Parallel Preprocessing for the Optimal Camera Placement Problem

Mathieu Brévilliers, Julien Lepagnot, Julien Kritter, and Lhassane Idoumghar

IRIMAS, University of Haute-Alsace, France

ICSMO 2018 — 6th International Conference on System Modeling and Optimization

University of Valenciennes, France — 7-11 February 2018

Outline

- Problem modeling
- Preprocessing and parallel approach
- Optimization with set-based DE

PROBLEM MODELING

Introduction

Optimal Camera Placement Problem

- Why?
  - Need to monitor areas of interest
- Constraints?
  - Requested quality of service
  - With a minimal cost
- Both constraints are related to the camera placement

Introduction

Optimal Camera Placement Problem

- Why?
  – Need to monitor areas of interest

- Constraints?
  – Requested quality of service
  – With a minimal cost

- Both constraints are related to the camera placement

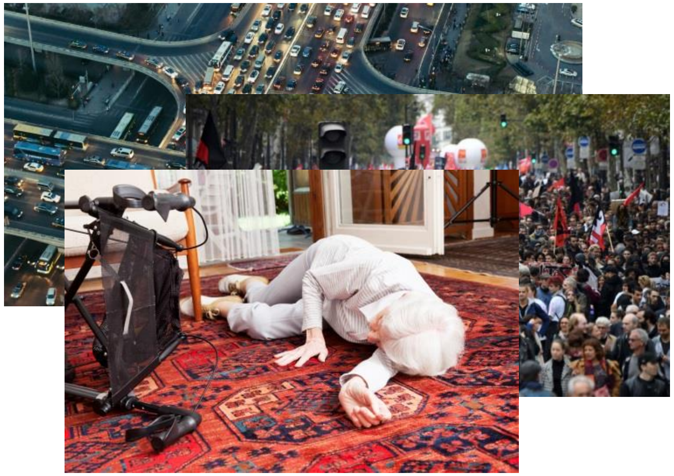
Introduction

Optimal Camera Placement Problem

- Why?
  - Need to monitor areas of interest
- Constraints?
  - Requested quality of service
  - With a minimal cost
- Both constraints are related to the camera placement

Introduction

Optimal Camera Placement Problem



- Why?
  - Need to monitor areas of interest
- Constraints?
  - Requested quality of service
  - With a minimal cost
- Both constraints are related to the camera placement

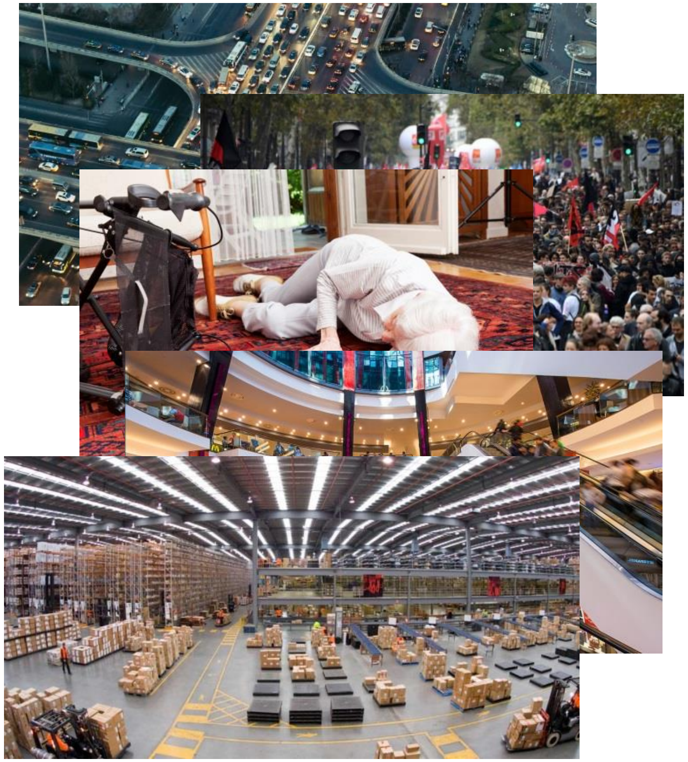
Introduction

Optimal Camera Placement Problem

- Why?
  - Need to monitor areas of interest
- Constraints?
  - Requested quality of service
  - With a minimal cost
- Both constraints are related to the camera placement

Introduction

Optimal Camera Placement Problem



- Why?
  - Need to monitor areas of interest
- Constraints?
  - Requested quality of service
  - With a minimal cost
- Both constraints are related to the camera placement

USCP formulation

Unicost Set Covering Problem

- Decision problem in a discrete search space:
  - 3D monitored area discretized with a grid of points
  - Finite set of feasible camera locations
- Given:
  - the set E of points to be covered
  - and the set S of possible camera locations,
- Find the minimal subset of S that covers E.

USCP formulation

$$\forall c \epsilon S, x_c = \begin{cases} 1 & \text{if camera location } c \text{ is used,} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

$$Min \sum_{c \epsilon S} x_c \tag{2}$$

$$\forall p \epsilon E, \sum_{c \epsilon S : p \epsilon c} x_c \geq 1 \tag{3}$$

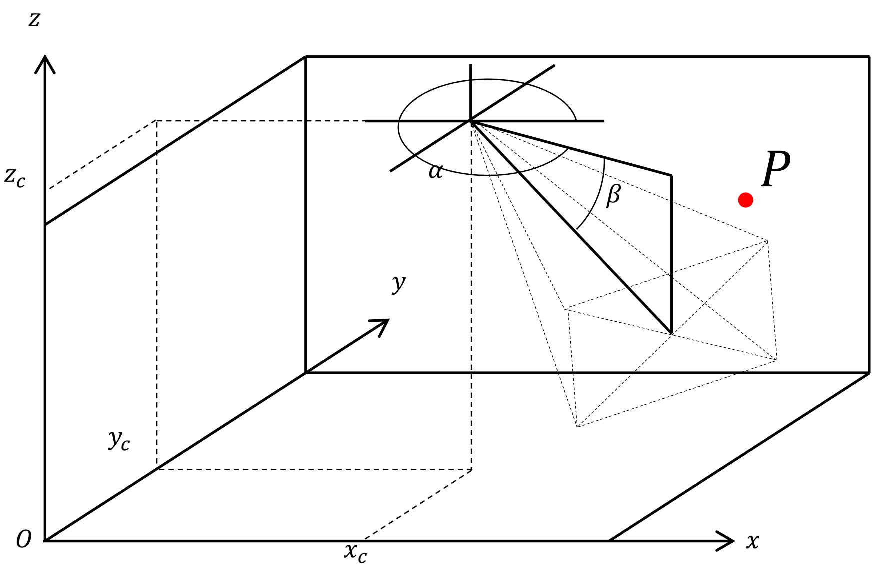
$$\forall c \epsilon S, x_c \epsilon \{0,1\} \tag{4}$$

PREPROCESSING AND PARALLEL APPROACH

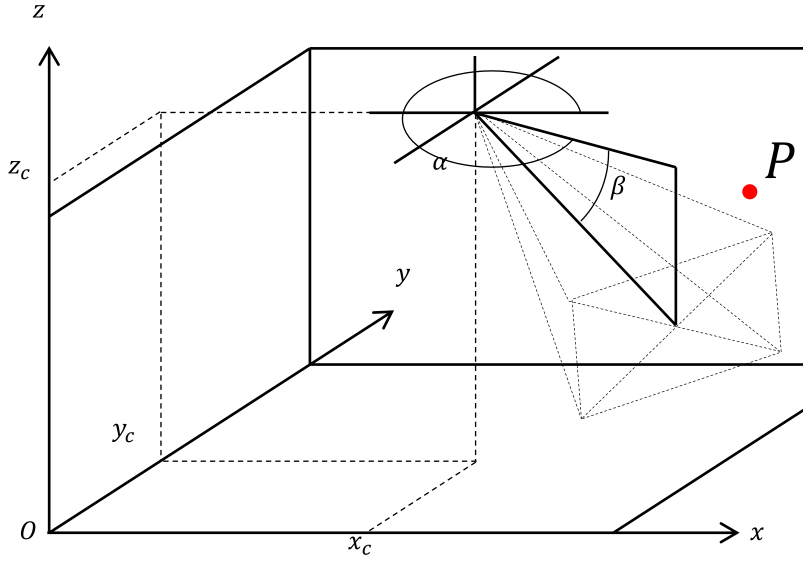
Introduction

- Aim:
  - Full coverage information
  - Minimum input data
- Visibility preprocessing:
  - What are the points covered by each camera location?
- Reduction preprocessing:
  - Which camera locations are useless?
- Main issue: high computational cost for large problems

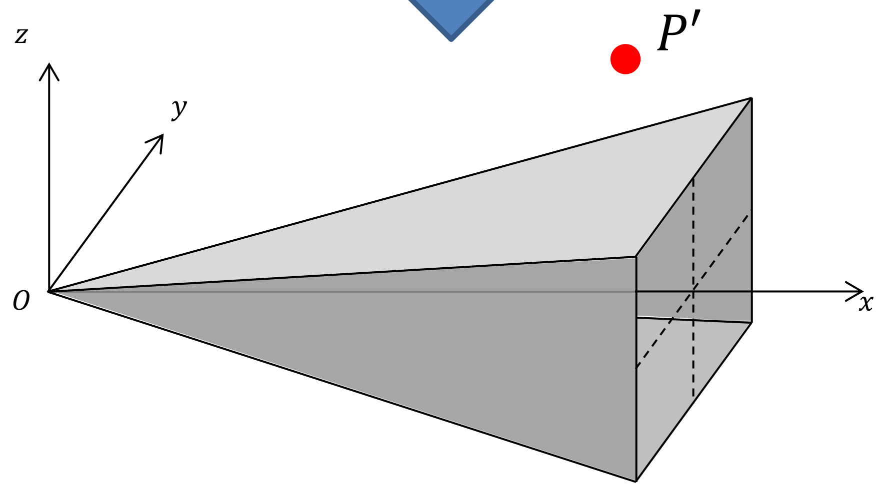
Visibility test



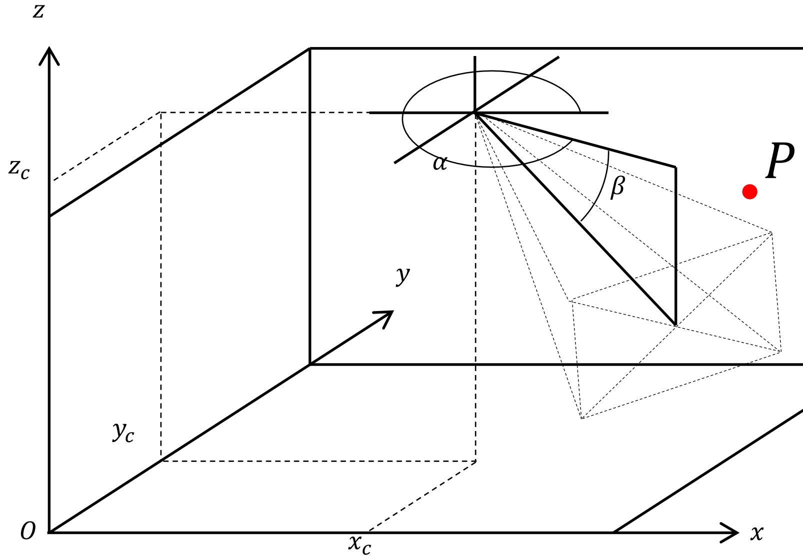
Visibility test



$$P' = PTR_ZR_Y$$

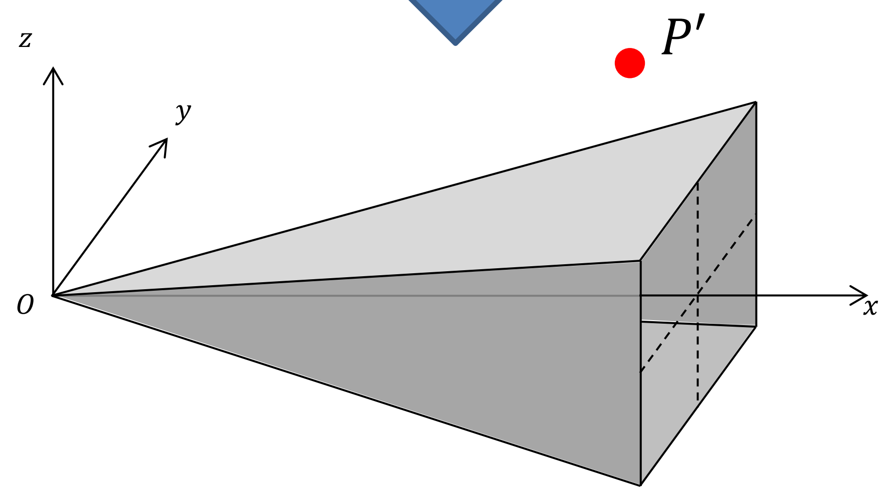


Visibility test

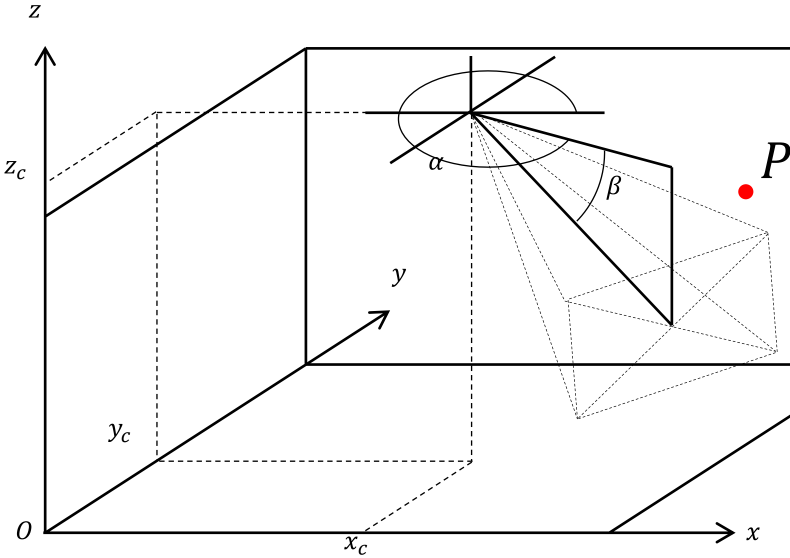


$$P' = PTR_ZR_Y$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_c & -y_c & -z_c & 1 \end{bmatrix}$$

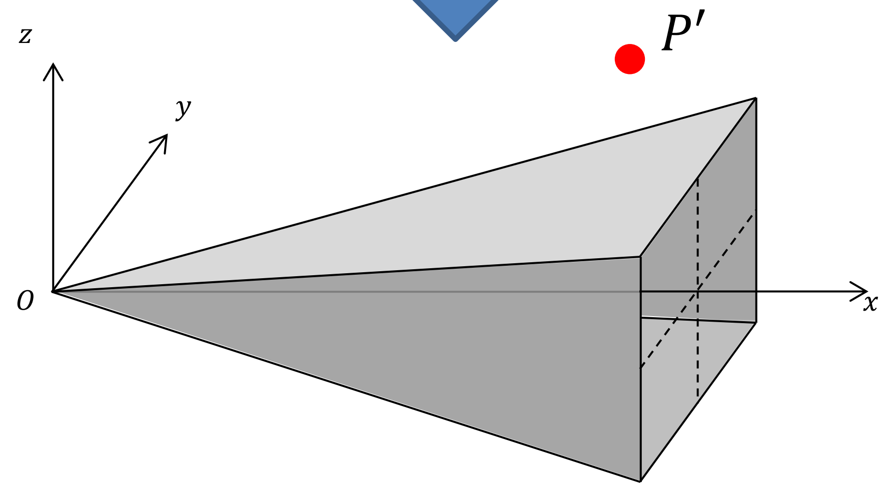


Visibility test

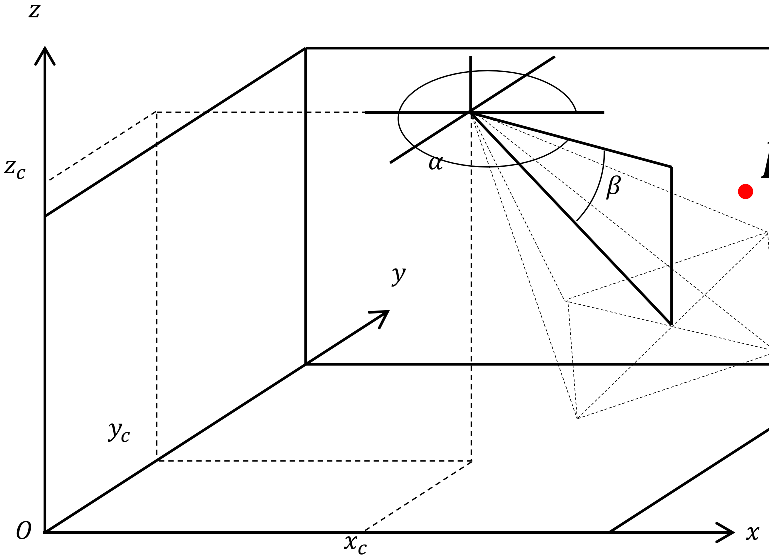


$$P' = P T R_Z R_Y$$

$$R_Z = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

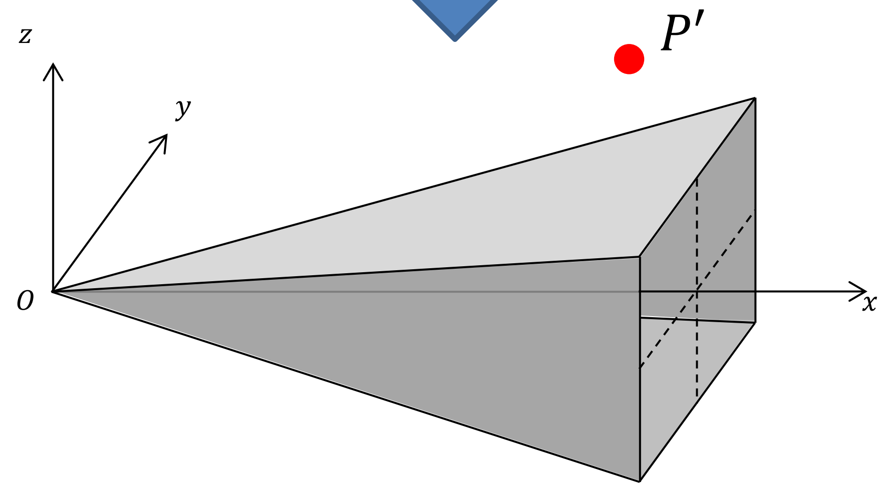


Visibility test

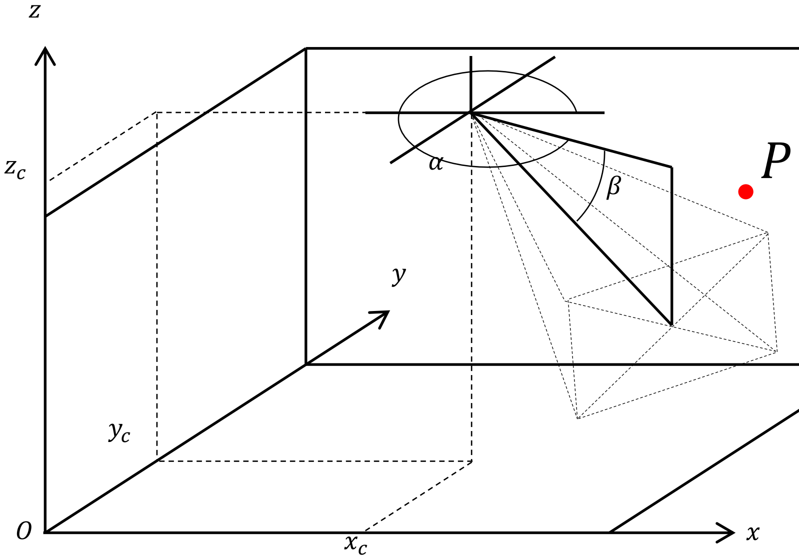


$$P' = PTR_ZR_Y$$

$$R_Y = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Visibility test



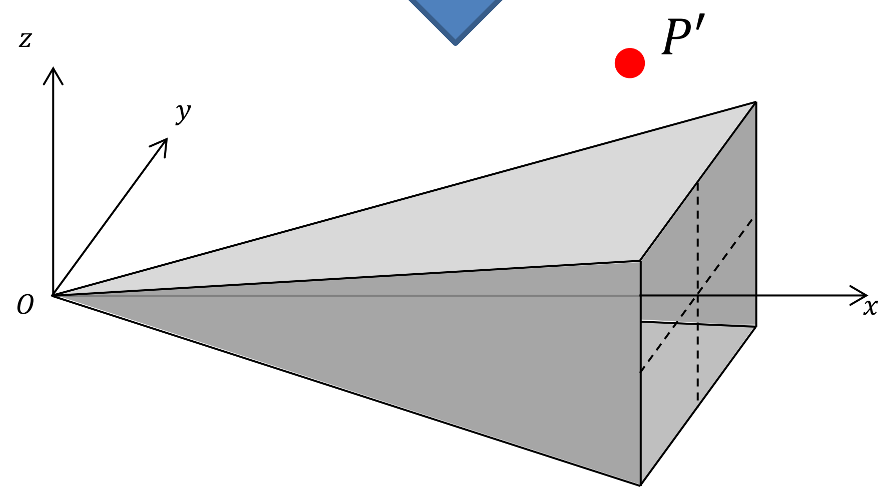
$$P' = PTR_ZR_Y$$

$$R_Y = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$0 \leq x' \leq D$$

$$|y'| \leq \frac{w}{2} \times \frac{x'}{D}$$

$$|z'| \leq \frac{h}{2} \times \frac{x'}{D}$$



Sequential visibility preprocessing

Input: The set of possible camera locations.
The set of points to be covered.
Output: The coverage information $cov(c)$ of each camera location c.

For each camera location c **do**
  For each point p **do**
    Compute new coordinates of p.
    If c covers p
      Add p in $cov(c)$.
    End if
  End for
End for

Sequential visibility preprocessing

Input: The set of possible camera locations.
The set of points to be covered.
Output: The coverage information $cov(c)$ of each camera location c.

For each camera location c **do**
  For each point p **do**
    Compute new coordinates of p.
    If c covers p
      Add p in $cov(c)$.
    End if
  End for
End for

Sequential visibility preprocessing

Input: The set of possible camera locations.
The set of points to be covered.

Output: The coverage information $cov(c)$ of each camera location c.

For each camera location c **do**
  For each point p **do**
    Compute new coordinates of p.
    If c covers p
      Add p in $cov(c)$.
    End if
  End for
End for

GPU

Parallel visibility preprocessing

Input: The set of possible camera locations.
The set of points to be covered.
Output: The coverage information $cov(c)$ of each camera location c.

```
Call the GPU kernel that performs the visibility preprocessing
and stores the results in an integer matrix M.
Copy M from GPU global memory to CPU main memory.
For each point p do
  For each camera location c listed in line of p in M do
    Add p in cov(c).
  End for
End for
```

Parallel visibility preprocessing

Input: The set of possible camera locations.
The set of points to be covered.
Output: The coverage information $cov(c)$ of each camera location c.

Call the GPU kernel that performs the visibility preprocessing and stores the results in an integer matrix M.
Copy M from GPU global memory to CPU main memory.
For each point p **do**
  For each camera location c listed in line of p in M **do**
    Add p in $cov(c)$.
  End for
End for

Up to a 15-time speedup

Reduction preprocessing

Given two camera locations c and c',

if $cov(c) \subseteq cov(c')$, then c is dominated by c'

thus c can be removed from the set of feasible camera locations

Reduction preprocessing

Input: All camera locations c and their $cov(c)$.
Output: The subset S of non-dominated camera locations.

Compute the list T of all $cov(c)$ sorted by increasing order of their size.
For each $cov(c)$ in T **do**
For each $cov(c')$ after $cov(c)$ in T **do**
If $cov(c) \subseteq cov(c')$
Mark $cov(c)$ as dominated
Break for loop
End if
End for
End for
For each $cov(c)$ in T **do**
If $cov(c)$ is not dominated
Add c in S
End if
End for

Reduction preprocessing

Input: All camera locations c and their $cov(c)$.
Output: The subset S of non-dominated camera locations.

Compute the list T of all $cov(c)$ sorted by increasing order of their size.
For each $cov(c)$ in T **do**
  For each $cov(c')$ after $cov(c)$ in T **do**
    If $cov(c) \subseteq cov(c')$
      Mark $cov(c)$ as dominated
      Break for loop
    End if
  End for
End for
For each $cov(c)$ in T **do**
  If $cov(c)$ is not dominated
    Add c in S
  End if
End for

Reduction preprocessing

Input: All camera locations c and their $cov(c)$.
Output: The subset S of non-dominated camera locations.

Compute the list T of all $cov(c)$ sorted by increasing order of their size.
For each $cov(c)$ in T **do** → **Distributed to a cluster of CPU**
  For each $cov(c')$ after $cov(c)$ in T **do**
    If $cov(c) \subseteq cov(c')$
      Mark $cov(c)$ as dominated
      Break for loop
    End if
  End for
End for
For each $cov(c)$ in T **do**
  If $cov(c)$ is not dominated
    Add c in S
  End if
End for

Reduction preprocessing

Input: All camera locations c and their $cov(c)$.

Output: The subset S of non-dominated camera locations.

Compute the list T of all $cov(c)$ sorted by increasing order of their size.
For each $cov(c)$ in T **do**
  For each $cov(c')$ after $cov(c)$ in T **do**
    If $cov(c) \subseteq cov(c')$
      Mark $cov(c)$ as dominated
      Break for loop
    End if
  End for
End for
For each $cov(c)$ in T **do**
  If $cov(c)$ is not dominated
    Add c in S
  End if
End for

Distributed to a cluster of CPU

Up to a 6-time speedup with 8 computers

General parallel approach

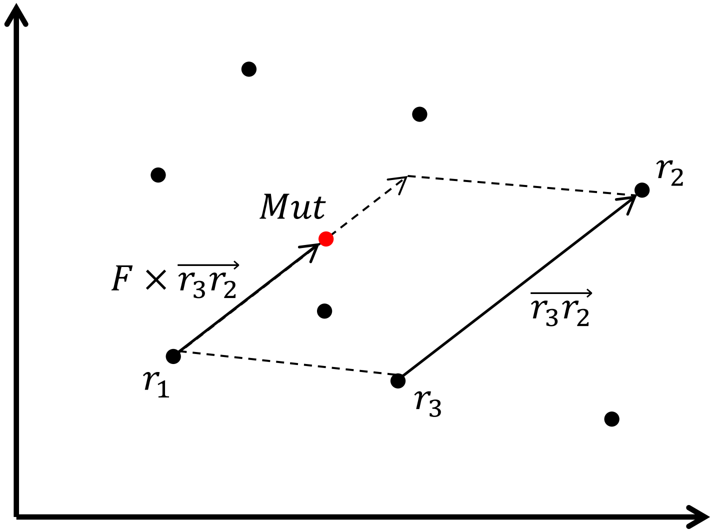
1. All nodes of the cluster perform the whole visibility preprocessing with the GPU.
2. The reduction preprocessing is then distributed to all nodes.
3. The master node:
   - aggregates all results,
   - and provides the final set of non-dominated camera locations and their coverage information.

OPTIMIZATION WITH SET-BASED DE

DE for continuous optimization

- Population-based evolutionary algorithm:

1. Mutation: $Mut_{i,j} = Pop_{r_1,j} + F \times \left(Pop_{r_2,j} - Pop_{r_3,j}\right)$
2. Crossover
3. Selection



Set-based DE

- Adaptation to combinatorial optimization:

1. Mutation: $Mut_i = Sol_{rand} \cup F \cdot \left(Pop_{r_1} \oplus Pop_{r_2}\right)$

2. Crossover:

   Solve the (much smaller) subproblem where the set of feasible camera locations is $Pop_i \cup Mut_i$

3. Selection

Experimental setting

- Compared algorithms (1000s max runtime)
  - CPLEX
  - Greedy
  - Set-based DE hybridized with CPLEX
  - Set-based DE hybridized with Greedy
- 10 Problem instances
  - Grid size from 10*10*4 to 50*50*4 units of length
  - 2 classes with max depth of view 10 or 20 units

Results

Instance	CPLEX	Greedy	DEset-CPLEX	Deset-Greedy
1	**7**	10	**7,00**	7,87
2	**21**	32	21,53	29,10
3	56	63	**48,07**	71,60
4	29 549	109	**92,80**	131,13
5	46 907	164	**155,80**	202,73
6	**7**	9	**7,00**	**7,00**
7	**5**	**5**	**5,00**	**5,00**
8	**9**	12	**9,00**	9,37
9	**14**	19	14,97	18,90
10	26	30	**23,43**	32,97

Conclusion

- Optimal camera placement problem
  - stated as a USCP
- Parallel preprocessing
  - Visibility on GPU = 15x speedup
  - Distributed reduction = 6x speedup
- Optimization
  - Comparison of CPLEX, Greedy and set-based DE
  - Promising results for set-based DE

Conclusion

- Future work
  - Sometimes visibility test is oubviously useless
  - Accelerate reduction by using neighborhood
  - Impact of set-based DE parameters
  - Comparison with other relevant algorithms

Parallel Preprocessing for the Optimal Camera Placement Problem

Mathieu Brévilliers, Julien Lepagnot, Julien Kritter, and Lhassane Idoumghar
IRIMAS, University of Haute-Alsace, France

ICSMO 2018 — 6th International Conference on System Modeling and Optimization
University of Valenciennes, France — 7-11 February 2018

Appendix: problem instances

Instance	X_a	Y_a	Z_a	Z_c	H	D	A
1	10	10	4	5	65	10	4
2	20	20	4	5	65	10	4
3	30	30	4	5	65	10	4
4	40	40	4	5	65	10	4
5	50	50	4	5	65	10	4
6	10	10	4	5	65	20	4
7	20	20	4	5	65	20	4
8	30	30	4	5	65	20	4
9	40	40	4	5	65	20	4
10	50	50	4	5	65	20	4

Appendix: visibility preprocessing

Instance	$\|E\|$	$\|S\|$	CPU	GPUv1	GPUv2
1	605	2 904	0.236	0.078 (3.03)	0.037 (6.38)
2	2 205	10 584	2.912	0.623 (4.67)	0.250 (11.65)
3	4 805	23 064	13.701	2.825 (4.85)	0.988 (13.87)
4	8 405	40 344	41.742	8.568 (4.87)	2.725 (15.32)
5	13 005	62 424	99.780	20.377 (4.90)	6.287 (15.87)
6	605	2 904	0.228	0.051 (4.47)	0.035 (6.51)
7	2 205	10 584	2.920	0.637 (4.58)	0.271 (10.78)
8	4 805	23 064	13.765	2.892 (4.76)	1.057 (13.02)
9	8 405	40 344	41.952	8.762 (4.79)	2.902 (14.46)
10	13 005	62 424	100.268	20.687 (4.85)	6.583 (15.23)

Appendix: reduction preprocessing

Instance	$\|S\|$	$\|S'\|$	$n = 1$	$n = 2$	$n = 4$	$n = 6$	$n = 8$
1	2 904	1 401	0.078	0.045(1.73)	0.023(3.39)	0.016(4.88)	0.012(6.50)
2	10 584	6 778	0.989	0.540(1.83)	0.315(3.14)	0.185(5.35)	0.153(6.46)
3	23 064	16 180	5.313	2.813(1.89)	1.712(3.10)	0.969(5.48)	0.913(5.82)
4	40 344	29 549	19.989	10.591(1.89)	6.718(2.98)	3.595(5.56)	3.400(5.88)
5	62 424	46 907	54.332	27.996(1.94)	18.020(3.02)	9.379(5.79)	9.056(6.00)
6	2 904	1 292	0.080	0.046(1.74)	0.023(3.48)	0.016(5.00)	0.012(6.67)
7	10 584	2 179	1.823	0.947(1.93)	0.531(3.43)	0.325(5.61)	0.267(6.83)
8	23 064	7 889	10.656	5.507(1.93)	3.172(3.36)	1.951(5.46)	1.581(6.74)
9	40 344	16 864	35.199	18.213(1.93)	10.688(3.29)	6.169(5.71)	5.586(6.30)
10	62 424	29 071	87.244	44.647(1.95)	26.937(3.24)	14.941(5.84)	13.457(6.48)