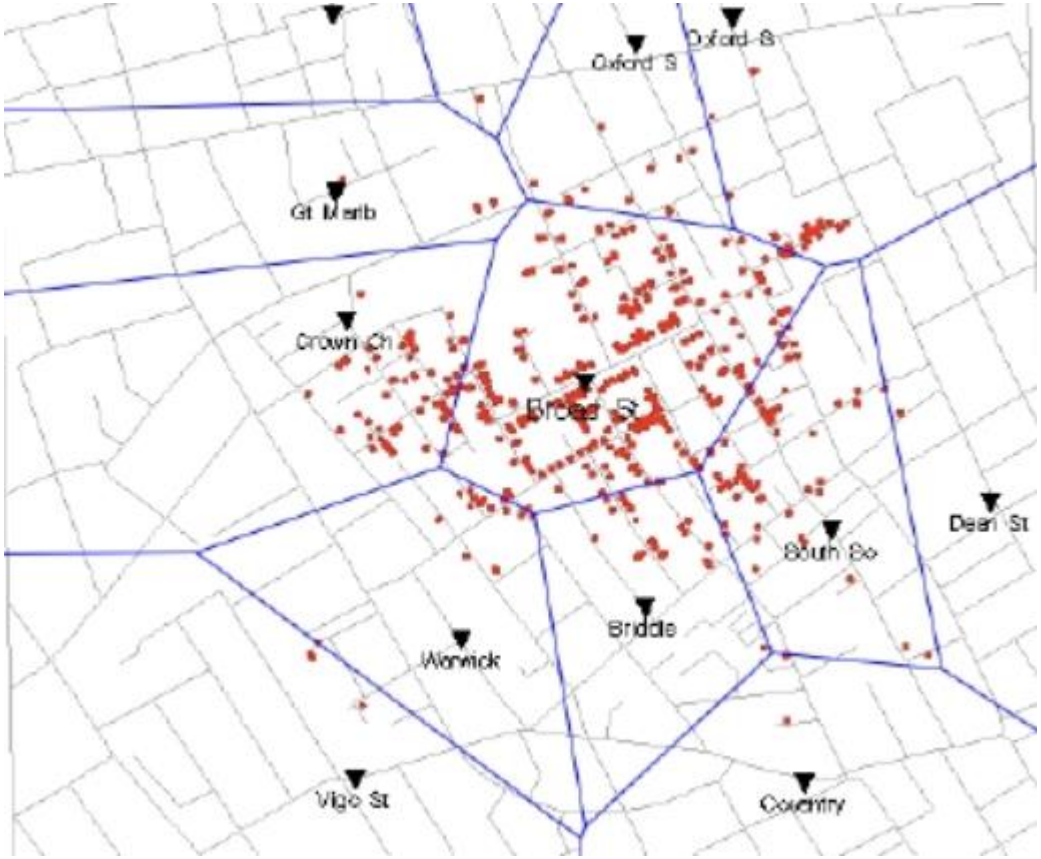


Figure 9

Références

- Figure 1,2,3,4,5,7,8 : Spatial Tesselations Applications of Voronoi Diagrams

- Figure 6 : https://www.ac-lyon.fr/l-enseignement-superieur-dans-la-region-academique-123616

- Figure 9 : https://www.researchgate.net/figure/A-Voronoi-tessellation-of-Snows-map-of-the-location-of-thirteen-pumps-that-served-as_fig1_241835844

Code
Paramètres

```
type point = float * float
type arete = point * point
type voronoi = {
    mutable na : int;
    mutable np : int;
    mutable indice_a : (arete,int) Hashtbl.t;
    mutable indice_p : (point,int) Hashtbl.t;

    mutable polygone_d : point array;
    mutable polygone_g : point array;
    mutable debut_a : point array;
    mutable fin_a : point array;
    mutable h_precedent : arete option array;
    mutable ah_precedent : arete option array;
    mutable h_suivant : arete option array;
    mutable ah_suivant : arete option array;

    mutable arete_autour_p : arete list array;

}

type quadtree = |Nil | Noeud of point * quadtree*quadtree*quadtree*quadtree

let tw,gw = 25,25
let th,gh = 25,25
let p1 = (0.5,3.0*.sqrt(2.0)/.2.0 +. 1.)
let p2 = (-.3.0*.sqrt(6.0)/.4.0 +. 0.5, -.3.0*.sqrt(2.0)/.2.0 +. 1. )
let p3 = (3.0*.sqrt(6.0)/.4.0 +. 0.5, -.3.0*.sqrt(2.0)/.2.0 +. 1. )
```

Code
Mathématiques

```
open Parameters

let milieu x1 x2 =
  ((fst x1 +. fst x2)/.2.,(snd x1 +. snd x2)/.2.)


  let identifiable a b =
    let v = a -. b in
    let va = if v < 0. then -.v else v in
    va < 1e-5

let equation ar =
  let (x1,y1),(x2,y2) = ar in
  if identifiable x1 x2
  then (Float.nan, x1)
  else let a = (y2 -. y1)/.(x2 -. x1) in
       let b = y1 -. a*.x1 in
       (a,b)

let dist p1 p2 =
  int_of_float (10000000.*.Float.sqrt ((fst p1 -. fst p2)*.(fst p1 -. fst p2)
  +. (snd p1 -. snd p2)*.(snd p1 -. snd p2)))
```

Code
Mathématiques

```
let intersection ar1 eq =
  let a1,b1 = equation ar1 in
  let a2,b2 = eq in
  if a1 -. a2 = 0.0
  then None
  else
      match Float.is_nan a1, Float.is_nan a2 with
      | true, true ->
            if identifiable b1 b2
            then Some (b1, 0.)
            else None
        | true, false -> Some (b1, a2 *. b1 +. b2)
        | false, true -> Some (b2, a1 *. b2 +. b1)
        | _ -> let x = (b2 -. b1)/.(a1 -. a2) in
               let y = (a1*.x +.b1) in
                    Some (x,y)

let mediatrice p1 p2 =
    let a,_ = equation (p1,p2) in
    let a' = -.1.0/.a in
    let x,y = milieu p1 p2 in
    let b' = y -. a'*.x in
    (a',b')


let ordre_triangle i j k  =
    let a = ((snd k) -. (snd i))*.((fst j) -. (fst i)) -. ((fst k) -. (fst i))*.((snd j) -. (snd i)) in
    (a>=1.0)
```

Code
Mathématiques

```
let epsilon_comp a1 a2 eps =
  fst a1 -. fst eps <= fst a2 && fst a1 +. fst eps >= fst a2
  && snd a1 -. snd eps <= snd a2 && snd a1 +. snd eps >= snd a2

let est_aigu wh wi wj =
  ((fst(fst wh)) -.(fst (fst wi)))*.((fst(fst wj)) -. (fst(fst wi)))
  +. ((snd(fst wh)) -.(snd (fst wi)))*.((snd(fst wj)) -. (snd(fst wi))) > 0.
```

Code
Données

```
open Parameters
open Math

let voronoi_copy v =
  {
    na = v.na;
    np = v.np;
    indice_a = Hashtbl.copy v.indice_a;
    indice_p = Hashtbl.copy v.indice_p;
    polygone_d = Array.copy v.polygone_d;
    polygone_g = Array.copy v.polygone_g;
    debut_a = Array.copy v.debut_a;
    fin_a = Array.copy v.fin_a;
    h_precedent = Array.copy v.h_precedent;
    ah_precedent = Array.copy v.ah_precedent;
    h_suivant = Array.copy v.h_suivant;
    ah_suivant = Array.copy v.ah_suivant;

    arete_autour_p = Array.copy v.arete_autour_p;
  }
```

```
let trouve_poly v s =
  let plus_proche_p = ref v.polygone_g.(0) in
  let dist_min = ref (dist s !plus_proche_p) in
  if dist s v.polygone_d.(0) < !dist_min
  then begin
    dist_min := dist s v.polygone_d.(0);
    plus_proche_p := v.polygone_d.(0)
  end;
  for i = 1 to v.na -1 do
    let a = dist s v.polygone_g.(i) in
    if a < !dist_min
    then begin
      dist_min := a; plus_proche_p := v.polygone_g.(i)
    end;
    let b = dist s v.polygone_d.(i) in
    if b < !dist_min
    then begin
      dist_min := b; plus_proche_p := v.polygone_d.(i)
    end
  done;
  (!plus_proche_p, (Hashtbl.find v.indice_p !plus_proche_p))
```

Code
Données

```
let ajoute_point_struct v p a_l =
  let n_p = v.np + 1 in
  Hashtbl.add v.indice_p p (n_p -1);
  let t = ref [||] in
  let n = Array.length v.arete_autour_p in
  if n_p < n
  then t := Array.make n []
  else t := Array.make (2*n) [];
  for i = 0 to (n_p-2) do
      !t.(i) <- v.arete_autour_p.(i)
  done;
  !t.(n_p - 1) <- a_l;
  v.np <- n_p;
  v.arete_autour_p <- !t
```

Code
Données

```
let ajoute_arete_struct v a g d =
  let n_a = v.na +1 in
  Printf.printf "indice : %d\n" (n_a-1);flush stdout;
  Hashtbl.add v.indice_a a (n_a -1);
  let polygone_d_r = ref [||] in
  let polygone_g_r = ref [||] in
  let debut_a_r = ref [||] in
  let fin_a_r = ref [||] in
  let h_precedent_r = ref [||] in
  let ah_precedent_r = ref [||] in
  let h_suivant_r = ref [||]  in
  let ah_suivant_r = ref [||] in
  if Array.length v.polygone_d < n_a
  then begin
        let polygone_d = Array.make (2*n_a) (0.0,0.0) in
        let polygone_g = Array.make (2*n_a) (0.0,0.0) in
        let debut_a = Array.make (2*n_a) (0.0,0.0) in
        let fin_a = Array.make (2*n_a) (0.0,0.0) in
        let h_precedent = Array.make (2*n_a) (Some ((0.0,0.0),(0.0,0.0))) in
        let ah_precedent = Array.make (2*n_a) (Some ((0.0,0.0),(0.0,0.0))) in
        let h_suivant = Array.make (2*n_a) (Some ((0.0,0.0),(0.0,0.0))) in
        let ah_suivant = Array.make (2*n_a) (Some ((0.0,0.0),(0.0,0.0))) in
        for i = 0 to (n_a -2 ) do
          polygone_d.(i) <- v.polygone_d.(i);
          polygone_g.(i) <- v.polygone_g.(i);
          debut_a.(i) <- v.debut_a.(i);
          fin_a.(i) <- v.fin_a.(i);
          h_precedent.(i) <- v.h_precedent.(i);
          ah_precedent.(i) <- v.ah_precedent.(i);
          h_suivant.(i) <- v.h_suivant.(i);
          ah_suivant.(i)<- v.ah_suivant.(i);
    done ;
```

Code
Données

```
    polygone_d_r := polygone_d;
    polygone_g_r := polygone_g;
    debut_a_r := debut_a;
    fin_a_r := fin_a;
    h_precedent_r := h_precedent;
    ah_precedent_r := ah_precedent;
    h_suivant_r := h_suivant;
    ah_suivant_r := ah_suivant;
    end
else begin
    polygone_d_r := v.polygone_d;
    polygone_g_r := v.polygone_g;
    debut_a_r := v.debut_a;
    fin_a_r := v.fin_a;
    h_precedent_r := v.h_precedent;
    ah_precedent_r := v.ah_precedent;
    h_suivant_r := v.h_suivant;
    ah_suivant_r := v.ah_suivant;
    end;
  !polygone_d_r.(n_a -1) <- d;
  !polygone_g_r.(n_a -1) <- g;
  !debut_a_r.(n_a - 1) <- fst a;
  !fin_a_r.(n_a -1) <- snd a;
```

Code
Données

```
for i = 0 to n_a - 2 do
  if !fin_a_r.(i) = fst a then begin

    if ordre_triangle (fst a) (snd a) (!debut_a_r.(i)) then begin

      !ah_suivant_r.(i) <- Some a;
      !ah_precedent_r.(n_a-1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
      !h_suivant_r.(n_a -1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
      !h_precedent_r.(i) <- Some a
    end

    else begin

      !ah_suivant_r.(n_a -1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
      !ah_precedent_r.(i) <- Some a;
      !h_suivant_r.(n_a -1) <- Some a;
      !h_precedent_r.(i) <- Some (!debut_a_r.(i),!fin_a_r.(i))
    end
    end
```

Code
Données

```
else
  if !debut_a_r.(i) = fst a then begin

    if ordre_triangle (fst a) (snd a) (!fin_a_r.(i)) then begin

      !ah_suivant_r.(i) <- Some a;
      !ah_precedent_r.(n_a-1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
      !h_suivant_r.(n_a -1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
      !h_precedent_r.(i) <- Some a
    end

    else begin

      !ah_suivant_r.(n_a -1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
      !ah_precedent_r.(i) <- Some a;
      !h_suivant_r.(n_a -1) <- Some a;
      !h_precedent_r.(i) <- Some (!debut_a_r.(i),!fin_a_r.(i))
    end
    end
```

Code
Données

```
else
  if !fin_a_r.(i) = snd a then begin

    if ordre_triangle (fst a) (snd a) (!debut_a_r.(i)) then begin

      !ah_suivant_r.(n_a -1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
      !ah_precedent_r.(i) <- Some a;
      !h_suivant_r.(n_a -1) <- Some a;
      !h_precedent_r.(i) <- Some (!debut_a_r.(i),!fin_a_r.(i))
    end

    else begin

      !ah_suivant_r.(i) <- Some a;
      !ah_precedent_r.(n_a-1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
      !h_suivant_r.(n_a -1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
      !h_precedent_r.(i) <- Some a
    end
    end
```

Code
Données

```
  else
    if !debut_a_r.(i) = snd a then begin

      if ordre_triangle (fst a) (snd a) (!fin_a_r.(i)) then begin

        !ah_suivant_r.(n_a -1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
        !ah_precedent_r.(i) <- Some a;
        !h_suivant_r.(n_a -1) <- Some a;
        !h_precedent_r.(i) <- Some (!debut_a_r.(i),!fin_a_r.(i))
      end

      else begin

        !ah_suivant_r.(i) <- Some a;
        !ah_precedent_r.(n_a-1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
        !h_suivant_r.(n_a -1) <- Some (!debut_a_r.(i),!fin_a_r.(i));
        !h_precedent_r.(i) <- Some a
      end
      end
done;
```

Code
Données

```
v.na <- n_a;
v.polygone_d <- !polygone_d_r;
v.polygone_g <- !polygone_g_r;
v.debut_a <- !debut_a_r;
v.fin_a <- !fin_a_r;
v.h_precedent <- !h_precedent_r;
v.ah_precedent <- !ah_precedent_r;
v.h_suivant <- !h_suivant_r;
v.ah_suivant <- !ah_suivant_r;
v.arete_autour_p <- v.arete_autour_p
```

Code
Données

```
let erase v a =
  match a with
  |None -> ()
  |Some x ->
  let epsilon = (0.0001,0.0001) in
  if not ((epsilon_comp (fst x) (0.,0.) epsilon) && (epsilon_comp (snd x) (0.,0.) epsilon))
  then begin
  try
  let temp = Hashtbl.find v.indice_a x in
  v.debut_a.(temp) <- (1000.,1000.);
  v.fin_a.(temp) <- (1000.,1000.)
  with Not_found -> (
  let temp = Hashtbl.find v.indice_a ((snd x),(fst x)) in
  v.debut_a.(temp) <- (1000.,1000.);
  v.fin_a.(temp) <- (1000.,1000.))
  end
```

Code
Données

```
let coupe_bas wj a_couperj newj =
  Printf.printf "bas\n"; flush stdout;
  if snd (fst (snd wj)) > snd (snd (snd wj))
    then begin
      a_couperj := (snd (snd wj),fst wj);
      newj := (fst wj,(fst (snd wj)))
    end
    else begin
      newj := (fst wj,snd (snd wj));
      a_couperj := (fst wj,(fst (snd wj)))
    end
```

```
let coupe_haut wj a_couperj newj =
  Printf.printf "haut\n"; flush stdout;
  if snd (fst (snd wj)) > snd (snd (snd wj))
    then begin
      a_couperj := (fst (snd wj),fst wj);
      newj := ((snd (snd wj)),fst wj)
    end
    else begin
      newj := (fst (snd wj),fst wj);
      a_couperj := (fst wj,(snd (snd wj)))
    end
```

```
let coupe_droite wj a_couperj newj =
  Printf.printf "droite\n"; flush stdout;
  if fst (fst (snd wj)) > fst (snd(snd wj))
  then begin
    a_couperj := (fst (snd wj),fst wj);
    newj := ((snd (snd wj)),fst wj)
  end
  else begin
    newj := (fst (snd wj),fst wj);
    a_couperj := (fst wj,(snd (snd wj)))
  end
```

```
let coupe_gauche wj a_couperj newj =
  Printf.printf "gauche\n"; flush stdout;
  if fst (fst (snd wj)) > fst (snd(snd wj))
  then begin
    a_couperj := (snd (snd wj),fst wj);
    newj := (fst wj,(fst (snd wj)))
  end
  else begin
    newj := (fst wj,snd (snd wj));
    a_couperj := (fst wj,(fst (snd wj)))
  end
```

Code
Données

```
let efface_arete v wh wi wj a p =
  let epsilon = (0.001,0.001) in
  let a_couperj = ref ((0.,0.),(0.,0.)) in
  let newj = ref (snd wi) in
  if epsilon_comp (0.,snd (fst wi)) (0.,snd (fst wj)) epsilon && fst (fst wi) > fst(fst wj)
  then begin
    if fst (fst wh) > fst(fst wi)
    then coupe_bas wi a_couperj newj
    else coupe_haut wi a_couperj newj
  end;
  let x,y = (pente (fst wh) (fst wi)),(pente (fst wi) (fst wj)) in
  if Float.is_nan(pente (fst (snd wi)) (snd (snd wi)) )
    && (fst(fst wh) < fst(fst wi) && fst(fst wj) > fst(fst wi)
  || fst(fst wh) > fst(fst wi) && fst(fst wj) < fst(fst wi)) then
  tronque wh wi wj a_couperj newj
  else
  if x > 0. && y < 0. || x < 0. && y > 0.
  then begin if (snd(fst wh) < snd(fst wi) && snd(fst wj) < snd(fst wi)
    || snd(fst wh) > snd(fst wi) && snd(fst wj) > snd(fst wi))
  then begin if((Float.min x y) > (pente (fst (snd wi)) (snd (snd wi)))
    || (Float.max x y) < (pente (fst (snd wi)) (snd(snd wi))))
  then begin
    tronque wh wi wj a_couperj newj;
    Printf.printf "teeeeest2\n";flush stdout
  end
  end
```

Code
Données

```
else if ((Float.min x y) < (pente (fst (snd wi)) (snd (snd wi)))
  && (Float.max x y) > (pente (fst (snd wi)) (snd(snd wi))))
  then tronque wh wi wj a_couperj newj
end
else if x > 0. && y > 0. || x < 0. && y < 0. then begin
  if not(est_aigu wh wi wj)
  then begin
  if ((Float.min x y) > (pente (fst (snd wi)) (snd (snd wi)))
    || (Float.max x y) < (pente (fst (snd wi)) (snd(snd wi))))
     then begin
     tronque wh wi wj a_couperj newj;
     Printf.printf "teeeeest4\n";flush stdout
     end
  end
  else if ((Float.min x y) < (pente (fst (snd wi)) (snd (snd wi)))
    && (Float.max x y) > (pente (fst (snd wi)) (snd(snd wi))))
    then begin
      tronque wh wi wj a_couperj newj;
      Printf.printf "teeeeest4\n";flush stdout
      end

  end;
if x > 0. && y < 0. || x < 0. && y > 0.
then if (fst(fst wh) < fst(fst wi) && fst(fst wj) < fst(fst wi)
  || fst(fst wh) > fst(fst wi) && fst(fst wj) > fst(fst wi)) then
  tronque wh wi wj a_couperj newj;
```

Code
Données

```
v.arete_autour_p.(Hashtbl.find v.indice_p a) <-
List.filter (fun i -> i <> snd wi) v.arete_autour_p.(Hashtbl.find v.indice_p a);
let flag = ref false in
for i = 0 to v.na - 1 do
  if v.ah_precedent.(i) = Some (snd wi)
  then begin
    v.ah_precedent.(i) <- None;
  end;
  if v.h_precedent.(i) = Some (snd wi)
  then begin
    flag := true;
    if not( epsilon_comp v.debut_a.(i) v.debut_a.(Hashtbl.find v.indice_a (snd wj)) epsilon
    && epsilon_comp v.fin_a.(i) v.fin_a.(Hashtbl.find v.indice_a (snd wj)) epsilon)
    then erase v (Some (v.debut_a.(i),v.fin_a.(i)))
    else flag := false
  end;
  if v.h_precedent.(i) = Some (snd wi)
  then begin
    v.h_precedent.(i) <- None;
  end;
  if v.ah_suivant.(i) = Some (snd wi)
  then  begin
    v.ah_suivant.(i) <- None;
  end;
  if v.h_suivant.(i) = Some (snd wj)
  then begin
    flag := false
  end;
  if !flag
  then erase v v.h_suivant.(i)
done;
```

Code
Données

```
  erase v (Some (snd wi));
  (try
  ajoute_arete_struct v !newj
  v.polygone_g.(Hashtbl.find v.indice_a (snd wi)) v.polygone_d.(Hashtbl.find v.indice_a (snd wi));
  with Not_found -> ajoute_arete_struct v !newj
  v.polygone_g.(Hashtbl.find v.indice_a ((snd (snd wi)),fst(snd wi)))
  v.polygone_d.(Hashtbl.find v.indice_a ((snd (snd wi)),fst(snd wi))););
  v.arete_autour_p.(Hashtbl.find v.indice_p a) <-
  !newj::v.arete_autour_p.(Hashtbl.find v.indice_p a)



let affichewi wi : unit =
  Printf.printf "\t((%f,%f),  ((%f,%f), (%f,%f)))\n" (fst (fst wi)) (snd (fst wi))
  (fst (fst (snd wi))) (snd (fst (snd wi))) (fst (snd (snd wi))) (snd (snd (snd wi)));
  flush stdout;
  ()

let rec affichelist l =
  match l with
  | [] -> Printf.printf "\n"
  | t::q ->
      Printf.printf "((%f,%f), (%f,%f))\n" (fst (fst t)) (snd (fst t)) (fst (snd t)) (snd (snd t));
      flush stdout;
      affichelist q
```

Code
Voronoï

```
open Math
open Donnee
open Parameters

let base_trig () =
    let p = Hashtbl.create 3 in
    Hashtbl.add p p1 0;
    Hashtbl.add p p2 1;
    Hashtbl.add p p3 2;
    let s = Hashtbl.create 4 in
    Hashtbl.add s (0.0,0.0) 0;
    Hashtbl.add s (milieu p1 p2)  1;
    Hashtbl.add s (milieu p1 p3)  2;
    Hashtbl.add s (milieu p2 p3)  3;
    let a = Hashtbl.create 3 in

    let n = 10. in

    let a1',b1 = mediatrice p1 p2 in
    let a2',b2 = mediatrice p1 p3 in

    let a1 = ((0.5,a1'*.0.5+.b1),(-.10.+.0.5,a1'*.(-.10.+.0.5)+.b1)) in
    let a2 = ((0.5,a2'*.0.5+.b2),(10.+.0.5,a2'*.(10.+.0.5)+.b2)) in
    let a3 = ((0.5,a2'*.0.5+.b2), (n*.(fst (milieu p2 p3) -. 0.5)
    +. 0.5, n*.(snd (milieu p2 p3) -. 0.5) +. 0.5)) in

    Hashtbl.add a a1 0;
    Hashtbl.add a a2 1;
    Hashtbl.add a a3 2;
```

Code
Voronoï

```
{
na = 3;
np = 3;
indice_a = a;
indice_p = p;


polygone_d = [|p1;p3;p2|];
polygone_g = [|p2;p1;p3|];
debut_a = [|fst a1;fst a2;fst a3|];
fin_a =[|snd a1;snd a2;snd a3|];
h_precedent = [|None;Some a3;Some a1|];
ah_precedent = [|Some a3;None; Some a2|];
h_suivant = [|Some a3;None;None|];
ah_suivant = [|None;Some a3;None|];

arete_autour_p = [|[a1;a2];[a1;a3];[a2;a3]|];
}
```

Code
Voronoï

```
let prochain_point_candidat la med mil  =
    let wi = ref ((200.,200.),((4000.,4000.),(3200.,3200.))) in
    let wj = ref ((200.,200.),((4000.,4000.),(3200.,3200.))) in
    let epsilon = (0.001,0.001) in
    List.iter(fun i -> match (intersection i med) with
                        |None -> ()
                        |Some temp ->
                       if (fst temp) <= fst mil &&  dist temp mil
                        < dist (fst !wi) mil && not(Float.is_nan (fst temp))
                       && not(Float.is_nan (snd temp)
                       then wi := (temp,i)
                       else if dist temp mil < dist (fst !wj) mil
                        && not(epsilon_comp temp (fst !wi) epsilon)
                        && not(Float.is_nan (fst temp))
                        && not(Float.is_nan (snd temp))
                            then wj := (temp,i)
            ) la;
    (wi,wj)
```

Code
Voronoï

```
let ajoute_point v p =
    let p',_= trouve_poly v p in
    let la = v.arete_autour_p.(Hashtbl.find v.indice_p p')  in
    let a_l = ref [] in
    let med = mediatrice p p' in
    let mil = milieu p p' in
    let w1,w2 = prochain_point_candidat la med mil in
    let p_actu = ref !w2 in
    let goal = ref !w1 in
    let deux = ref !w2 in
    let sens = ref false in
    let p_prec = ref p' in
    let v' = voronoi_copy v in
    let wh = ref !w1 in
    if snd p' > snd p
    then begin
        p_actu :=!w1;
        deux := !w1;
        goal := !w2;
        wh := !w2;
        sens := true;
        ajoute_arete_struct v' (fst !w2,fst !w1) p p';
        a_l := (fst !w2,fst !w1)::!a_l;
        v'.arete_autour_p.(Hashtbl.find v.indice_p p') <-
        (fst !w1, fst !w2)::(v'.arete_autour_p.(Hashtbl.find v.indice_p p'))
        end
```

Code
Voronoï

```
    else begin
        ajoute_arete_struct v' (fst !w1,fst !w2) p p';
        v'.arete_autour_p.(Hashtbl.find v.indice_p p') <-
        (fst !w2, fst !w1)::(v'.arete_autour_p.(Hashtbl.find v.indice_p p'));
        a_l := (fst !w1,fst !w2)::!a_l
    end;
    let deux' = ref !w1 in
    let a = ref p' in
    let epsilon = (0.001,0.001) in
    while not (epsilon_comp (fst !p_actu) (fst !goal) epsilon)  do
        if !wh = !deux
        then deux' := !p_actu;
        let temp = !a in
        a:= v.polygone_d.(Hashtbl.find v.indice_a (snd !p_actu));
        if epsilon_comp temp !a epsilon then
        a := v.polygone_g.(Hashtbl.find v.indice_a (snd !p_actu));
        p_prec := temp;
        let med' = mediatrice p !a in
        let x,y = !a in
        Printf.printf "a : %f,%f\n" x y;flush stdout;
        let la' = v.arete_autour_p.(Hashtbl.find v.indice_p !a) in
        let mil' = milieu p !a in
        let (wi,wj)= prochain_point_candidat la' med' mil' in
```

Code
Voronoï

```
if (fst (fst !wi) < fst ( fst !p_actu) && fst (fst !wj) < fst ( fst !p_actu))
    || (fst (fst !wi) > fst ( fst !p_actu) && fst (fst !wj) > fst ( fst !p_actu))
then begin
    if dist (fst !wj) (fst !p_actu) < dist (fst !wi) (fst !p_actu)
    then begin
        let temp = !wj in
        wj := !wi;
        wi := temp
    end;
    end;
if epsilon_comp (fst !wi) (200.,200.) epsilon
then wi := !wj;
if epsilon_comp (fst !wj) (200.,200.) epsilon
then wj := !wi;
if ordre_triangle p (fst !p_actu) (fst !wi)  || epsilon_comp (fst !goal) (fst !wi) epsilon
then begin
    affichewi !wi;
    efface_arete v' !wh !p_actu !wi temp p;
    ajoute_arete_struct v' (fst !p_actu,fst !wi) p !a;
    v'.ah_suivant.(Hashtbl.find v'.indice_a (fst !p_actu,fst !wi )) <- Some (fst !wh,fst !p_actu);
    v'.h_suivant.(Hashtbl.find v'.indice_a (fst !wh,fst !p_actu )) <- Some (fst !p_actu,fst !wi );
    wh:= !p_actu ;
    a_l:=(fst !p_actu, fst !wi)::!a_l;
    v'.arete_autour_p.(Hashtbl.find v'.indice_p !a) <-
    (fst !p_actu, fst !wi)::(v'.arete_autour_p.(Hashtbl.find v'.indice_p !a));
    p_actu := !wi;
    end
```

Code
Voronoï

```
    else
        begin
        affichewi !wj;
        efface_arete v' !wh !p_actu !wj temp p;
        ajoute_arete_struct v' (fst !p_actu,fst !wj) !a p;
        v'.ah_suivant.(Hashtbl.find v'.indice_a (fst !p_actu,fst !wj )) <- Some (fst !wh,fst !p_actu);
        v'.h_suivant.(Hashtbl.find v'.indice_a (fst !wh,fst !p_actu )) <- Some (fst !p_actu,fst !wj );
        wh := !p_actu;
        let test' = fst !p_actu in
        a_l:=(fst !p_actu,fst !wj)::!a_l;
        v'.arete_autour_p.(Hashtbl.find v'.indice_p !a) <-
        (fst !p_actu, fst !wj)::(v'.arete_autour_p.(Hashtbl.find v'.indice_p !a));
        p_actu := !wj;
        end

done;
efface_arete v' !wh !p_actu !deux' !a p;
ajoute_point_struct v' p !a_l;
v.na <- v'.na;
v.np <- v'.np;
v.indice_a <- v'.indice_a;
v.indice_p <- v'.indice_p;
v.polygone_d <- v'.polygone_d;
v.polygone_g <- v'.polygone_g;
v.debut_a <- v'.debut_a;
v.fin_a <- v'.fin_a;
v.h_precedent <- v'.h_precedent;
v.h_suivant <- v'.h_suivant;
v.ah_precedent <- v'.ah_precedent;
v.ah_suivant <- v'.ah_suivant;
v.arete_autour_p <- v'.arete_autour_p
```

Code
Optimisations

```
open Parameters
open Math
open Donnee
open Voronoi

let rec near_neigh v p pi =
  let l = ref [] in
  List.iter(fun i -> let a = v.polygone_d.(Hashtbl.find v.indice_a i) in
                     if a <> pi then l := a::!l
                      else l := v.polygone_d.(Hashtbl.find v.indice_a i)::!l )
                      v.arete_autour_p.(Hashtbl.find v.indice_p pi);
  let d = ref (dist p pi) in
  let p_near = ref pi in
  List.iter (fun i -> if dist p i < !d then begin d:= dist p i; p_near := i end) !l;
  if !p_near = pi
  then pi
  else near_neigh v p !p_near
```

Code
Optimisations

```
let ajoute_point_2 v p pi =
  let p' = near_neigh v p pi in
  let la = v.arete_autour_p.(Hashtbl.find v.indice_p p')  in
  let a_l = ref [] in
  let med = mediatrice p p' in
  let mil = milieu p p' in
  let w1,w2 = prochain_point_candidat la med mil in
  let p_actu = ref !w2 in
  let goal = ref !w1 in
  let deux = ref !w2 in
  let sens = ref false in
  let p_prec = ref p' in
  let v' = voronoi_copy v in
  let wh = ref !w1 in
  if snd p' > snd p
  then begin
        p_actu :=!w1;
        deux := !w1;
        goal := !w2;
        wh := !w2;
        sens := true;
        ajoute_arete_struct v' (fst !w2,fst !w1) p p';
        a_l := (fst !w2,fst !w1)::!a_l;
        v'.arete_autour_p.(Hashtbl.find v.indice_p p') <-
        (fst !w1, fst !w2)::(v'.arete_autour_p.(Hashtbl.find v.indice_p p'))
        end
  else begin
        ajoute_arete_struct v' (fst !w1,fst !w2) p p';
        v'.arete_autour_p.(Hashtbl.find v.indice_p p') <-
        (fst !w2, fst !w1)::(v'.arete_autour_p.(Hashtbl.find v.indice_p p'));
        a_l := (fst !w1,fst !w2)::!a_l
```

Code
Optimisations

```
let deux' = ref !w1 in
let a = ref p' in
let epsilon = (0.001,0.001) in
while not (epsilon_comp (fst !p_actu) (fst !goal) epsilon)  do
    if !wh = !deux
    then deux' := !p_actu;
    let temp = !a in
    a:= v.polygone_d.(Hashtbl.find v.indice_a (snd !p_actu));
    if epsilon_comp temp !a epsilon then
    a := v.polygone_g.(Hashtbl.find v.indice_a (snd !p_actu));
    p_prec := temp;
    let med' = mediatrice p !a in
    let x,y = !a in
    Printf.printf "a : %f,%f\n" x y;flush stdout;
    let la' = v.arete_autour_p.(Hashtbl.find v.indice_p !a) in
    let mil' = milieu p !a in
    let (wi,wj)= prochain_point_candidat la' med' mil' in
    if (fst (fst !wi) < fst ( fst !p_actu) && fst (fst !wj) < fst ( fst !p_actu))
        || (fst (fst !wi) > fst ( fst !p_actu) && fst (fst !wj) > fst ( fst !p_actu))
    then begin
          if dist (fst !wj) (fst !p_actu) < dist (fst !wi) (fst !p_actu)
          then begin
              let temp = !wj in
              wj := !wi;
              wi := temp
          end;
        end;
```

Code
Optimisations

```
if epsilon_comp (fst !wi) (200.,200.) epsilon
then wi := !wj;
if epsilon_comp (fst !wj) (200.,200.) epsilon
then wj := !wi;
if ordre_triangle p (fst !p_actu) (fst !wi)  || epsilon_comp (fst !goal) (fst !wi) epsilon
then begin
    affichewi !wi;
    efface_arete v' !wh !p_actu !wi temp p;
    ajoute_arete_struct v' (fst !p_actu,fst !wi) p !a;
    v'.ah_suivant.(Hashtbl.find v'.indice_a (fst !p_actu,fst !wi )) <- Some (fst !wh,fst !p_actu);
    v'.h_suivant.(Hashtbl.find v'.indice_a (fst !wh,fst !p_actu )) <- Some (fst !p_actu,fst !wi );
    wh:= !p_actu ;
    a_l:=(fst !p_actu, fst !wi)::!a_l;
    v'.arete_autour_p.(Hashtbl.find v'.indice_p !a) <-
    (fst !p_actu, fst !wi)::(v'.arete_autour_p.(Hashtbl.find v'.indice_p !a));
    p_actu := !wi;
    end
else
    begin
    affichewi !wj;
    efface_arete v' !wh !p_actu !wj temp p;
    ajoute_arete_struct v' (fst !p_actu,fst !wj) !a p;
    v'.ah_suivant.(Hashtbl.find v'.indice_a (fst !p_actu,fst !wj )) <- Some (fst !wh,fst !p_actu);
    v'.h_suivant.(Hashtbl.find v'.indice_a (fst !wh,fst !p_actu )) <- Some (fst !p_actu,fst !wj );
    wh := !p_actu;
    let test' = fst !p_actu in
    a_l:=(fst !p_actu,fst !wj)::!a_l;
    v'.arete_autour_p.(Hashtbl.find v'.indice_p !a) <-
    (fst !p_actu, fst !wj)::(v'.arete_autour_p.(Hashtbl.find v'.indice_p !a));
    p_actu := !wj;
    end
```

Code
Optimisations

```
  efface_arete v' !wh !p_actu !deux' !a p;
  ajoute_point_struct v' p !a_l;
  v.na <- v'.na;
  v.np <- v'.np;
  v.indice_a <- v'.indice_a;
  v.indice_p <- v'.indice_p;
  v.polygone_d <- v'.polygone_d;
  v.polygone_g <- v'.polygone_g;
  v.debut_a <- v'.debut_a;
  v.fin_a <- v'.fin_a;
  v.h_precedent <- v'.h_precedent;
  v.h_suivant <- v'.h_suivant;
  v.ah_precedent <- v'.ah_precedent;
  v.ah_suivant <- v'.ah_suivant;
  v.arete_autour_p <- v'.arete_autour_p

let rec search_pow_4 powk n k  =
  if abs(powk - n) < abs(4*powk - n)
  then powk
  else search_pow_4 (4*powk) n (k+1)

let cherche_haut_gauche l h =
  let p = ref (List.hd l) in
  List.iter (fun i -> if (fst i) < (fst !p ) && Float.abs((fst !p) -. (fst i)) > h then p := i;
                      if (snd i) > (snd !p ) && Float.abs((snd i) -. (snd !p)) > h then p := i) l;
  !p
```

Code
Optimisations

```
let rec create_tree v_p pow_4 org =
  let h = 1./.float_of_int(pow_4) in
  if v_p <> []
  then begin
    let a = cherche_haut_gauche v_p h in
    let h_g = List.filter(fun i -> (fst i) < ((fst org) +. h*.float_of_int(pow_4/4))
    && (snd i) >  ((snd org) +. h*.float_of_int(pow_4/4))) v_p in
    let h_d = List.filter(fun i -> (fst i) > ((fst org) +. h*.float_of_int(pow_4/4))
    && (snd i) >  ((snd org) +. h*.float_of_int(pow_4/4))) v_p in
    let b_g = List.filter(fun i -> (fst i) < ((fst org) +. h*.float_of_int(pow_4/4))
    && (snd i) <  ((snd org) +. h*.float_of_int(pow_4/4))) v_p in
    let b_d = List.filter(fun i -> (fst i) > ((fst org) +. h*.float_of_int(pow_4/4))
    && (snd i) <  ((snd org) +. h*.float_of_int(pow_4/4))) v_p in
    Noeud (a,create_tree h_g (pow_4/4) ((fst org),((snd org) +. h*.float_of_int(pow_4/4))),
    create_tree h_d (pow_4/4) (((fst org) +. h*.float_of_int(pow_4/4)),((snd org) +. h*.float_of_int(pow_4/4)))
    ,create_tree b_g (pow_4/4) org,
    create_tree b_d (pow_4/4) ((fst org) +. h*.float_of_int(pow_4/4),(snd org)))
  end
  else Nil
```

Code
Optimisations

```
let rec parcours v arbre =
  match arbre with
  |Nil -> ()
  |Noeud(p,eg,g,d,ed) -> match eg with
                         |Nil -> ()
                         |Noeud(p1,_,_,_,_) -> (ajoute_point_2 v p1 p; parcours v eg);
                         match g with
                         |Nil -> ()
                         |Noeud(p2,_,_,_,_) -> (ajoute_point_2 v p2 p; parcours v g;);
                         match d with
                         |Nil -> ()
                         |Noeud(p3,_,_,_,_) -> (ajoute_point_2 v p3 p; parcours v d;);
                         match ed with
                         |Nil -> ()
                         |Noeud(p4,_,_,_,_) -> (ajoute_point_2 v p4 p;parcours v ed)
```

Code
Main

```
open Graphics
open Voronoi
open Parameters
open Math


let trace_diagramme (n : float ) v =
  let dcl = tw*gw/2 - int_of_float (0.5 *. n) in
  for i = 0 to v.na -1 do
    fill_circle (int_of_float (n*.((fst v.polygone_d.(i)))) + dcl)
    (int_of_float (n*.(snd v.polygone_d.(i)) ) + dcl) 3;
    fill_circle (int_of_float (n*.((fst v.polygone_g.(i)))) + dcl)
    (int_of_float (n*.(snd v.polygone_g.(i))) + dcl) 3
  done;
  for j = 0 to v.na - 1 do
    let epsilon = (0.001,0.001) in
    if not(epsilon_comp v.debut_a.(j) (1000.,1000.) epsilon) then begin
    moveto ((int_of_float (n *. (fst v.debut_a.(j)))) + dcl)
    ((int_of_float (n *. (snd v.debut_a.(j))))+ dcl);
    lineto (int_of_float (n *. (fst v.fin_a.(j)))+ dcl)
    ((int_of_float (n  *. (snd v.fin_a.(j)))) + dcl)
    end
  done
```

Code
Main

```
let main =
  open_graph (Printf.sprintf " %dx%d" (tw*gw) (th*gh));
  set_window_title "TIPE";
    set_color (rgb 0 0 0);
    let n = float_of_int (tw*gw)*.0.04 in
    let v = (base_trig ()) in
    ajoute_point v (0.01,0.94);
    ajoute_point v (0.73,0.34);
    trace_diagramme n v;
    let wait = 1000. in
    let time = Sys.time () in
    while Sys.time () < time +. wait do
      ()
    done;
  close_graph ()
```