

Optimisation de la couverture vidéo-surveillance

Le problème de la galerie d'art

Baudoin Solal - Option informatique
Candidat n° 26655

TIPE 2023

Thème : La ville

Présentation: pages 1 à 28
Annexe: pages 29 à 77

Problème

Étant donné un polygone représentant le plan d'un lieu à surveiller, quel est le nombre minimum de gardes pouvant le surveiller et comment les placer?

Problème

Étant donné un polygone représentant le plan d'un lieu à surveiller, quel est le nombre minimum de gardes pouvant le surveiller et comment les placer?

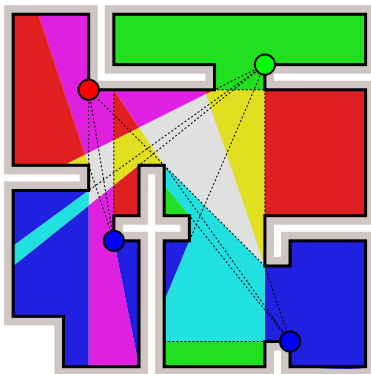
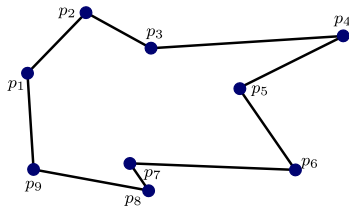


Figure: Exemple de solution où 4 gardes suffisent à garder un polygone

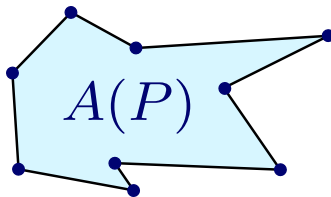
source: https://en.wikipedia.org/wiki/Art_gallery_problem

Notations

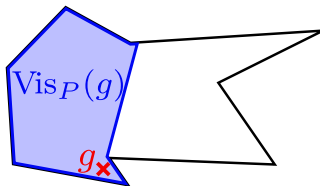


- Polygone: $P = p_1 p_2 \dots p_n$

Notations

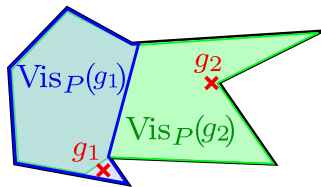


- Polygone: $P = p_1 p_2 \dots p_n$
- Surface d'un polygone P : $A(P)$



- Polygone: $P = p_1 p_2 \dots p_n$
- Surface d'un polygone P : $A(P)$
- Champ de vision: si $g \in A(P)$

$$\text{Vis}_P(g) = \{q \in A(P) \mid [g, q] \subset A(P)\}$$

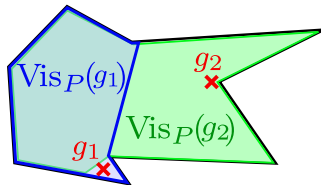


- Polygone: $P = p_1 p_2 \dots p_n$
- Surface d'un polygone P : $A(P)$
- Champ de vision: si $g \in A(P)$

$$Vis_P(g) = \{q \in A(P) \mid [g, q] \subset A(P)\}$$

- Solution: ensemble $S \subset A(P)$ de gardes tel que

$$\bigcup_{g \in S} Vis_P(g) = A(P)$$



- Polygone: $P = p_1 p_2 \dots p_n$
- Surface d'un polygone P : $A(P)$
- Champ de vision: si $g \in A(P)$

$$Vis_P(g) = \{q \in A(P) \mid [g, q] \subset A(P)\}$$

- Solution: ensemble $S \subset A(P)$ de gardes tel que

$$\bigcup_{g \in S} Vis_P(g) = A(P)$$

On cherche des solutions S de cardinalité minimale.

Plan de l'exposé

- ① Une borne supérieure pour les solutions optimales
 - Le théorème de la galerie d'art
 - Point de départ
- ② Décomposition en polygones convexes
 - Formulation en problème de programmation dynamique
 - Problèmes avec l'espace des états naïf
 - Réduction de l'espace des états
 - Schéma algorithmique retenu
- ③ Décomposition en polygones étoilés
 - Formulation en problème de programmation dynamique
 - Problèmes avec l'espace des états naïf
 - Vers un espace des états valide
 - Schéma algorithmique retenu
- ④ Application à la ville de Paris
 - Jeu de données
 - Heuristique supplémentaire
 - Résultats
- ⑤ Références
- ⑥ Annexe

Progression

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

1 Une borne supérieure pour les solutions optimales

Le théorème de la galerie d'art

Point de départ

2 Décomposition en polygones convexes

Formulation en problème de programmation dynamique

Problèmes avec l'espace des états naïf

Réduction de l'espace des états

Schéma algorithmique retenu

3 Décomposition en polygones étoilés

Formulation en problème de programmation dynamique

Problèmes avec l'espace des états naïf

Vers un espace des états valide

Schéma algorithmique retenu

4 Application à la ville de Paris

Jeu de données

Heuristique supplémentaire

Résultats

5 Références

6 Annexe

Le théorème de la galerie d'art

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Théorème de la galerie d'art

dû à Chvátal

Soit P un polygone à n cotés. Il existe une solution S pour le problème de la galerie d'art associé à P composée de $\lfloor \frac{n}{3} \rfloor$ gardes. Cette borne est atteinte.

Le théorème de la galerie d'art

Théorème de la galerie d'art

dû à Chvátal

Soit P un polygone à n cotés. Il existe une solution S pour le problème de la galerie d'art associé à P composée de $\lfloor \frac{n}{3} \rfloor$ gardes. Cette borne est atteinte.

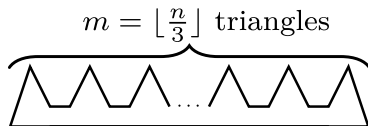


Figure: Exemple où cette borne est atteinte

Le théorème de la galerie d'art

Preuve [Steve Fisk, 1975]

On commence par **triangler** le polygone.

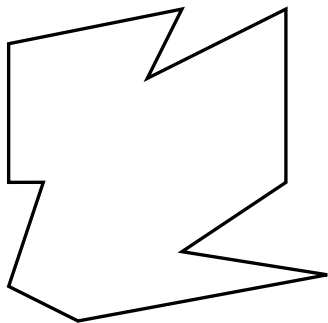


Figure: Un polygone quelconque

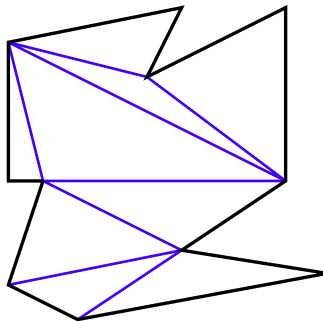


Figure: Polygone triangulé

Le théorème de la galerie d'art

Preuve [Steve Fisk, 1975]

- Le graphe de la triangulation est **3-coloriable**
- Chaque triangle possède **un sommet de chaque couleur**

On choisit donc de placer les gardes sur une des trois couleurs.

On a donc au plus $\lfloor \frac{n}{3} \rfloor$ gardes.

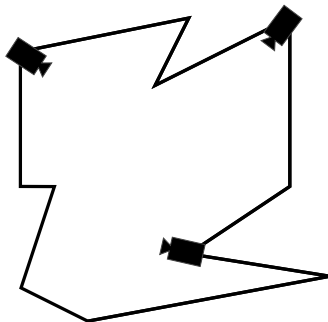
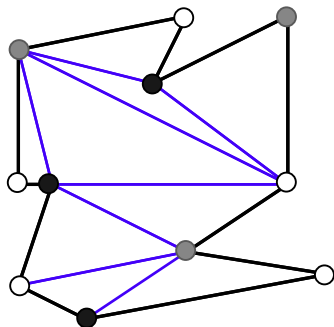


Figure: Un 3-coloriage du polygone

Figure: Placement final des caméras

- Le problème est en général **NP-complet** [Joseph O'Rourke, 1987]

Heuristique

Décomposer le polygone en sous polygones que l'on peut surveiller avec **un seul garde**.

Par exemple:

- En triangles
- En polygones convexes
- En polygones étoilés

Progression

- ① Une borne supérieure pour les solutions optimales
Le théorème de la galerie d'art
Point de départ
- ② Décomposition en polygones convexes
Formulation en problème de programmation dynamique
Problèmes avec l'espace des états naïf
Réduction de l'espace des états
Schéma algorithmique retenu
- ③ Décomposition en polygones étoilés
Formulation en problème de programmation dynamique
Problèmes avec l'espace des états naïf
Vers un espace des états valide
Schéma algorithmique retenu
- ④ Application à la ville de Paris
Jeu de données
Heuristique supplémentaire
Résultats
- ⑤ Références
- ⑥ Annexe

Formulation en problème de programmation dynamique

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Programmation dynamique

- Identifier les **sous-problèmes** dans un problème plus grand et leurs dépendances
- **Ne pas recalculer** des sous problèmes communs
- Solution $\leftrightarrow$ suite de décisions qui font évoluer l'état

$$e_0 \xrightarrow{d_0} e_1 \xrightarrow{d_1} \dots$$

Formulation en problème de programmation dynamique

On considère $P = p_1 p_2 \dots p_n$

Notation On note si $i < j + 1$, $P_{i,j} = p_i p_{i+1} \dots p_j$.

- Les états seront les $(e_{i,j})_{i < j+1 \in \llbracket 1, n \rrbracket^2}$, qui représentent la décision d'avoir décomposé $P_{i,j}$ de façon optimale.
- Une décision sera un couple $(e_{i,m}, e_{m,j})$ avec $i < m < j$ qui représente la formation d'une décomposition minimale de $P_{i,j}$ **avec celles de $P_{i,m}$ et $P_{m,j}$.**
- On a une relation de **préordre** sur les $(P_{i,j})$:

$$P_{i,j} \leq P_{l,k} \Leftrightarrow j - i \leq k - l$$

Formulation en problème de programmation dynamique

Notation

Il existe pour chaque DMC D d'un polygone $P_{i,j}$ un sous polygone noté $C(D)$, l'unique polygone de D contenant la diagonale $[p_i, p_j]$.

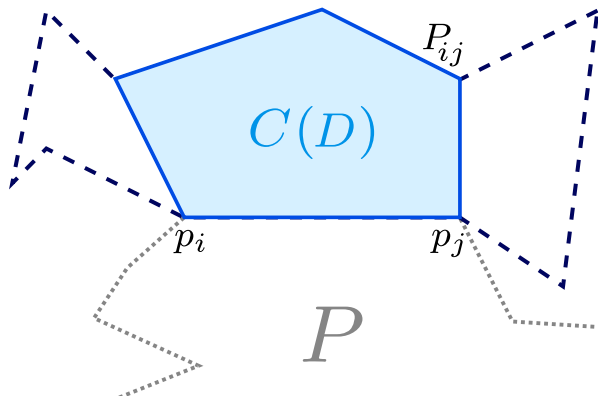


Figure: Polygone convexe de base

Formulation en problème de programmation dynamique

Construire une solution

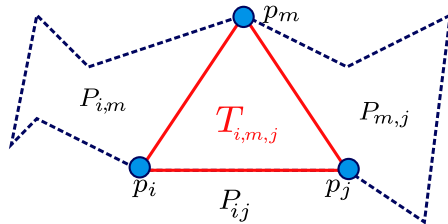
DMC des polygones $P_{l,k} \leq P_{i,j} \rightarrow \text{DMC de } P_{i,j}$

Formulation en problème de programmation dynamique

Construire une solution

DMC des polygones $P_{i,k} \leq P_{i,j} \rightarrow \text{DMC de } P_{i,j}$

- On considère les triangles $(T_{i,m,j})_{i < m < j}$

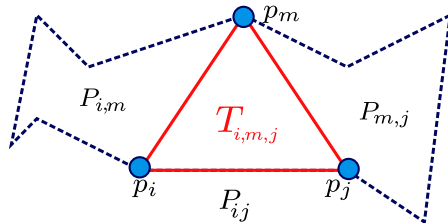


Formulation en problème de programmation dynamique

Construire une solution

DMC des polygones $P_{i,k} \leq P_{i,j} \rightarrow \text{DMC de } P_{i,j}$

- On considère les triangles $(T_{i,m,j})_{i < m < j}$



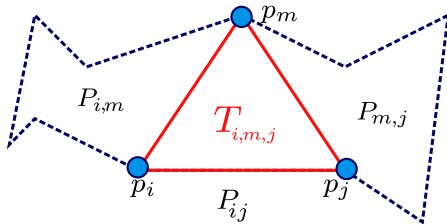
- Combinaison de décompositions $D_{i,m}$ et $D_{m,j}$ de $P_{i,m}$ et $P_{m,j}$ et $T_{i,m,j}$
 - Fusion simple si $C(D_{i,m}) \cup T_{i,m,j}$ ou $C(D_{m,j}) \cup T_{i,m,j}$ est convexe
 - Fusion double si $C(D_{i,m}) \cup T_{i,m,j} \cup C(D_{m,j})$ est convexe
 - Chaque $T_{i,m,j}$ donne un candidat de DMC de $P_{i,j}$

Formulation en problème de programmation dynamique

Construire une solution

DMC des polygones $P_{i,k} \leq P_{i,j} \rightarrow \text{DMC de } P_{i,j}$

- On considère les triangles $(T_{i,m,j})_{i < m < j}$



- Combinaison de décompositions $D_{i,m}$ et $D_{m,j}$ de $P_{i,m}$ et $P_{m,j}$ et $T_{i,m,j}$
 - Fusion simple** si $C(D_{i,m}) \cup T_{i,m,j}$ ou $C(D_{m,j}) \cup T_{i,m,j}$ est convexe
 - Fusion double** si $C(D_{i,m}) \cup T_{i,m,j} \cup C(D_{m,j})$ est convexe
 - Chaque $T_{i,m,j}$ donne un candidat de DMC de $P_{i,j}$
- Comme à chaque fois $P_{i,m}, P_{m,j} \leq P_{i,j}$ on peut calculer une décomposition minimale de $P_{1,n} = P$ par **approche ascendante**.

Problèmes avec l'espace des états naïf

- ❶ Il peut y avoir **plusieurs DMC** d'un sous polygone et elles ne sont **pas équivalentes**.

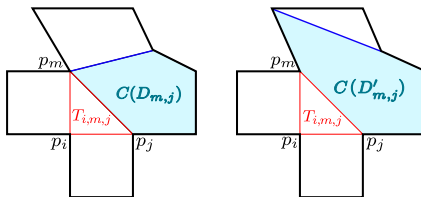


Figure: La DMC de gauche de $P_{m,j}$ fusionne avec $T_{i,m,j}$, pas celle de droite

Problèmes avec l'espace des états naïf

Couverture
vidéo-surveillance

Candidat n° 26655

- ① Il peut y avoir **plusieurs DMC** d'un sous polygone et elles ne sont **pas équivalentes**.

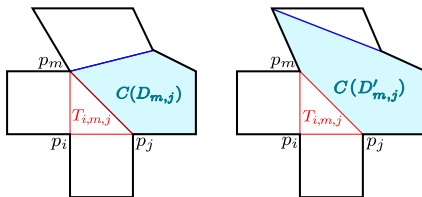


Figure: La DMC de gauche de $P_{m,j}$ fusionne avec $T_{i,m,j}$, pas celle de droite

- ② Il y a possiblement un nombre **exponentiel** de DMC

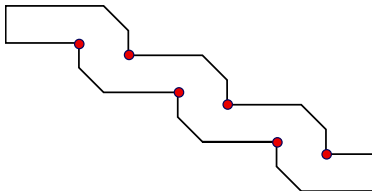


Figure: Polygone avec $O(2^n)$ DMCs

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Conséquences

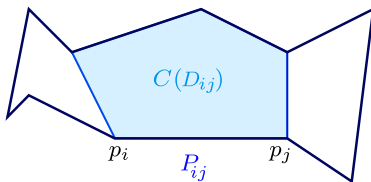
- 1 On doit stocker plus d'une DMC par $P_{i,j}$

Conséquences

- ① On doit stocker plus d'une DMC par $P_{i,j}$
- ② On ne peut pas stocker toutes les DMCs si on veut un algorithme en temps sub-exponentiel.

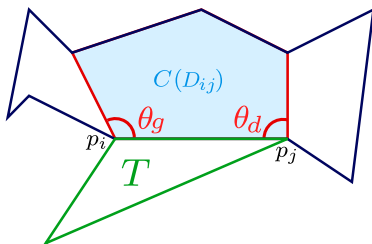
Réduction de l'espace des états

- La capacité des DMC des sous-polygones à fusionner avec un triangle dépend uniquement de leur **polygone convexe de base**.



Réduction de l'espace des états

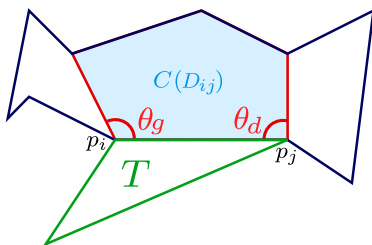
- La capacité des DMC des sous-polygones à fusionner avec un triangle dépend uniquement de leur **polygone convexe de base**.



- En particulier des angles à gauche et à droite de leur diagonale de base.

Réduction de l'espace des états

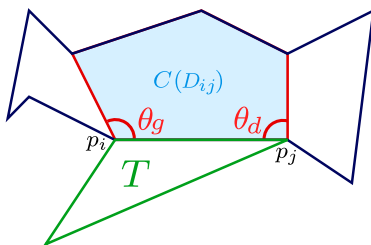
- La capacité des DMC des sous-polygones à fusionner avec un triangle dépend uniquement de leur **polygone convexe de base**.



- En particulier des angles à gauche et à droite de leur diagonale de base.
- On peut donc stocker une DMC de $P_{i,j}$ par paire de θ_g, θ_d

Réduction de l'espace des états

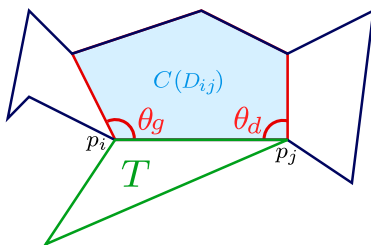
- La capacité des DMC des sous-polygones à fusionner avec un triangle dépend uniquement de leur **polygone convexe de base**.



- En particulier des angles à gauche et à droite de leur diagonale de base.
- On peut donc stocker une DMC de $P_{i,j}$ par paire de θ_g, θ_d
- Le nombre de paires de θ_g, θ_d est en $O\left(\binom{n}{2}\right) = O(n^2)$

Réduction de l'espace des états

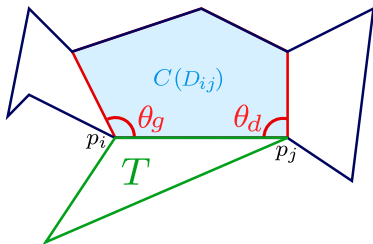
- La capacité des DMC des sous-polygones à fusionner avec un triangle dépend uniquement de leur **polygone convexe de base**.



- En particulier des angles à gauche et à droite de leur diagonale de base.
- On peut donc stocker une DMC de $P_{i,j}$ par paire de θ_g, θ_d
- Le nombre de paires de θ_g, θ_d est en $O\left(\binom{n}{2}\right) = O(n^2)$
- Le nombre de $P_{i,j}$ est en $O\left(\binom{n}{2}\right) = O(n^2)$

Réduction de l'espace des états

- La capacité des DMC des sous-polygones à fusionner avec un triangle dépend uniquement de leur **polygone convexe de base**.



- En particulier des angles à gauche et à droite de leur diagonale de base.
- On peut donc stocker une DMC de $P_{i,j}$ par paire de θ_g, θ_d
- Le nombre de paires de θ_g, θ_d est en $O\left(\binom{n}{2}\right) = O(n^2)$
- Le nombre de $P_{i,j}$ est en $O\left(\binom{n}{2}\right) = O(n^2)$
- Il y a $O(n)$ triangles $T_{i,m,j}$ à i, j fixés

On a donc une complexité totale en $O\left(n^2 \times n \times \binom{n^2}{2}\right) = O(n^7)$

Schéma algorithmique retenu

Schéma Algorithmique : Décomposition en polygones convexes minimale

```
1 Déterminer les  $(P_{i,j})$ 
2 Trier les  $(P_{i,j})$  par la mesure  $j - i$ 
3 Initialiser pour chaque  $P_{i,j}$  un ensemble vide de DMCs
4 pour chaque  $P_{i,j}$  dans l'ordre calculé faire
5   Créer un ensemble de candidats pour les DMC de  $P_{i,j}$ 
6   si  $j - i = 2$  alors On a un triangle
7     L'unique DMC de  $P_{i,j}$  est le triangle  $T_{i,i+1,j}$ 
8   sinon
9     pour chaque  $m$  tel que  $p_i p_m p_j$  est un triangle inclus dans  $A(P)$  faire
10      pour chaque DMC  $D_{i,m}$  et  $D_{m,j}$  de  $P_{i,m}$  et  $P_{m,j}$  faire
11        Rajouter la fusion de  $D_{i,m}$  et  $D_{m,j}$  et  $T_{i,m,j}$  aux candidats
12        On se ramène à des DMC en  $O(n^2)$ 
13      Conserver les décompositions minimales en cardinalité
14      Ne garder qu'une décomposition par paire de  $\theta_g, \theta_d$ 
14 retourner une décomposition minimale de  $P_{1,n} = P$ 
```

Progression

- ① Une borne supérieure pour les solutions optimales
 - Le théorème de la galerie d'art
 - Point de départ
- ② Décomposition en polygones convexes
 - Formulation en problème de programmation dynamique
 - Problèmes avec l'espace des états naïf
 - Réduction de l'espace des états
 - Schéma algorithmique retenu
- ③ Décomposition en polygones étoilés
 - Formulation en problème de programmation dynamique
 - Problèmes avec l'espace des états naïf
 - Vers un espace des états valide
 - Schéma algorithmique retenu
- ④ Application à la ville de Paris
 - Jeu de données
 - Heuristique supplémentaire
 - Résultats
- ⑤ Références
- ⑥ Annexe

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Définition : Polygone étoilé

Un **polygone étoilé** P est un polygone dans lequel il existe au moins un point $g \in A(P)$ tel que $Vis_P(g) = A(P)$.

Ce sont les polygones disposant de **solutions à 1 garde**.

Polygones étoilés

Définition : Polygone étoilé

Un **polygone étoilé** P est un polygone dans lequel il existe au moins un point $g \in A(P)$ tel que $\text{Vis}_P(g) = A(P)$.

Ce sont les polygones disposant de **solutions à 1 garde**.

Définition : Noyau d'un polygone étoilé

Un polygone étoilé P dispose d'un **noyau** noté $\text{Ker}(P)$ ensemble non vide défini par.

$$\text{Ker}(P) = \{n \in A(P), \text{Vis}_P(n) = A(P)\}$$

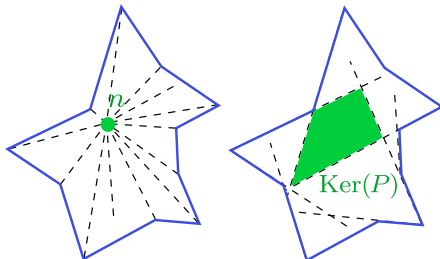


Figure: Polygone étoilé, une solution et son noyau

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Formulation en problème de programmation dynamique

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

- Même préordre sur les $P_{i,j}$

Formulation en problème de programmation dynamique

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

- Même préordre sur les $P_{i,j}$
- Si D dénote une décomposition minimale étoilée de $P_{i,j}$, $E(D)$ est défini de manière analogue aux polygones convexes de base

Problèmes avec l'espace des états naïf

Couverture
vidéo-surveillance

Candidat n° 26655

- ❶ Il peut y avoir **plus d'une DME** d'un sous polygone et **elles ne sont pas équivalentes**.

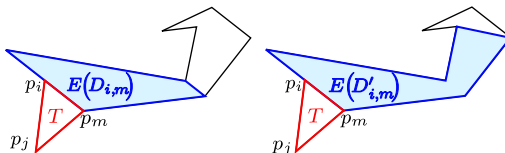


Figure: La DME de gauche fusionne avec T , pas celle de droite

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Problèmes avec l'espace des états naïf

- ❶ Il peut y avoir **plus d'une DME** d'un sous polygone et **elles ne sont pas équivalentes**.

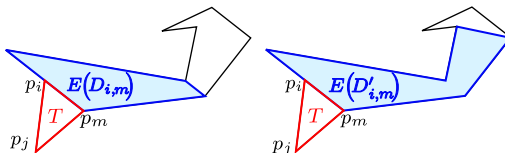


Figure: La DME de gauche fusionne avec T , pas celle de droite

- ❷ Stocker toutes les DMEs des $(P_{i,j})$ ne suffit pas:

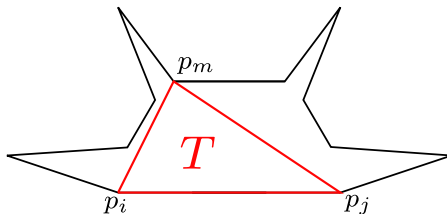


Figure: Toute DME $D_{i,m}$ de $P_{i,m}$ ne fusionne pas avec T alors que P est étoilé

③ Il y a possiblement un nombre **exponentiel** de DME

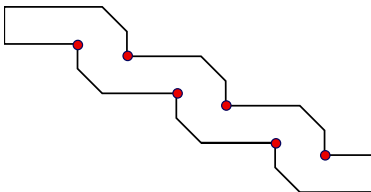


Figure: Polygone avec $O(2^n)$ DMEs

Vers un espace des états valide

Idée: polygones pseudo-étoilés

Définition : Polygone pseudo-étoilé

Un sous polygone $P_{i,j}$ de P est **pseudo-étoilé** par rapport à $g \in A(P)$ si
 $A(P_{i,j}) \subset \text{Vis}_P(g)$.

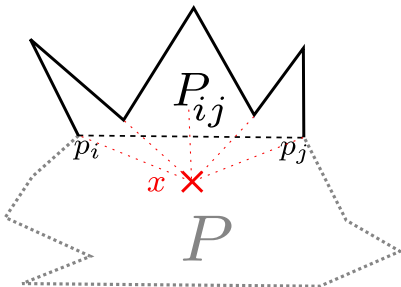


Figure: $P_{i,j}$ est pseudo étoilé par rapport à x

Idée: polygones pseudo-étoilés

Définition : Polygone pseudo-étoilé

Un sous polygone $P_{i,j}$ de P est **pseudo-étoilé** par rapport à $g \in A(P)$ si
 $A(P_{i,j}) \subset \text{Vis}_P(g)$.

"Lemme technique"

On dispose de $\mathcal{K} \subset A(P)$ fini tel que $\forall i, j$, si D est une DME de $P_{i,j}$,

$$\mathcal{K} \cap \text{Ker}(E(D)) \neq \emptyset$$

De plus cet ensemble est calculable en $O(n^4)$ opérations et $|\mathcal{K}| = O(n^4)$.

Vers un espace des états valide

"Lemme technique"

On dispose de $\mathcal{K} \subset \mathcal{A}(P)$ fini tel que $\forall i, j$, si D est une DME de $P_{i,j}$,

$$\mathcal{K} \cap \text{Ker}(E(D)) \neq \emptyset$$

De plus cet ensemble est calculable en $O(n^4)$ opérations et $|\mathcal{K}| = O(n^4)$.

On peut s'autoriser des DMEs des $P_{i,j}$ où $E(P_{i,j})$ est pseudo-étoilé par rapport à $x \in \mathcal{K}$ à travers $[p_i, p_j]$

On les note DME_x .

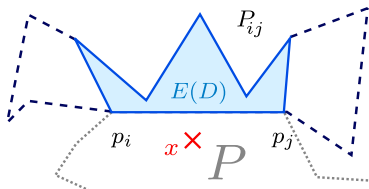


Figure: $\mathcal{A}(E(D)) \subset \text{Vis}_P(x)$

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Schéma algorithmique retenu

Schéma Algorithmique : Décomposition en polygones convexe minimale

```
1 Déterminer les  $(P_{i,j})$   $O(n^2)$ 
2 Calculer  $\mathcal{K}$   $O(n^4)$ 
3 Trier les  $(P_{i,j})$  par la mesure  $j - i$   $O(n^2 \log(n))$ 
4 Initialiser pour chaque  $P_{i,j}$  un ensemble vide de DMCs
5 pour chaque  $P_{i,j}$  dans l'ordre calculé faire  $O(n^2)$ 
6 |   Créer un ensemble de candidats pour les DMC de  $P_{i,j}$ 
7 |   si  $j - i = 2$  alors On a un triangle
8 |   |   pour chaque  $x \in \mathcal{K}$  faire
9 |   |   |   si  $p_i, p_{i+1}, p_j \in \text{Visp}(x)$  alors
10 |   |   |   |   Le triangle  $T_{i,i+1,j}$  est une  $DME_x$  de  $P_{i,j}$ 
11 |   sinon pour chaque  $m$  tel que  $T_{i,m,j} \subset A(P)$  faire  $O(n)$ 
12 |   |   pour chaque  $x$  dans  $\mathcal{K}$  faire  $O(n^4)$ 
13 |   |   |   initialiser un ensemble de candidats pour  $DME_x$ 
14 |   |   |   si les  $DME D_x^{i,m}$  et  $D_x^{m,j}$  de  $P_{i,m}$  et  $P_{m,j}$  double fusionnent alors
15 |   |   |   |   Ajouter la fusion de  $D_{i,m}$ ,  $D_{m,j}$  et  $T_{i,m,j}$  aux candidats  $O(1)$ 
16 |   |   |   si la  $DME D_x^{i,m}$  de  $P_{i,m}$  fusionne simplement avec  $T_{i,m,j}$  alors
17 |   |   |   |   Ajouter la fusion de  $D_{i,m}$ ,  $T_{i,m,j}$  et la plus petite DME réelle
18 |   |   |   |   |   de  $P_{i,j}$  aux candidats.  $O(1)$ 
19 |   |   |   si il n'y a pas de candidat alors
20 |   |   |   |   Ajouter la fusion de  $T_{i,m,j}$  et les plus petites DME réelles
21 |   |   |   |   |   (non-pseudo) de  $P_{i,j}$  et  $P_{m,j}$   $O(n \log n)$ 
22 |   |   |   Conserver une  $DME_x$  de  $P_{i,j}$  minimale en cardinalité
23 retourner une décomposition minimale de  $P_{1,n} = P$ 
```

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

On a un calcul en $O(\log(n) \times n^7)$

Progression

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

- ① Une borne supérieure pour les solutions optimales
Le théorème de la galerie d'art
Point de départ
- ② Décomposition en polygones convexes
Formulation en problème de programmation dynamique
Problèmes avec l'espace des états naïf
Réduction de l'espace des états
Schéma algorithmique retenu
- ③ Décomposition en polygones étoilés
Formulation en problème de programmation dynamique
Problèmes avec l'espace des états naïf
Vers un espace des états valide
Schéma algorithmique retenu
- ④ Application à la ville de Paris
Jeu de données
Heuristique supplémentaire
Résultats
- ⑤ Références
- ⑥ Annexe

Jeu de données

Plan de voirie - Chaussées: opendata.paris.fr
Environ 21000 polygones

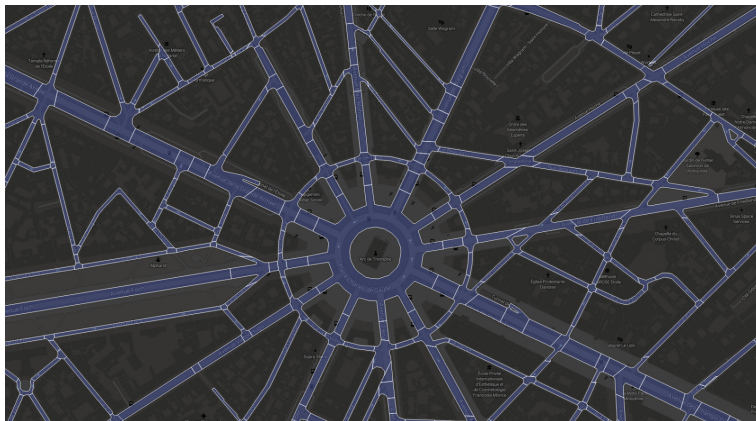


Figure: Les Champs-Élysées

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Jeu de données

Plan de voirie - Chaussées: opendata.paris.fr

Environ 21000 polygones



Figure: La butte Montmartre

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

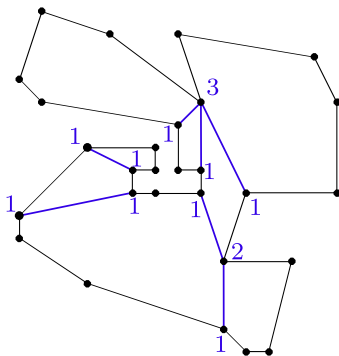
Application à la ville
de Paris

Références

Annexe

Heuristique supplémentaire

De manière analogue à la preuve du théorème de la galerie d'art on peut être **glouton** pour le placement des caméras



Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

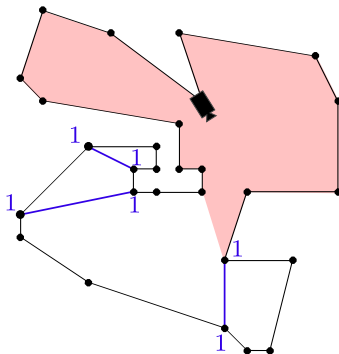
Annexe

Heuristique supplémentaire

Couverture
vidéo-surveillance

Candidat n° 26655

De manière analogue à la preuve du théorème de la galerie d'art on peut être **glouton** pour le placement des caméras



Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

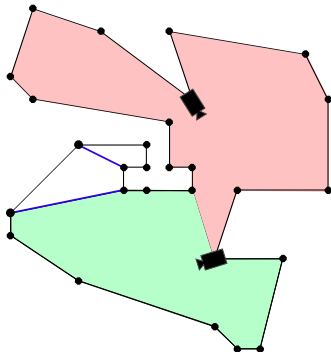
Application à la ville
de Paris

Références

Annexe

Heuristique supplémentaire

De manière analogue à la preuve du théorème de la galerie d'art on peut être **glouton** pour le placement des caméras



etc..

Résultats: nombre de caméras par polygones par approche

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

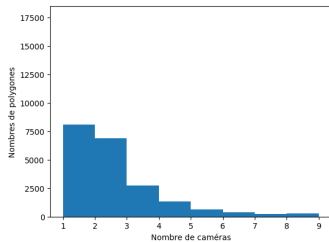


Figure: Décomposition en $\lfloor \frac{n}{3} \rfloor$

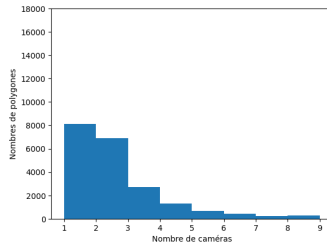


Figure: DMC

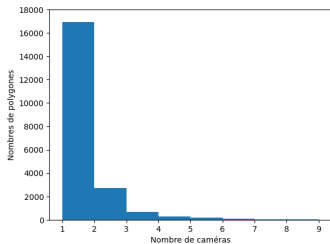


Figure: DMC + heuristique

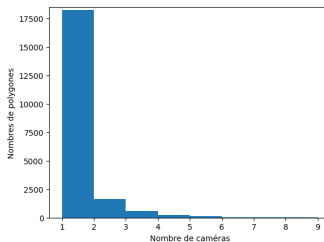


Figure: DME

Comparaison des résultats à la borne supérieure

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

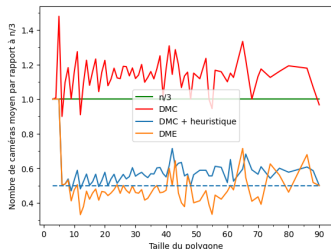


Figure: Comparaison

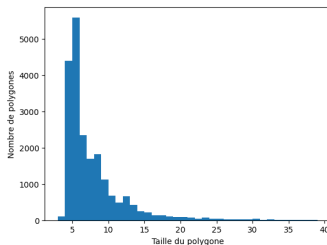


Figure: Quantité par taille de polygones

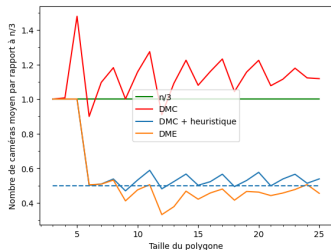


Figure: Comparaison - détail

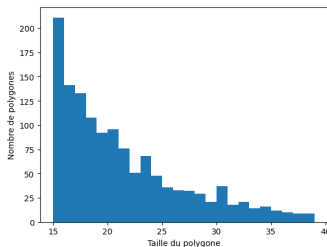


Figure: Quantité par taille de polygones
- détail

Comparaison - zone intéressante

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

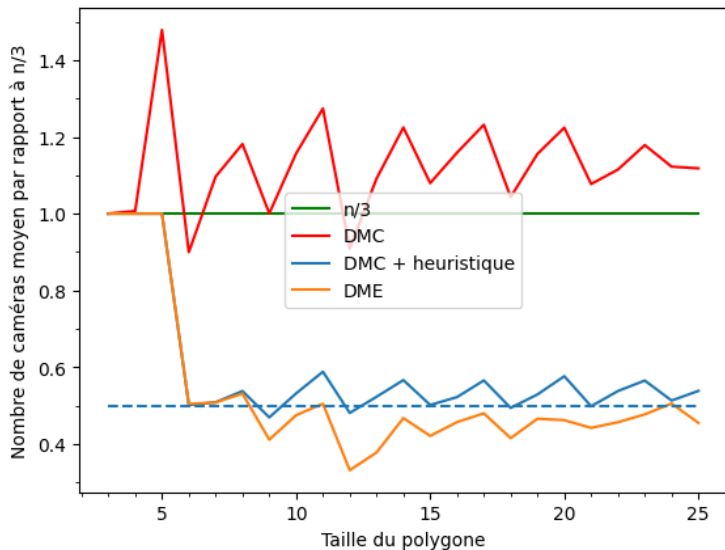


Figure: Quantité par taille de polygones - détail

Résultats : nombre total de caméras par approche

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Décomposition	# caméras	Calcul	Caméras / borne sup
Triangulaire	43892	N.A	100 %
Convexe	49365	2min	112 %
Convexe + heuristique	27730	2min	63 %
Étoilé	25733	3h	59 %

Progression

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

- ① Une borne supérieure pour les solutions optimales
Le théorème de la galerie d'art
Point de départ
- ② Décomposition en polygones convexes
Formulation en problème de programmation dynamique
Problèmes avec l'espace des états naïf
Réduction de l'espace des états
Schéma algorithmique retenu
- ③ Décomposition en polygones étoilés
Formulation en problème de programmation dynamique
Problèmes avec l'espace des états naïf
Vers un espace des états valide
Schéma algorithmique retenu
- ④ Application à la ville de Paris
Jeu de données
Heuristique supplémentaire
Résultats
- ⑤ Références
- ⑥ Annexe

Références



J. Mark Keil (1985)

Decomposing a polygon into simpler components

SIAM Journal on Computing Volume 14, (1985),799–817.



Steve Fisk (1975)

A short proof of Chvátal's watchman theorem

Journal of Combinatorial Theory Series B 8, 39-41.



Joseph O'Rourke (1987)

Art Gallery Theorems and Algorithms : Chapitre 9

Oxford University Press ISBN: 0-19-503965-3.



Direction de la Voirie et des Déplacements - Ville de Paris

Plan de voirie - Chaussées

[https://opendata.paris.fr/explore/dataset/
plan-de-voirie-chaussees/](https://opendata.paris.fr/explore/dataset/plan-de-voirie-chaussees/)

(visité le 10/12/2022)

Progression

- ① Une borne supérieure pour les solutions optimales
 - Le théorème de la galerie d'art
 - Point de départ
- ② Décomposition en polygones convexes
 - Formulation en problème de programmation dynamique
 - Problèmes avec l'espace des états naïf
 - Réduction de l'espace des états
 - Schéma algorithmique retenu
- ③ Décomposition en polygones étoilés
 - Formulation en problème de programmation dynamique
 - Problèmes avec l'espace des états naïf
 - Vers un espace des états valide
 - Schéma algorithmique retenu
- ④ Application à la ville de Paris
 - Jeu de données
 - Heuristique supplémentaire
 - Résultats
- ⑤ Références
- ⑥ Annexe

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

- Preuves
 - Existence de $\mathcal{K}$
- Code python
 - Algorithmes
 - Décomposition convexe
 - Décomposition étoilée
 - Représentation et débogage
 - Polygones
 - Affichage

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

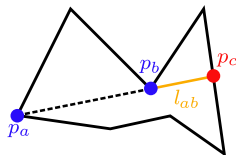
Annexe

Existence de $\mathcal{K}$

$\mathcal{K}$ est un ensemble de points potentiels des $\text{Ker} (E (D_{i,j}))$

Définition

Soient p_a et p_b sont des sommets de P si la demi-droite $(p_b, \overrightarrow{p_a p_b})$ intersecte ∂P^1 en p_c , on note l_{ab} le segment $[p_b, p_c] \subset A(P)$.



On note encore $L = \{l_{ab}, \text{l'angle en } p_b \text{ est rentrant et } p_a \in \text{Vis}_P(p_b)\}$

$$\mathcal{K}(P) = \{p_1, \dots, p_n\} \cup \left(\bigcup_{(l_1, l_2) \in L^2} (l_1 \cap l_2) \right)$$

Cardinal de $\mathcal{K}$: $|L| = O(n^2)$, d'où $|\mathcal{K}(P)| = O\left(n + \binom{n^2}{2}\right) \rightarrow |\mathcal{K}(P)| = O(n^4)$

Calcul de L : $p_a \in \text{Vis}_P(p_b)$? se calcule en $O(n) \rightarrow$ calcul en $O\left(n \times \binom{n}{2}\right) = O(n^3)$

Calcul de $\mathcal{K}$: les intersections sont en $O(1)$ à l_1, l_2 donnés donc $O\left(\binom{n^2}{2}\right)$

$\rightarrow$ calcul de $\mathcal{K}$ en $O(n^4)$

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

$$1_{\partial P} = \bigcup_{i=1}^n [p_i, p_{i+1}]$$

Existence de $\mathcal{K}$

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

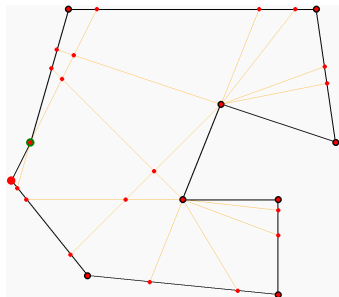


Figure: Exemple d'ensemble $\mathcal{K}$

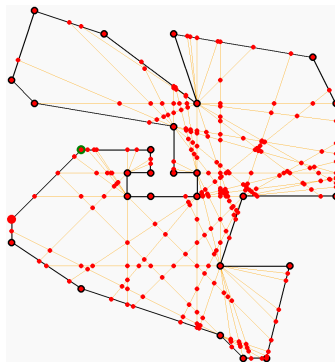


Figure: Exemple d'ensemble $\mathcal{K}$

“Lemme technique”

On dispose de $\mathcal{K} \subset A(P)$ fini tel que $\forall i, j$, si D est une DME de $P_{i,j}$,

$$\mathcal{K} \cap \text{Ker}(E(D)) \neq \emptyset$$

De plus cet ensemble est calculable en $O(n^4)$ opérations et $|\mathcal{K}| = O(n^4)$.

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Existence de $\mathcal{K}$

Soit D une DME de $P_{i,j}$

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Existence de $\mathcal{K}$

Soit D une DME de $P_{i,j}$

- 1 $\text{Ker}(E(D))$ est l'intersection des demis plans définis par les segments de $E(D)$

Soit D une DME de $P_{i,j}$

- ① $\text{Ker}(E(D))$ est l'intersection des demis plans définis par les segments de $E(D)$
- ② Si $E(D)$ est convexe, $\text{Ker}(E(D)) = E(D)$ donc $p_i, p_j \in \text{Ker}(E(D)) \cap \mathcal{K} \neq \emptyset$

Soit D une DME de $P_{i,j}$

- ① $\text{Ker}(E(D))$ est l'intersection des demis plans définis par les segments de $E(D)$
- ② Si $E(D)$ est convexe, $\text{Ker}(E(D)) = E(D)$ donc $p_i, p_j \in \text{Ker}(E(D)) \cap \mathcal{K} \neq \emptyset$
- ③ Sinon tout segment de $\text{ker}(E(D))$ est contenu dans un l_{ab} de L , prenons un sommet p de $\text{Ker}(E(D))$ alors deux segments de $\partial \text{Ker}(E(D))$ (qui sont dans des l_{ab}, l_{cd}) se coupent en p .
donc $p \in \bigcup_{\{l_1, l_2\} \subset L^2} (l_1 \cap l_2) \subset \mathcal{K}(P)$, d'où $\mathcal{K}(P) \cap \text{Ker}(E(D)) \neq \emptyset$

Décomposition convexe I

```
from __future__ import annotations
from copy import copy
import functools

from polygone import Point, Polygone, det, Segment

from futile import recherche_dichotomique

from affichage import Vue, debug

#preprocessing

#types utiles
Sommet = int
TriangleBase = Sommet
SousPolygoneConvexe = [Sommet]
SousPolygoneValide = (Sommet,Sommet)

""" TODO: tester l'amélioration que donne le rajout de point de steiner à intervalle régulier
"""

def sommets_visibles(P: Polygone, i1 : int, i2 : int) -> bool:
    """
    Arguments:
    - P : Polygone
    - i1: Sommet de P
    - i2: Sommet de P

    Sortie:
    Retourne vrai si i1 et i2 sont visibles dans P
    """
```

Décomposition convexe II

```
"""
i1, i2 = i1 % len(P), i2 % len(P)
i1, i2 = min(i1, i2), max(i1, i2)
if i1 == i2-1: return False

"""retourne vrai si le sommet d'indice [i1, i2] est inclus dans P"""
S = Segment(P[i1], P[i2])
"""On verifie que le segment ne passe pas en dehors du polygone"""

#On verifie que le segment est localement dans le cone P_(i1-1)i1(i1+1)
if P.est_convexe(i1):
    if not (det(P[i1-1] - P[i1], P[i2] - P[i1]) >= 0 and
            det(P[i2] - P[i1], P[i1+1] - P[i1]) >= 0 ):
        return False
elif not (det(P[i1-1] - P[i1], P[i2] - P[i1]) > 0 or
          det(P[i2] - P[i1], P[i1+1] - P[i1]) > 0 ):
    return False

#On verifie que le segment est localement dans le cone P_(i2-1)i2(i2+1)
if P.est_convexe(i2):
    if not (det(P[i2-1] - P[i2], P[i1] - P[i2]) >= 0 and
            det(P[i1] - P[i2], P[i2+1] - P[i2]) >= 0 ):
        return False
elif not (det(P[i2-1] - P[i2], P[i1] - P[i2]) > 0 or
          det(P[i1] - P[i2], P[i2+1] - P[i2]) > 0 ):
    return False

"""On vérifie que S = [P[i1], P[i2]] n'intersecte aucun segment de P"""
for i in range(len(P)):
    if i != i1 and i != i2 and i+1 != i1 and i+1 != i2:
        if S.intersecte(Segment(P[i], P[i+1])):
            return False
return True
```

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Décomposition convexe III

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
class DecompositionConvexe:
```

```
    def __init__(self, décomposé: Polygone,
                  polygones: list[SousPolygoneConvexe],
                  sPol : SousPolygoneValide,
                  polBase : SousPolygoneConvexe) -> None:

        """
        - décomposé : polygone dans lequel on fait la décomposition
        - polygones : polygones de la décompositions
        - sPol      : sous polygone valide décomposé par cette instance
        - polBase   : polygone convexe de base associé à sPol

        Par convention, polygones[0] = polBase
        """
        self.décomposé = décomposé
        self.sous_polygones = copy(polygones)
        self.polBase = polBase
        #nb polBase est dans self.polygones
        self.sous_polygone_valide = sPol
        #on calcule les arretes gauches

        # à priori ces deux valeurs sont 0 et -1
        self.indice_i = polBase.index(sPol[0])
        self.indice_j = polBase.index(sPol[1])

        self.arrete_gauche = décomposé[polBase[self.indice_i+1]] - décomposé[polBase[self.indice_i]]
        self.arrete_droite = décomposé[polBase[self.indice_j-1]] - décomposé[polBase[self.indice_j]]

    def __len__(self) -> int:
        return len(self.sous_polygones)

    def peut_fusionner_a_gauche(self, i: Sommet, m :Sommet, j : Sommet) -> bool:
```

Décomposition convexe IV

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
"""
- (i,m,j) : représente un triangle de fusion potentielle à gauche
"""
ij = self.décomposé[j] - self.décomposé[i]
mj = self.décomposé[j] - self.décomposé[m]

return (det(ij, self.arrete_gauche) >= 0) and (det(self.arrete_droite, mj) >= 0)

def peut_fusionner_a_droite(self, i: Sommet, m: Sommet, j: Sommet) -> bool:
    """
    - (i,m,j) : représente un triangle de fusion potentielle à droite
    """
    ji = self.décomposé[i] - self.décomposé[j]
    mi = self.décomposé[i] - self.décomposé[m]

    return (det(mi, self.arrete_gauche) >= 0) and (det(self.arrete_droite, ji) >= 0)

def __str__(self) -> str:
    polys = " ".join(map(str, self.sous_polygones))
    return f"Deco: {self.sous_polygone_valide[0]}-{self.sous_polygone_valide[1]}: {polys}"

def __repr__(self) -> str:
    return str(self)

def segmentsDeCoupe(self) -> list[Segment]:
    # Ce sont les sauts d'indices
    segments = set()
    for pol in self.sous_polygones:
        for i in range(len(pol)-1):
            if pol[i+1] - pol[i] > 1:
                segments.add(Segment(self.décomposé[pol[i+1]],
                                     self.décomposé[pol[i]]))
    )
    return list(segments)
```

Décomposition convexe V

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
def nb_de_caméra(self) -> int:
    # on construit le graphe de la décomposition
    adjacence = [set() for _ in range(len(self.décomposé))]
    for i_spol in range(len(self.sous_polygones)):
        pol = self.sous_polygones[i_spol]
        for i in range(len(pol)+1):
            j, k = max((i+1) % len(pol), i % len(pol)), min((i+1) %
                                                             len(pol), i % len(pol))
            if pol[j] - pol[k] > 1:
                # pi+1 est un segment de coupe
                adjacence[pol[j]].add(i_spol)
                adjacence[pol[k]].add(i_spol)

    # On procède gloutonnement pour couvrir chaque sous polygone
    cameras = 0
    while len(adj_max := max(adjacence, key=lambda e: len(e))) > 0:
        cameras += 1
        for adj in adjacence:
            if not (adj is adj_max):
                adj.difference_update(adj_max)
        adj_max.clear()
    return cameras
```

```
def mettre_à_jour_XR(P : Polygone,
                    XR : dict[SousPolygoneValide, list[DecompositionConvexe]],
                    XL : dict[SousPolygoneValide, list[DecompositionConvexe]],
                    t : tuple[Sommet, TriangleBase, Sommet], #(i, m, j) est le triangle de base sur lequel
                    ↪ on travaille
```

Décomposition convexe VI

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
        minimum : int, #taille minimale
    ) -> None:
i,m,j = t

def fusionne_a_gauche(A: DecompositionConvexe) -> bool:
    return A.peut_fusionner_a_gauche(i,m,j)

def fusionne_a_droite(A: DecompositionConvexe) -> bool:
    return A.peut_fusionner_a_droite(i,m,j)

#On récupère la décomposition fusionnant à gauche d'angle gauche minimal nomée A

fusionnant_gauche = filter(fusionne_a_gauche, XL[(i,m)])

A : DecompositionConvexe = next(fusionnant_gauche, None)
for D in fusionnant_gauche:
    # on compare les angles à gauche
    if det(D.arrete_gauche, A.arrete_gauche) > 0:
        A = D

base = [i,m,j]
polygones = [base]

if A != None:
    base = A.polBase + [j]
    polygones[0] = base
    polygones.extend(A.sous_polygones[1:])
elif len(XL[(i,m)]) > 0:
    A = XL[(i,m)][0]
    polygones.extend(A.sous_polygones)
```

Décomposition convexe VII

#On choisit B une décomposition minimale de P_mj quelconque

```
if len(XR[(m,j)]) > 0:
    B = XR[(m,j)][0]
    polygones.extend(B.sous_polygones)
```

```
D = DecompositionConvexe(
    décomposé=P,
    polygones= polygones,
    sPol=(i,j),
    polBase=base
)
```

```
if len(D) > minimum:
    pass
if len(D) < minimum:
    #on a trouvé mieux
    XR[(i,j)] = [D]
    minimum = len(D)
if len(D) == minimum:
    #on a trouvé une solution de meme cardinalité
    XR[(i,j)].append(D)
```

#On prend une decomposition B' fusionnée avec T_img

```
fusonnant_droit = filter(fusionne_a_droite, XR[(m,j)])
```

```
B : DecompositionConvexe = next(fusonnant_droit, None)
for D in fusonnant_droit:
    # on compare les angles à droite
    if det(B.arrete_droite, D.arrete_droite) > 0:
        B = D
```

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Décomposition convexe VIII

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
base = [i,m,j]
polygones = [base]

if B != None:
    # On fusionne B et T_imj à droite
    base = [i] + B.polBase
    polygones[0] = base
    polygones.extend(B.sous_polygones[1:])

elif len(XR[(m,j)]) > 0:
    B = XR[(m,j)][0]
    polygones.extend(B.sous_polygones)

#On choisit A une décomposition minimale de P_mj quelconque
if len(XL[(i,m)]) > 0:
    A = XL[(i,m)][0]
    polygones.extend(A.sous_polygones)

D = DecompositionConvexe(
    décomposé=P,
    polygones = polygones,
    sPol=(i,j),
    polBase=base
)

if len(D) > minimum:
    pass
if len(D) < minimum:
    #on a trouvé mieux
    XR[(i,j)] = [D]
    minimum = len(D)
if len(D) == minimum:
    #on a trouvé une solution de meme cardinalité
```

Décomposition convexe IX

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
XR[(i,j)].append(D)

return minimum

def decomposition_convexe_keil(P : Polygone) -> tuple[list[DecompositionConvexe],
↳ list[DecompositionConvexe]]:

     #(PREPROCESSING)
    # 1 déterminer les sommets de référence
    est_ref = lambda i : P.est_entaille(i) or i in (0,n-1)
    n = len(P)
    ref : list[Sommet] = list(filter(est_ref, range(len(P))))  #NB: ref est triée
    #est_ref = lambda i: recherche_dichotomique(ref, i)
    N = len(ref)
    # Détermination des listes de visibilité
    Visibilités : list[list[Sommet]] = list(list() for _ in range(n))

     # on stocke les paires dans un ensemble
    # -> pas de répétition, les paires sont aussi des ensembles
    Visibilité = set()  # les paires calculées définissent les P_ij
    for i in ref:
        for j in range(n):
            if i != j and sommets_visibles(P, i, j):
                Visibilités[i].append(j)
                if P.est_convexe(j):
                    Visibilités[j].append(i)
                Visibilité.add((min(i,j), max(i,j)))

    estSousPolygoneValide = lambda s : (est_ref(s[0]) or est_ref(s[1])) and (s[1]-s[0]) > 1
     # pas de segments de P
    SousPolygonesValides : list[SousPolygoneValide] = list(filter(estSousPolygoneValide, Visibilité))
    SousPolygonesValides.append((0,n-1))
```

Décomposition convexe X

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
SousPolygonesValides.sort(key=lambda p : max(p) - min(p))

#Calcul des triangles de base
TrianglesBase : dict[SousPolygoneValide, list[Sommet]] = {s: list() for s in SousPolygonesValides}

#On trie les visibilitées pour accélérer les vérifications grace à la recherche dichotomique

for v in Visibilités:
    v.sort()

est_triangle_de_base = lambda i,m,j : ((m-i == 1) or recherche_dichotomique(Visibilités[m], i)) and
↪ ((j-m == 1) or recherche_dichotomique(Visibilités[m], j))

for s in SousPolygonesValides: #O(n**2) iterations -> O(n**3.log(N))
    i,j=s
    for m in range(i+1, j): #O(n) itérations de la boucle -> O(n.log(N))
        if est_triangle_de_base(i,m,j): #!-> O(log(N)) (recherche dichotomique)
            TrianglesBase[s].append(m)

# On dispose maintenant de
# * Visibilités (list[list[Sommet]])
# * SousPolygones (list[SousPolygone])
# * TrianglesBase

#Partie programmation dynamique
XR : dict[SousPolygoneValide,list[DecompositionConvexe]] = {s:list() for s in SousPolygonesValides}
XL : dict[SousPolygoneValide,list[DecompositionConvexe]] = {s:list() for s in SousPolygonesValides}
#On initialise les bords
for i in range(len(P)-1):
    XR[(i, i+1)] = list()
    XL[(i, i+1)] = list()

# On trie toujours les XR et XL par angle a gauche
```

Décomposition convexe XI

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
for s in SousPolygonesValides:
    i,j = s
    # Calcul de XR[s] et XL[s]

    #cas de base
    if j-i == 2: #triangles
        tri = [i, i+1, j]
        XR[s] = [DecompositionConvexe(décomposé=P,
                                       polygones=[tri],
                                       sPol=s,
                                       polBase=tri)
                ]
        XL[s] = [DecompositionConvexe(décomposé=P,
                                       polygones=[tri],
                                       sPol=s,
                                       polBase=tri)]

    continue
    # taille d'une décomposition minimale
    M = n

    for m in TrianglesBase[s]:
        # par croissance de SousPolygonesValides pour (i,j) |-> j-i on a déjà calculé
        # XR[(i,m)], XL[(i,m)], XR[(m,j)], XL[(m,j)]

        # XR-ij et XL-ij se différencient dans la partie
        # où on les épure pour ne garder que leurs représentants
        #M = mettre_à_jour_XL(P, XR, XL, (i,m,j), M)
        M = mettre_à_jour_XR(P, XR, XL, (i,m,j), M)

    XL[(i,j)] = copy(XR[(i,j)])

    #On donne la propriété RR aux XR[(i,j)]
    # (on ne garde que les représentants)
    #On trie par angle à droite
```

Décomposition convexe XII

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
XR[(i,j)].sort(key=functools.cmp_to_key(lambda D1 , D2 : det(D1.arrete_droite,
↪ D2.arrete_droite)))
RR = [XR[(i,j)][0]]

for D in XR[(i,j)][1:]:
    if det(D.arrete_gauche, RR[-1].arrete_gauche) >= 0:
        RR.append(D)
XR[(i,j)] = RR

#On donne la propriété LR aux XL[(i,j)]
#Fonction de comparaison
XL[(i,j)].sort(key=functools.cmp_to_key(lambda D1 , D2 : det(D2.arrete_gauche,
↪ D1.arrete_gauche)))

LR = [XL[(i,j)][0]]
for D in XL[(i,j)][1:]:
    if det(RR[-1].arrete_droite, D.arrete_droite) >= 0:
        LR.append(D)
XL[(i,j)] = LR

#Ces ensembles sont non tous vides
return XR[(0,n-1)], XL[(0,n-1)]

if __name__ == '__main__':
    P1 = Polygone([Point(0,2), Point(1, 3), Point(7,3), Point(2,2), Point(3, 0), Point(1,0)])
    P2 = Polygone([Point(0,0), Point(1,0), Point(1,1), Point(2,1), Point(2,0), Point(3,0), Point(3, -1),
↪ Point(2, -1), Point(2, -2), Point(1, -2), Point(1, -1), Point(0, -1)])
    vue = Vue()
    vue.ajouterPolygone(P2)
```

Décomposition convexe XIII

```
#vue.afficher().show()
for C in decomposition_convexe_keil(P2):
    for D in C:
        print(D.segmentsDeCoupe())
        print(" ".join(map(str, D.sous_polygones)))
```

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Décomposition étoilée I

```
from __future__ import annotations
from polygone import Point, Polygone, Segment, det, Vecteur, number
from itertools import combinations
from affichage import Vue
```

```
Sommet = int
Triangle = list[Sommet, Sommet, Sommet]
TriangleBase = Sommet
SousPolygone = tuple[Sommet, Sommet]
SousPolygoneEtoile = list[Sommet]
```

#TODO: ajouter recherche dichotomique

```
def signe(x: number) -> (1 | 0 | -1):
    if x == 0:
        return 0
    elif x > 0:
        return 1
    else:
        return -1
```

```
def sommets_visibles(P: Polygone, i1 : int, i2 : int) -> bool:
```

"""

Arguments:

- P : Polygone

- i1: Sommet de P

- i2: Sommet de P

Sortie:

Retourne vrai si Vis_P(i1) contient i2

"""

```
i1, i2 = i1 % len(P), i2 % len(P)
```

```
i1, i2 = min(i1, i2), max(i1, i2)
```

```
if abs(i1 - i2) == 1 or (i1, i2) == (0, len(P)-1):
```

Décomposition étoilée II

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
return True

"""retourne vrai si le sommet d'indice [i1, i2] est inclus dans P"""
S = Segment(P[i1], P[i2])
"""On verifie que le segment ne passe pas en dehors du polygone"""

#On verifie que le segment est localement dans le cone P_(i1-1)i1(i1+1)
if P.est_convexe(i1):
    if not (det(P[i1-1] - P[i1], P[i2] - P[i1]) >= 0 and
            det(P[i2] - P[i1], P[i1+1] - P[i1]) >= 0 ):
        return False
elif not (det(P[i1-1] - P[i1], P[i2] - P[i1]) > 0 or
          det(P[i2] - P[i1], P[i1+1] - P[i1]) > 0 ):
    return False

#On verifie que le segment est localement dans le cone P_(i2-1)i2(i2+1)
if P.est_convexe(i2):
    if not (det(P[i2-1] - P[i2], P[i1] - P[i2]) >= 0 and
            det(P[i1] - P[i2], P[i2+1] - P[i2]) >= 0 ):
        return False
elif not (det(P[i2-1] - P[i2], P[i1] - P[i2]) > 0 or
          det(P[i1] - P[i2], P[i2+1] - P[i2]) > 0 ):
    return False

"""On vérifie que S = [P[i1], P[i2]] n'intersecte aucun segment de P"""
for i in range(len(P)):
    if i != i1 and i != i2 and i+1 != i1 and i+1 != i2:
        if S.intersecte(Segment(P[i], P[i+1])):
            return False
return True

# TODO: - remettre polBase pour pouvoir fusionner les triangles, finir l'algo (bientot)
# TODO: calculer les KER des polygones pour s'assurer que ca marche
class DecompositionEtoile:
    """Implémente les (MD_x)"""
```

Décomposition étoilée III

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
def __init__(self,
    pseudo: bool,
    décomposé: Polygone,
    polygones: list[SousPolygoneEtoile],
    pol_base: list[Sommet],
    sPol: SousPolygone) -> None:
    """
    - x (Point): le point à partir du quel sPol est vu
    - pseudo (bool): Booléen indiquant si la décomposition est
    - décomposé (Polygone): Polygone décomposé
    - polygones (list[Polygone]): Les sous polygones de la décomposition
    - sPol (SousPolygone): le P_ij décomposé par cette instance, P_in sera la décomposition de P
    """
    self.est_pseudo = pseudo
    self.décomposé = décomposé
    self.polygones = polygones
    self.pol_base = pol_base
    self.sPol = sPol

@property
def cout(self) -> number:
    return len(self.polygones)

def __repr__(self) -> str:
    return f"{'pseudo-' if self.est_pseudo else '' }deco{self.sPol}"

def debug(self):
    couleurs = ["green", "blue", "red", "orange", "purple", "yellow"]
    v = Vue()
    v.ajouterPolygone(self.décomposé)
    for k in range(len(self.polygones)):
        p = self.polygones[k]
        Q = Polygone(list(map(lambda i : self.décomposé[i], p)))
```

Décomposition étoilée IV

```
v.ajouterPolygone(Q, couleur=couleurs[k%len(couleurs)])
return v.afficher()

def intersection_demi_droite_segment(O: Point, v: Vecteur, s: Segment) -> (None | Point):
    """ Retourne si il en existe une unique, intersection de la demie droite affine D et de s
    où  $D = \{O + kv, k \text{ réel}\}$ 
    """
    a, b = s.p1, s.p2
    ab = b-a
    Oa = a - O

    D = det(v, ab)
    if D == 0:
        # on regarde si les deux points sont bien dans la demie droite affine
        if det(a - O, v) == 0: # meme droite affine
            lambda_1 = (a.x - O.x) * signe(v.x)
            lambda_2 = (b.x - O.x) * signe(v.x)
            if lambda_1 > 0 and lambda_2 > 0: # demi-droite
                if lambda_1 < lambda_2:
                    return a
                else:
                    return b
            else:
                return None
        # si c'est le cas on retourne le plus proche

    L = Oa.x * ab.y - ab.x * Oa.y
    t = Oa.x * v.y - v.x * Oa.y

    # on s'épargne une division yes!
    if L * signe(D) > 0 and 0 <= t * signe(D) <= abs(D):
        return O + (L/D) * v
```

Décomposition étoilée V

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
else:
    return None

def point_dans_triangle(T: tuple[Point, Point, Point], P: Point) -> bool:
    #On utilise B = (AB, AC) comme base de l'espace
    #On determine les coordonnées (s,t) de AP dans la base B
    # P est dans T si 0 <= s, t <=1 et s+t <=1 (intersection des demis plans)
    A, B, C = T

    AP = P-A
    AB, AC = B-A, C-A

    Aire = det(AB, AC)

    s = det(AP, AC) * signe(Aire) #en fait s = det(AP, AC)/Aire
    t = det(AB, AP) * signe(Aire) #en fait t = det(AB, AP)/Aire
    return 0 <= s <= abs(Aire) and 0 <= t <= abs(Aire) and s+t <= 1

def coutDM(DM):
    return float("inf") if DM is None else DM.cout

def decomposition_etoile_keil(P: Polygone) -> DecompositionEtoile:

    N = len(P)
    # Preprocessing
    # 1. Visibilité entre sommets
    Visibilité: list[tuple[Sommet, Sommet]] = list()
    Visibilités: list[set[Sommet]] = list(set() for _ in range(N))
    for i in range(N):
        for j in range(i+1, N):
            if sommets_visibles(P, i, j):
                Visibilités[i].add(j)
                Visibilités[j].add(i)
            if j-i > 1:
```

Décomposition étoilée VI

```
Visibilité.append((i, j))

# 1.1 Trier les paires v_i v_j (avec i < j) par (j-i)
Visibilité.sort(key=lambda p: p[1] - p[0])

# 2. calcul de KER(P)
KER: set[Point] = set()
# on ajoute les points de P
for i in range(N):
    KER.add(P[i])

# on calcule L tel que défini par Keil
L: set[Segment] = set()

Visibilité_KER: dict[Point, set[Sommet]] = {x: set() for x in KER}

for i, j in Visibilité:
    if P.est_entaille(i):
        for s in P.segments:
            I = intersection_demi_droite_segment(P[i], P[i] - P[j], s)
            if I != None:
                S = Segment(P[i], I)
                # On vérifie que S est bien valide
                # d'abord S ne sort pas
                segment_ne_sort_pas = det(
                    P[i-1] - P[i], I - P[i]) > 0 or det(I - P[i], P[i+1] - P[i]) > 0
                segment_n_intersecte_pas = all(
                    not t.intersecte(S) for t in P.segments if t != s)

                if segment_ne_sort_pas and segment_n_intersecte_pas:
                    L.add(S)
                    if not (I in Visibilité_KER):
                        Visibilité_KER[I] = set()
                    Visibilité_KER[I].add(i % N)
```

Décomposition étoilée VII

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
# v.ajouterSegments([Segment(P[i], P[j])], "red")

if P.est_entaille(j):
    for s in P.segments:
        I = intersection_demi_droite_segment(P[j], P[j] - P[i], s)
        if I != None:
            S = Segment(P[j], I)
            segment_ne_sort_pas = det(
                P[j-1] - P[j], I - P[j]) > 0 or det(I - P[j], P[j+1] - P[j]) > 0
            segment_n_intersecte_pas = all(
                not t.intersecte(S) for t in P.segments if t != s)

            if segment_ne_sort_pas and segment_n_intersecte_pas:
                L.add(S)
                if not (I in Visibilité_KER):
                    Visibilité_KER[I] = set()
                Visibilité_KER[I].add(j % N)

# l_ab ou ab = i(i+1), i(i-1)
for i in range(N):
    if P.est_entaille(i):
        for s in P.segments:
            # i(i+1)
            I = intersection_demi_droite_segment(P[i], P[i] - P[i-1], s)
            if I != None:
                S = Segment(P[i], I)

                segment_ne_sort_pas = det(
                    P[i-1] - P[i], I - P[i]) > 0 or det(I - P[i], P[i+1] - P[i]) > 0
                segment_n_intersecte_pas = all(
                    not t.intersecte(S) for t in P.segments if t != s)

                if segment_ne_sort_pas and segment_n_intersecte_pas:
                    if not (I in Visibilité_KER):
                        Visibilité_KER[I] = set()
```

Décomposition étoilée VIII

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
Visibilité_KER[I].add(i-1 % N)
L.add(S)

# i(i-1)
I = intersection_demi_droite_segment(P[i], P[i] - P[i+1], s)

if I != None:
    S = Segment(P[i], I)

    SegmentNeSortPas = det(
        P[i-1] - P[i], I - P[i]) >= 0 or det(I - P[i], P[i+1] - P[i]) >= 0
    SegmentNintersectePas = all(
        not t.intersecte(S) for t in P.segments if t != s)

    if SegmentNeSortPas and SegmentNintersectePas:
        if not I in Visibilité_KER:
            Visibilité_KER[I] = set()

        Visibilité_KER[I].add(i+1 % N)
        L.add(S)

# Calcul des intersections de L
for s1, s2 in combinations(L, r=2):
    I = s1.intersection(s2)

    if I != None:
        KER.add(I)
        if not (I in Visibilité_KER):
            Visibilité_KER[I] = set()

for s in L:
    KER.add(s.p2)
    if not (I in Visibilité_KER):
        Visibilité_KER[I] = set()
```

Décomposition étoilée IX

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
# KER est constuit
```

```
# 3. Visibilité des éléments de KER
```

```
for x in KER:
    try: # si x est un point de P, on a déjà fait le calcul
        i = P.points.index(x)

        Visibilité_KER[x] = Visibilité_KER[x].union(Visibilités[i])
        continue
    except ValueError:
        for i in range(N):
            # On détermine si P[i] est visible depuis x
            # On vérifie que xP[i] est localement dans le cone en v
            if P.est_convexe(i):
                if not (det(P[i-1] - P[i], x - P[i]) >= 0 and
                        det(x - P[i], P[i+1] - P[i]) >= 0):
                    continue
            elif not (det(P[i-1] - P[i], x - P[i]) > 0 or
                     det(x - P[i], P[i+1] - P[i]) > 0):
                continue

        visibles = True
        for j in range(N):
            s = Segment(P[j], P[j+1])
            if s.intersecte(Segment(x, P[i])):
                if s.contient(x):
                    # on vérifie alors que xP[i] est localement dans le cone en x
                    if not det(P[j] - x, P[i] - x) >= 0:
                        visibles = False
                        break
                else:
                    visibles = False
```

Décomposition étoilée X

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
        break

    if visibles:
        Visibilité_KER[x].add(i)

# 4. Calcul des triangles de base des  $P_{ij}$  pour ( $i < j$ )
TrianglesBase: list[set[TriangleBase]] = [
    set() for _ in range(len(Visibilité))]
for k in range(len(Visibilité)):
    i, j = Visibilité[k]

    for m in range(i+1, j):
        if m in Visibilités[i] and m in Visibilités[j]:
            TrianglesBase[k].add(m)

# procédure de programmation dynamique
decompositions: dict[SousPolygone, dict[Point, DecompositionEtoile]] = {
    sp: dict() for sp in Visibilité}
for k in range(len(Visibilité)):
    s = Visibilité[k]
    i, j = s

    if j-i == 2:
        # on a un triangle
        for x in KER:
            # soit x est dans le triangle  $P_{i(i+1)j}$ , on a une décomposition étoilé minimale
            if point_dans_triangle((P[i], P[i+1], P[i+2]), x):
                dec = DecompositionEtoile(pseudo=False,
                    décomposé=P,
                    polygones=[[i, i+1, i+2]],
                    pol_base=[i, i+1, i+2],
                    sPol=(i, j),
                )
                decompositions[s][x] = dec
```

Décomposition étoilée XI

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
# sinon si x voit les sommets de  $P_{-i(i+1)j}$ , on a une pseudo décomposition étoilée minimale
elif Visibilité_KER[x].issuperset({i, i+1, i+2}):
    dec = DecompositionEtoile(pseudo=True,
                              décomposé=P,
                              polygones=[[i, i+1, i+2]],
                              pol_base=[i, i+1, i+2],
                              sPol=(i, j),
                              )
    decompositions[s][x] = dec
continue

for m in TrianglesBase[k]:
    for x in KER:
        if j-m == 1:
            decompositions[(m, j)] = dict()
            if Visibilité_KER[x].issuperset({j, m}) or x in (P[m], P[j]):
                decompositions[(m, j)][x] = DecompositionEtoile(pseudo=False, décomposé=P,
                                                                    ↪ polygones=[], pol_base=[], sPol=(m, j))
        if m-i == 1:
            decompositions[(i, m)] = dict()
            if Visibilité_KER[x].issuperset({i, m}) or x in (P[i], P[m]):
                decompositions[(i, m)][x] = DecompositionEtoile(pseudo=False, décomposé=P,
                                                                    ↪ polygones=[], pol_base=[], sPol=(i, m))

DM_x: (None | DecompositionEtoile) = None
if x in decompositions[(i, m)] and x in decompositions[(m, j)]:
    # on peut double fusionner
    d1 = decompositions[(i, m)][x]
    d2 = decompositions[(m, j)][x]

    # les polygones partagent le point  $P_m$ 

    if len(d1.polygones) == 0:
```

Décomposition étoilée XII

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
d1.polygones = [list(d1.sPol)]
if len(d2.polygones) == 0:
    d2.polygones = [list(d2.sPol)]

polBase = d1.polygones[0] + d2.polygones[0][1:]

polys = list()
polys.append(polBase)
polys.extend(d1.polygones[1:])
polys.extend(d2.polygones[1:])

dfus = DecompositionEtoile(
    pseudo=d1.est_pseudo or d2.est_pseudo,
    décomposé=P,
    polygones=polys,
    pol_base=polBase,
    sPol=(i, j)
)
DM_x = min(DM_x, dfus, key=coutDM)
else:
    # on cherche des fusions simples
    if x in decompositions[(i, m)] and x in decompositions[(m, j)]:
        if j in Visibilité_KER[x]:
            # fusion simple de Pim avec Timj
            d1 = decompositions[(i, m)][x]
            d2 = decompositions[(m, j)][x]

            polBase = d1.polygones[0] + [j]

            polys = list()
            polys.append(polBase)
            polys.extend(d1.polygones[1:])
            polys.extend(d2.polygones)
            dfus = DecompositionEtoile(
                pseudo=point_dans_triangle((P[i], P[m], P[j]), x),
```

Décomposition étoilée XIII

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
décomposé=P,
polygones=polys,
pol_base=polBase,
sPol=(i, j)
)
DM_x = min(DM_x, dfus, key=coutDM)
if i in Visibilité_KER[x]:
    # fusion simple de P_mj avec T_imj
    d1 = decompositions[(i, m)][x]
    d2 = decompositions[(m, j)][x]

    polBase = [i] + d2.polygones[0]

    polys = list()
    polys.append(polBase)
    polys.extend(d1.polygones)
    polys.extend(d2.polygones[1:])
    dfus = DecompositionEtoile(
        pseudo=point_dans_triangle((P[i], P[m], P[j]), x),
        décomposé=P,
        polygones=polys,
        pol_base=polBase,
        sPol=(i, j)
    )
    DM_x = min(DM_x, dfus, key=coutDM)

if (DM_x is None and
    (point_dans_triangle((P[i], P[m], P[j]), x) or # soit x est dans P_imj
     (Visibilité_KER[x].issuperset({i, m, j}) and Segment(x, P[m]).intersecte(Segment(P[i],
     ↪ P[j])))): # Soit x voit P_i, P_j, P_m à travers p_i p_j

try:
    # on fusionne les minimaux à gauche et à droite
    d1 = min(filter(lambda d: not d.est_pseudo,
                    decompositions[(i, m)].values()), key=coutDM)
```

Décomposition étoilée XIV

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
d2 = min(filter(lambda d: not d.est_pseudo,
                decompositions[(m, j)].values()), key=coutDM)

polBase = [i, m, j]
polys = [polBase] + d1.polygones + d2.polygones

dfus = DecompositionEtoile(
    pseudo=point_dans_triangle((P[i], P[m], P[j])), x,
    décomposé=P,
    polygones=polys,
    pol_base=polBase,
    sPol=(i, j)
)
DM_x = min(DM_x, dfus, key=coutDM)
except:
    pass

if DM_x != None:
    decompositions[(i, j)][x] = DM_x

return min(decompositions[(0, N-1)].values(), key=coutDM)

if __name__ == '__main__':
    import matplotlib.pyplot as plt
    P1 = Polygon([Point(250,600), Point(400,600), Point(400,550), Point(350,550), Point(350,500),
    ↪ Point(400,500), Point(500,500), Point(500,550), Point(450,550), Point(450,650), Point(150,700),
    ↪ Point(100,750), Point(150,900), Point(300,850), Point(500,700), Point(450,850), Point(750,800),
    ↪ Point(800,700), Point(800,500), Point(600,500), Point(550,350), Point(700,350), Point(650,150),
    ↪ Point(600,150), Point(550,200), Point(250,300), Point(100,400), Point(100,450)])
    P = Polygon([Point(0,0), Point(0,1), Point(1,1), Point(1,0)])
    croix = Polygon([Point(300,450), Point(300,550), Point(400,550), Point(400,650), Point(500,650),
    ↪ Point(500,550), Point(600,550), Point(600,450), Point(500,450), Point(500,350), Point(400,350),
    ↪ Point(400,450)])
    croix=P1
    croix.write = True
```

Décomposition étoilée XV

```
from affichage import dbg
plt.imshow(dbg(croix))
plt.show()
plt.imshow(decomposition_etoile_keil(croix).debug())
plt.show()
```

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Polygones I

```
from __future__ import annotations

from typing import Generator
"""
Ici on implémente les structures de données fondamentales à savoir
- les points
- les polygones
"""

from math import sqrt

number = (float | int)

# points ou vecteurs
class Point:
    def __init__(self, x : number, y : number) -> None:
        self.x : number = x
        self.y : number = y

    def __eq__(self, __o: object) -> bool:
        if __o != None:
            return self.x == __o.x and self.y == __o.y
        else:
            return False

    def __hash__(self) -> int:
        return hash((self.x, self.y))

    def __str__(self) -> None:
        return f"Point({self.x},{self.y})"
```

Polygones II

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
def __repr__(self) -> str:
    return str(self)

"""On munit  $\mathbb{R}^2$  de la structure de  $\mathbb{R}$ -espace vectoriel"""

def __add__(self, other) -> Point:
    return Point(self.x + other.x, self.y + other.y)

def __sub__(self, other) -> Point:
    return self.__add__(-1 * other)

def __rmul__(self, scalaire) -> Point:
    return Point(scalaire * self.x, scalaire * self.y)

@property
def norme(self) -> float:
    return sqrt(self.x**2 + self.y**2)

def scalaire(self, other : Point) -> float:
    return self.x * other.x + self.y * other.y

def copie(self) -> Point:
    return Point(self.x, self.y)

def __le__(self, __o : Point) -> bool:
    if self.x == __o.x:
        return self.y <= __o.y
    else:
        return self.x <= __o.x

def det(vec1: Point, vec2: Point) -> float:
    """Determinant dans le plan  $\mathbb{R}^2$ """
    return vec1.x * vec2.y - vec2.x * vec1.y
```

Polygones III

Vecteur = Point

class Segment:

```
def __init__(self, p1 :Point, p2 :Point) -> None:
    self.p1 = p1
    self.p2 = p2

def __eq__(self, s: Segment) -> bool:
    return ((s.p1 == self.p1 and self.p2 == s.p2)
            or (s.p2 == self.p1 and s.p1 == self.p2))
def __repr__(self) -> str:
    return f"|({self.p1.x}, {self.p1.y}), ({self.p2.x}, {self.p2.y})|"

def __str__(self) -> str:
    return repr(self)

def __le__(self, __o : Segment) -> bool:
    if self.__eq__(__o):
        return True
    else:
        self.p1 <= __o.p2

def __hash__(self) -> int:
    return hash((self.p1, self.p2))

def __lt__(self, __o: Segment) -> bool:
    return self.__le__(__o) and (not self.__eq__(__o))

def intersecte(self, __o: Segment) -> bool:
    """renvoie vrai si le segment __o intersecte self"""
    ab = self.p2 - self.p1
    cd = __o.p2 - __o.p1
    #colinéarité
    d = det(ab, cd)
```

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Polygones IV

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
if d == 0:
    #On vérifie que les segments sont dans la meme
    #espace droite affine
    if det(self.p1 - __o.p1, ab) == 0:
        #si le segment __o intersecte self alors au moins un des points
        #est dans self, ce qui se traduit par
        return (self.p1 - __o.p1).scalaire(self.p2 - __o.p1) < 0 or (self.p1 - __o.p2).scalaire(self.p2
        ↪ - __o.p2) < 0
    else:
        return False

#detail calcul photo 13 mai
ac = __o.p1 - self.p1
t1 = (ac.x * cd.y - cd.x * ac.y)/d
t2 = (ab.y * ac.x - ab.x * ac.y)/d

#si les deux segments ne partagent aucun point en commun
if (self.p1 not in (__o.p1, __o.p2)) and (self.p2 not in (__o.p1, __o.p2)):
    return (0 <= t1 <= 1) and (0 <= t2 <= 1)
else:
    return (0 < t1 < 1) and (0 < t2 < 1)

def intersection(self, __o: Segment) -> (Point | None):
    """renvoie l'intersection de self et __o si le segment __o intersecte self"""
    a,b,c,d = self.p1, self.p2, __o.p1, __o.p2

    ab = b-a
    cd = d-c
    #colinéarité
    D = det(ab, cd)
    if D == 0:
        return None

    ac = __o.p1 - self.p1
```

Polygones V

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
t1 = (ac.x * cd.y - cd.x * ac.y)/D
t2 = (ac.x * ab.y - ab.x * ac.y)/D

#si les deux segments ne partagent aucun point en commun
if not ((a in (c,d)) or (b in (c, d))):
    if (0 <= t1 <= 1) and (0 <= t2 <= 1):
        return a + t1 * ab
    elif (0 < t1 < 1) and (0 < t2 < 1):
        return a + t1 * ab

def contient(self, c : Point) -> bool:
    a,b = self.p1, self.p2
    return det(b-a, c-a) == 0 and (a-c).scalaire(b-c) <= 0

def copie(self) -> Segment:
    return Segment(self.p1.copie(), self.p2.copie())

class Polygone:
    def __init__(self, points: list[Point], fill=None, outline="black") -> None:
        self.points : list[Point] = points
        self.fill = fill
        self.outline = outline
        self.write = False

    def __len__(self) -> int:
        return len(self.points)

    def __str__(self) -> str:
        return f"<{'', ' '.join(map(str, self.points))}>"

    def __repr__(self) -> str:
        return str(self)
```

Polygones VI

```
def copie(self) -> Polygone:
    return Polygone(
        list(map(lambda p: p.copie(), self.points)),
        fill=self.fill,
        outline=self.outline,
    )

def __eq__(self, __o: Polygone) -> bool:
    return self.points == __o.points

def est_entaille(self, index : int) -> bool:
    """Renvoie un booléen signifiant si le sommet d'indice index est une entaille"""
    """identité de polarisation"""
    prev = self[index-1]
    p = self[index]
    fol = self[index+1]
    segments = prev - p, fol - p
    return det(*segments) < 0

def est_convexe(self, index : int) -> bool:
    """Renvoie un booléen signifiant si le sommet d'indice index est convexe"""
    return not self.est_entaille(index)

@property
def segments(self) -> Generator[Segment]:
    for i in range(len(self)):
        yield Segment(self[i], self[i+1])

def __getitem__(self, index):
    return self.points[index%len(self)]
```

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Polygones VII

```
class Triangle(Polygone):
    pass

if __name__ == "__main__":
    """test module"""
    carre = Polygone([Point(1, 1), Point(0, 2), Point(1, 3), Point(2,2)])
    print(carre)
    print(carre.est_convexe())
```

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

Affichage I

```
"""
ici on implémente une 'vue' qui permet d'afficher des objets géométriques
"""
from polygone import Point, Polygone, Segment
from math import ceil, atan2, pi, cos, sin
import PIL.ImageDraw as ImageDraw
import PIL.Image as Image
from PIL.ImageOps import flip

from PIL import ImageFont

RÉSOLUTION = 500

X = lambda p: p.x
Y = lambda p: p.y
coord = lambda p : (X(p), Y(p))

inf = float("inf")

def cercle(draw, centre, rayon, outline):
    x, y = X(centre), Y(centre)
    draw.ellipse([(x - rayon, y - rayon), (x + rayon, y + rayon)], fill=outline)

def fleche(draw, ptA, ptB, width=1, color=(0,255,0)):
    draw.line((ptA,ptB), width=width, fill=color)
    x0, y0 = ptA
    x1, y1 = ptB

    xb = 0.95*(x1-x0)+x0
    yb = 0.95*(y1-y0)+y0

    # ligne verticale?
    if x0==x1:
```

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
        vtx0 = (xb-5, yb)
        vtx1 = (xb+5, yb)
# ligne horizontale?
    elif y0==y1:
        vtx0 = (xb, yb+5)
        vtx1 = (xb, yb-5)
    else:
        alpha = atan2(y1-y0,x1-x0)-90*pi/180
        a      = 20*cos(alpha)
        b      = 20*sin(alpha)
        vtx0   = (xb+a, yb+b)
        vtx1   = (xb-a, yb-b)

draw.polygon([vtx0, vtx1, ptB], fill=color)

class Vue:
    def __init__(self, res=None) -> None:
        self.polygones: list[Polygone] = list()
        self.segments: list[Segment] = list()
        self.points = list()

        self.xmin = Point(-inf, 0)
        self.xmax = Point(inf, 0)
        self.ymax = Point(0, -inf)
        self.ymin = Point(0, inf)
        if res != None:
            self.res = res
        else:
            self.res = RÉSOLUTION

    def afficher(self) -> Image:
        assert self.polygones != 0, "Il faut au moins un polygone pour dessiner une vue"

        DX = self.xmax.x - self.xmin.x
        DY = self.ymax.y - self.ymin.y
```

Affichage III

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
delta = max(DX, DY)
échelle = self.res / delta

f_marge = 0.015
marge = f_marge * self.res

echelliser = lambda p: Point(
    marge + échelle * ((p.x - self.xmin.x)*(1 - 2*f_marge)),
    marge + échelle * ((p.y - self.ymin.y)*(1 - 2*f_marge)),
)

obtenir_points_poly = lambda polygone: tuple(map(echelliser, polygone.points))
obtenir_points_segment = lambda s: Segment(echelliser(s.p1), echelliser(s.p2))

image = None
W, H = None, None
if DX == delta:
    W, H = self.res + ceil(0.1 * échelle), int(DY*échelle) + ceil(0.1 * échelle)
else:
    W, H = int(DX*échelle) + ceil(0.1 * échelle), self.res + ceil(0.1 * échelle)

image = Image.new("RGB", (W,H), "#f9f9f9")

draw = ImageDraw.Draw(image)

for couleur, slist in self.segments:
    for s in map(obtenir_points_segment, slist):
        draw.line(((s.p1.x, s.p1.y), (s.p2.x, s.p2.y)), fill=couleur, width=1)

for couleur, polygone in self.polygones:
    # segments
    points_dessin = obtenir_points_poly(polygone)
    draw.polygon(
        tuple(map(coord, points_dessin)),
```

Affichage IV

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
        fill=None,
        outline=couleur,
        width=3,
    )
    if hasattr(polygone, "ajouterfleche"):
        if polygone.ajouterfleche:
            for i in range(len(polygone)):
                fleche(polygone[i], polygone[i+1])
    # sommets
    cercle(draw, points_dessin[0], 0.012 * self.res, "green")

    for p in points_dessin[1:-1]:
        cercle(draw, p, 0.01 * self.res, polygone.outline)
    cercle(draw, points_dessin[-1], 0.012 * self.res, "red")
    #rajout de points arbitraire

    for couleur, plist, _ in self.points:
        for p in map(echelliser, plist):
            cercle(draw, p, 0.006 * self.res, couleur)
    image = flip(image)
    foreground = Image.new("RGBA", (W, H))

    d2 = ImageDraw.Draw(foreground)
    for couleur, polygone in self.polygones:
        if polygone.write:
            for i in range(len(points_dessin)):
                p = points_dessin[i]
                fnt = ImageFont.truetype(
                    "/usr/share/fonts/nerd-fonts-complete/TTF/Iosevka Term Nerd Font Complete.ttf",
                    ↪ 30)
                d2.text((p.x, H - p.y - 15), str(i), font=fnt,
                    fill=(0, 0, 0, 255), )

    for couleur, plist, write in self.points:
        for i in range(len(plist)):
```

Borne sup solution
optimalesDécomposition en
polygones convexesDécomposition en
polygones étoilésApplication à la ville
de Paris

Références

Annexe

```
p = echelliser(plist[i])
fnt = ImageFont.truetype(
    "/usr/share/fonts/nerd-fonts-complete/TTF/Iosevka Term Nerd Font Complete.ttf", 30)
if write:
    d2.text((p.x, H - p.y - 15), str(i), font=fnt,
            fill=couleur, )
image.paste(foreground, (0, 0), foreground)

return image#.resize((W, H), resample=Image.ANTIALIAS)

def ajouterPoints(self, points: list[Point], couleur="orange", write=False) -> None:

    self.points.append((couleur, points, write))

def clearPoints(self) -> None:
    self.points = list()
    self.afficher().show()

def ajouterPolygone(self, polygone: Polygone, couleur="black") -> None:
    # on recalcule les points extremaux
    self.xmin, self.xmax = (
        min(polygone.points + [self.xmin], key=X),
        max(polygone.points + [self.xmax], key=X),
    )
    self.ymin, self.ymax = (
        min(polygone.points + [self.ymin], key=Y),
        max(polygone.points + [self.ymax], key=Y),
    )
    self.polygones.append((couleur, polygone))

def ajouterSegments(self, segments: list[Segment], couleur = "blue") -> None:
    #recalculer points extremaux
    def update(p: Point):
        if p.x < self.xmin.x:
            self.xmin = p
```

Affichage VI

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe

```
        if p.y < self.ymin.y:
            self.ymin = p
        if p.x > self.xmax.x:
            self.xmax = p
        if p.y > self.ymax.y:
            self.ymax = p
    for s in segments:
        update(s.p1)
        update(s.p2)
    self.segments.append((couleur, segments))

def debug(P : Polygone) -> Vue:
    V = Vue()
    V.ajouterPolygone(P)
    V.afficher().show()
    return V

def dbg(P: Polygone) -> Image:
    V = Vue(res=1000)
    V.ajouterPolygone(P)
    return V.afficher()

if __name__ == "__main__":
    """test du module"""
    carre = Polygone([Point(1, 1), Point(1, 2), Point(2, 2), Point(2, 1)])
    carre = Polygone([Point(300, 450), Point(300, 550), Point(400, 550), Point(400, 650), Point(500,
↵ 650), Point(
    500, 550), Point(600, 550), Point(600, 450), Point(500, 450), Point(500, 350), Point(400, 350),
↵ Point(400, 450)])
    carre.write = True
    carre.couleur = 128
    vue = Vue()
    # print(vue.polygones)
    vue.ajouterPolygone(carre)
```

Affichage VII

```
vue.afficher().show()
```

Couverture
vidéo-surveillance

Candidat n° 26655

Borne sup solution
optimales

Décomposition en
polygones convexes

Décomposition en
polygones étoilés

Application à la ville
de Paris

Références

Annexe