

Projet d'optimisation



15 novembre 2018

DESLANDES Benoît
LACOMBE Arthur
SCHERRER Matthieu
STERLIN Dorian

Table des matières

Table des figures	2
1 Introduction et présentation du projet	3
2 Organisation du groupe et modification de la chaîne d'optimisation	3
2.1 Organisation et répartition du travail	3
2.2 Modification de la chaîne d'optimisation	5
3 Nouveaux modèles et algorithmes développés	5
3.1 Partie acquisition	5
3.1.1 Modification du fichier <i>BadAcquisitionPlanner</i>	6
3.1.2 Modification du code OPL	6
3.2 Partie déchargement d'images	7
3.2.1 Modification du fichier <i>BadDownloadPlanner</i>	8
3.2.2 Création du code <i>OPL</i> pour l'optimisation du download	9
3.3 Visualisation des résultats	10
3.3.1 Outil de visualisation des acquisitions effectuées	10
3.3.2 Qualité de la photo	10
3.3.3 Présence de nuages dans la photo	11
3.3.4 Gestion de la priorité dans les acquisitions	12
3.3.5 Validation des quotas par utilisateur	13
3.4 Automatisation du process à l'aide d'un script BASH	13
4 Analyse des résultats	14
4.1 Analyse des performances de l'acquisition	14
4.1.1 Analyse des algorithmes	14
4.1.2 Analyse des acquisitions sélectionnées	16
4.2 Analyse des performances du déchargement d'images	17
4.3 Améliorations possibles et pistes de réflexion supplémentaires	18

Table des figures

2.1	Organigramme du projet	4
2.2	Chaîne d'optimisation originale	5
2.3	Nouvelle chaîne d'optimisation	5
3.1	Passage d'une gestion indépendante des deux satellites à une constellation	6
3.2	Description des fichiers .dat nécessaires à l'optimisation du Download	8
3.3	Caption	10
3.4	Validation de la qualité dans notre critère	11
3.5	Validation de la probabilité de nuage dans notre critère	12
3.6	Validation de la probabilité de nuage dans notre critère	12
3.7	Diagramme des quotas par utilisateur	13
4.1	Évolution des contraintes et variables en fonction du nombre d'acquisition candidates	15
4.2	Visualisation des caractéristiques des acquisitions candidates avec le projet original sur le cas complexe 4h pendant 10 min	16
4.3	Visualisation des caractéristiques des acquisitions candidates avec le projet optimisé sur le cas complexe 4h	17
4.4	Visualisation du processus d'acquisition et déchargement avec le projet original	17
4.5	Visualisation du processus d'acquisition et déchargement avec le projet optimisé	18

1 Introduction et présentation du projet

Le projet d'optimisation spatiale consiste à optimiser la prise d'images d'une constellation composée de 2 satellites ainsi que le déchargement des images prises. Pour se faire, plusieurs méthodes et critères d'optimisation ont été explorés pour déterminer une solution pertinente du problème. Le problème étant néanmoins trop complexe, une solution parfait n'existe pas et dépend des critères que l'on impose.

2 Organisation du groupe et modification de la chaîne d'optimisation

2.1 Organisation et répartition du travail

La première partie de ce projet fut une compréhension approfondie du code existant ainsi que du sujet. Pour ce faire, nous avons décidé de le faire tous les quatre, afin que chacun puisse bien comprendre les attendus et les objectifs de ce projet. Une fois cette tâche accomplie, nous nous sommes répartis le travail en fonction de nos décisions et de notre avancement dans le projet.

Dans un deuxième temps, la modification du code JAVA initial et du modèle d'optimisation a été effectuée. Puis, nous avons considéré utile de vérifier que nos nouveaux critères d'optimisation fonctionnaient correctement. Pour cela, nous avons procédé au développement de différents outils afin de valider ou non notre modèle. Par la suite, le déchargement des données a été étudié, afin de l'optimiser. Finalement, nous avons analyser les résultats donnés par nos algorithmes.

Le tableau 2.1 et l'organigramme présenté à la figure 2.1 résument l'organisation du projet ainsi que la répartition du travail.

Tâches	Benoît	Arthur	Matthieu	Dorian
Compréhension du sujet et assimilation du code existant	✓	✓	✓	✓
Travail sur le code JAVA pour n'obtenir qu'un fichier DAT afin d'optimiser la constellation			✓	
Modification de l'optimisation des acquisitions (.mod) pour la constellation et ajout de nouveaux critères	✓	✓		✓
Validation de l'optimisation et développement d'outils pour visualiser les résultats		✓		✓
Optimisation du déchargement des données : création du code JAVA générant le DAT et du modèle OPL	✓		✓	
Analyse des résultats	✓	✓	✓	✓
Préparation de la soutenance orale et rédaction du présent rapport	✓	✓	✓	✓

TABLE 2.1 – Tableau récapitulatif de l'organisation du groupe et des différentes tâches

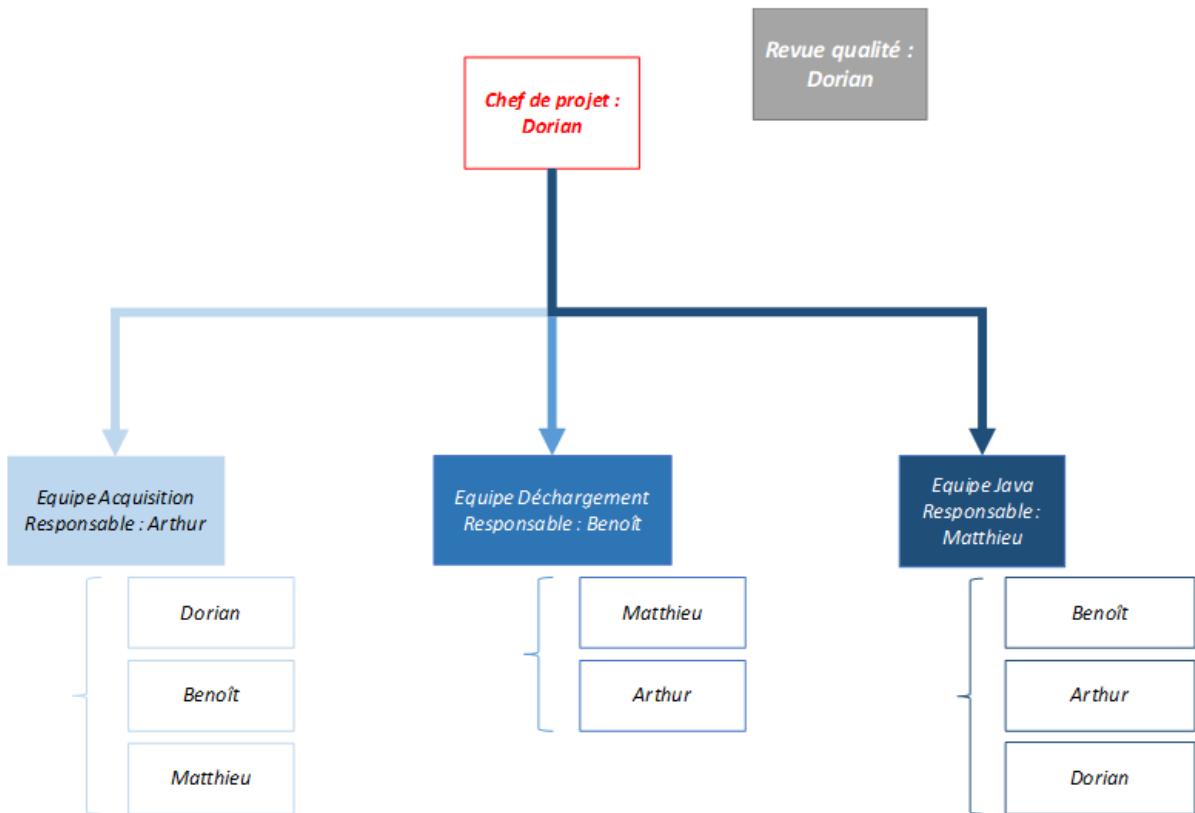


FIGURE 2.1 – Organigramme du projet

2.2 Modification de la chaîne d'optimisation

La chaîne d'optimisation originale est présentée en figure 2.2.

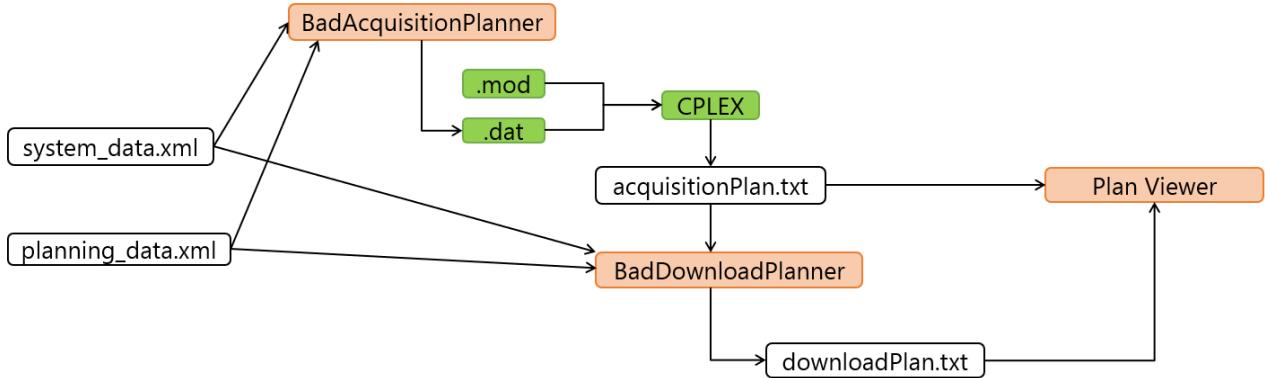


FIGURE 2.2 – Chaîne d'optimisation originale

Comme on peut le voir, la classe `BadDownloadPlanner` servait initialement de modèle d'optimisation dans la chaîne. Dans la nouvelle chaîne, la classe `DownloadPlanner` crée un fichier `.dat` qui utilise les solutions de l'optimisation de l'acquisition et les données des acquisitions. Un modèle `.mod` a été développé (voir section 3.2) et prend en entrée ce fichier `.dat`. En sortie, nous obtenons toujours le même format pour le fichier `downloadPlan.txt` qui est utilisé dans la classe `PlanViewer`.

De plus, un script bash a été développé afin d'automatiser tout le processus. En effet, ce projet requiert de copier plusieurs fichiers par cycle d'optimisation. En modifiant les différents algorithmes et modèles, nous devons donc effectuer un grand nombre de fois ces opérations. Pour être sur de ne pas oublier de copier/coller certains fichiers et ainsi diminuer les erreurs possibles, le script bash fait tourner toute la chaîne d'optimisation en une seule fois. Ses fonctionnalités détaillées sont présentées dans la section 3.4.

La nouvelle chaîne est présentée en figure 2.3.

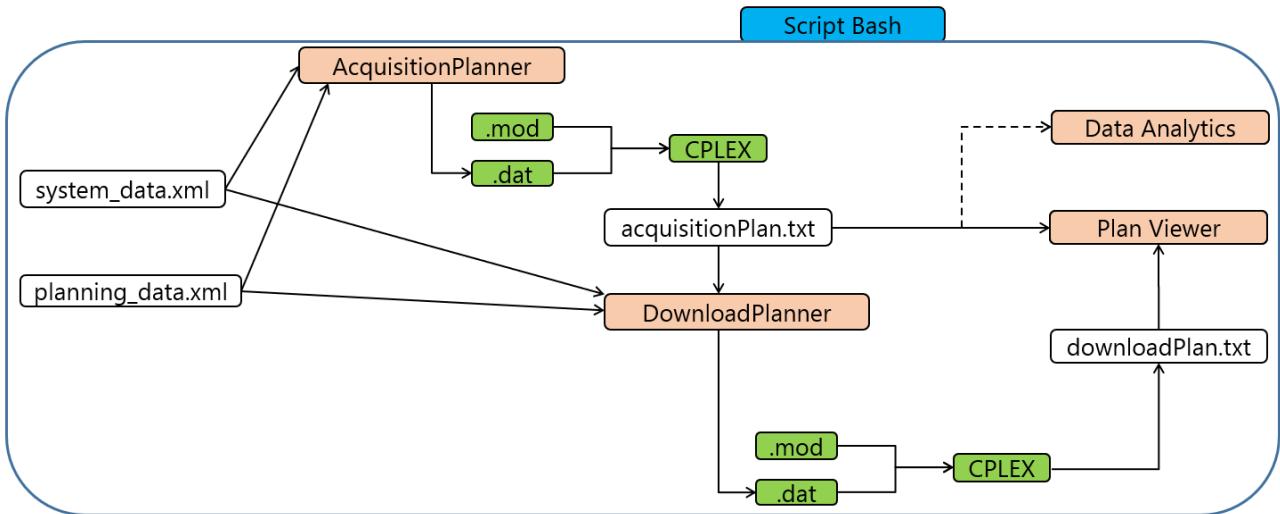


FIGURE 2.3 – Nouvelle chaîne d'optimisation

3 Nouveaux modèles et algorithmes développés

3.1 Partie acquisition

A l'origine, les fichiers et programmes étaient faits pour diviser le travail en deux parties indépendantes : le satellite 1 et le satellite 2. Après discussions avec l'ensemble de l'équipe nous avons convenu que le mieux était de procéder par une optimisation de la constellation dans son ensemble et non satellite par satellite. Cette

approche permet de contrôler plus facilement les éventuelles redondances d'images et optimiser la prise des images pour la constellation.

3.1.1 Modification du fichier *BadAcquisitionPlanner*

Le premier fichier que nous avons exploré par son utilisation est le fichier java *BadAcquisitionPlanner* qui, à partir des données XML du problème à traiter produisait 2 documents **.dat** qui étaient les plans d'acquisitions possibles du satellite 1 et du satellite 2.

— Création d'un fichier unique **.dat**

Pour pouvoir optimiser la constellation entière, nous avons décidé de modifier le code java pour obtenir un seul fichier **.dat** comportant à la fois les acquisitions possibles du satellite 1 et du satellite 2.

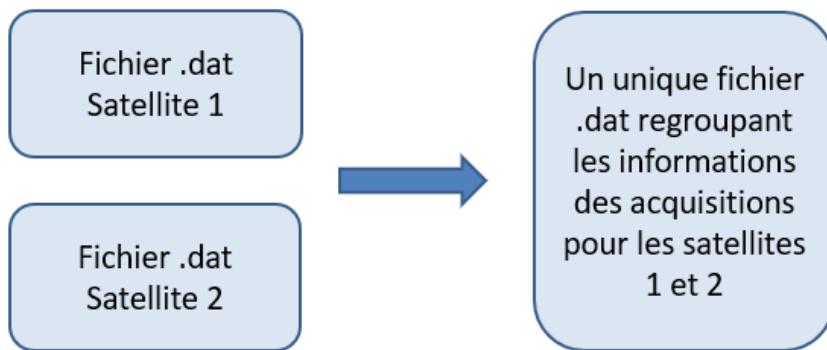


FIGURE 3.1 – Passage d'une gestion indépendante des deux satellites à une constellation

Pour cela, il a fallut modifier les caractéristiques des acquisitions candidates. Tout d'abord, étant donné qu'il n'y a alors qu'un seul fichier **.dat** pour les deux satellites, il devient nécessaire de préciser quel satellite fait quelle acquisition candidate. De plus, afin de s'offrir la possibilité d'avoir une fonction objective très variée à optimiser via l'utilisation de poids pour chaque paramètres, nous avons gardé toutes les informations sur les acquisitions candidates.

Les fichiers **.dat** d'origine n'offraient l'accès que aux dates de départ et de fin de l'acquisition, ainsi que à son ID. Nous avons alors modifié le fichier java afin que chaque acquisition soit également caractérisée par son volume, sa priorité, sa probabilité d'avoir des nuages, sa qualité, le satellite devant la faire ainsi que l'utilisateur pour laquelle elle est destinée.

— Ordonnancement des acquisitions possibles par ordre chronologique

Après quelques essais, on peut remarquer que l'on obtient de meilleurs résultats en réordonnant le fichier **.dat** par ordre chronologique du début au plus tôt de l'acquisition plutôt que par index. Il a donc été décidé dans la suite de conserver cette ordre chronologique qui n'affecte en rien le code du reste du projet.

— Élimination des acquisitions candidates parasites

Pour simplifier la résolution du problème, les acquisitions qui étaient jugées non faisables ou non satisfaisantes étaient éliminées dès la création du fichier **.dat**. Ainsi, lorsque la probabilité de nuage était de 0,7, l'acquisition candidate était jugée comme non recevable. De cette manière, dès la création du fichier **.dat**, 15 – 20% des acquisitions se retrouvent éliminées et le document **.dat** s'en sort fortement allégé.

3.1.2 Modification du code OPL

Le code OPL original produit 2 documents **.txt** comprenant les acquisitions choisies par l'algorithme. Pour fonctionner, un fichier **.dat** est nécessaire. Le code OPL existant donne une solution pour un seul satellite à la fois. Dans le but d'optimiser la constellation entière nous n'utiliserons donc plus qu'un seul fichier **.dat** d'entrée mais ressortiront tout de même 2 documents **.txt** dans le but de ne pas entraver le fonctionnement du *Plan Viewer*.

— Variables de décision

L'OPL original comprenait une variable de décision de type tableau de données binaires permettant de

savoir quelle acquisition candidate était sélectionnée ou non. Cette variable ce retrouve 2 fois dans notre algorithme (une pour le satellite 1 et une autre pour le satellite 2). Ces variables sont complétées par une expression composée de variables de décision qui est en fait la somme des deux tableaux précédents faisant ainsi valoir la sélection de l'acquisition au niveau de la constellation entière.

Les variables de décision concernant le temps de départ de l'acquisition et le tableau récapitulant les acquisitions qui se précédent ont été conservés mais nous avons joué sur ces derniers pour tester différents cas de figures. La matrice next étant bien évidemment très consommatrice en terme de calculs et rallonge le temps d'optimisation dans les cas complexes et simples avec de nombreuses acquisitions candidates.

Par exemple, la variable de décision concernant le temps de départ a été modifiée pour accélérer le temps de calcul. De façon grossière, nous avons fixé le temps de départ comme étant égal au temps de départ le plus tôt. Ceci constraint évidemment plus le problème, mais permet d'éviter l'optimisation sur le temps de départ qui est très coûteuse.

— Fonction objectif

La fonction objectif initiale qui était de maximiser le nombre d'images prises a été modifiée afin de mieux répondre aux besoins de nos clients et des exigences en terme de quota, priorité, qualité des images, probabilité de nuage. La fonction objectif optimal implémenté est une somme pondérée des différents facteurs : volume, angle de zénith, probabilité de nuage et priorité. Le plus gros coefficient est attribué au volume, le but étant d'obtenir le plus gros volume possible d'images de bonne qualité et avec un taux de nuages limité.

— Contraintes

Des contraintes supplémentaires ont été ajoutées à celles déjà existantes dans le projet. Ces dernières ont été adaptées pour pouvoir fonctionner dans le cas de l'optimisation de la constellation toute entière. Les contraintes finales sont :

1. *Non redondance des images prises pour la constellation*

Cette contrainte essentielle assure qu'un même satellite ne prenne pas deux fois la même image ET que l'image ne soit prise que par UN SEUL des satellites de la constellation.

2. *Quotas par utilisateur*

Cette contrainte a été rajoutée par rapport au projet initiale. Elle permet de faire respecter des quotas de volumes et d'images par utilisateur avec une marge de 10% pour ne pas trop contraindre le modèle. Ces quotas sont de 50% pour l'utilisateur 1, 30% pour l'utilisateur 2 et de 20% pour l'utilisateur 3.

3. *Séparation temporelle*

La contrainte temporelle permet d'éviter de choisir deux acquisitions incompatibles. Il s'agit ici d'assurer que l'acquisition de l'image 2 se fasse après la fin de l'acquisition de l'image 1 en prenant également en compte la durée de repositionnement du satellite pour prendre l'image 2. Cette contrainte est originellement spécifiée en utilisant la matrice next dans une formulation Big-M. Dans notre algorithme optimisé, nous substituons cette matrice en utilisant simplement la dvar selectAcq. Le problème est ainsi moins coûteux en temps de calcul et est résolu plus rapidement.

4. *Sélection de satellite*

Cette contrainte provient de notre ambition à optimiser la constellation de satellites d'un seul coup. Il faut donc être sur que l'acquisition choisie est faite par un satellite qui peut la faire.

— Modes de résolution

Au cours du projet, nous avons expérimenté 2 modes de résolution. Le mode simplex ainsi que le mode CP. Après analyse des résultats il a été préféré de conserver le mode de résolution simplex.

3.2 Partie déchargement d'images

Nous avons constaté que l'outil initial n'exécute aucune optimisation lors du vidage des données, il se contente de vider toutes les acquisitions en mémoire dès qu'il se trouve dans une fenêtre de déchargement, suivant une

règle "First In - First Out". Cette opération présente plusieurs défauts, dont un que nous avons jugé majeur : beaucoup de fenêtres de vidage sont utilisées, parfois pour ne vider que peu d'acquisitions. Ceci résulte d'une part en une sur-utilisation des actionneurs de roulis, et d'autre part en un surcoût de la mission, car la monopolisation d'une antenne au sol pour le vidage est très onéreuse.

Pour pallier ces défauts, nous avons donc décidé de créer un nouveau problème d'optimisation, visant à minimiser le nombre de fenêtres utilisées pour le déchargement des données. Pour ce faire, nous avons procédé en deux temps : nous avons en premier lieu modifié le projet Java afin de générer les fichiers dat, puis nous avons créé un nouveau projet OPL réalisant l'optimisation. Ce processus a ensuite été intégré au script bash, que nous détaillerons plus tard.

3.2.1 Modification du fichier *BadDownloadPlanner*

Pour les raisons données précédemment, nous avons entrepris de modifier le code du fichier Java "BadDownloadPlanner" afin de travailler sur le download des acquisitions qui ont été effectuées par le satellite. L'objectif a donc été de créer un fichier .dat afin de pouvoir relancer un nouveau projet Opl optimisant ce download des images.

— Création de deux fichiers .dat

L'optimisation du download des acquisitions doit se faire pour chaque satellite étant donné que pour cette tâche ils sont indépendants dans le cadre de ce projet. Pour cette raison, le fichier Java BadDownloadPlanner doit générer un fichier .dat pour chacun des satellites. Ces fichiers .dat doivent contenir toutes les informations nécessaires à l'optimisation souhaitée de download.



FIGURE 3.2 – Description des fichiers .dat nécessaires à l'optimisation du Download

Comme on le voit sur la figure précédente, le fichier .dat de chaque satellite se divise en deux composantes. La première composante consiste en la description de toutes les acquisitions qui ont été effectuées et qui doivent être download aux stations au sol. En plus des acquisitions candidates (notées CAN) qui ont été effectuées en suivant le planning obtenu via la première optimisation, on retrouve des acquisitions qui avaient été effectuées avant et qui étaient déjà dans la mémoire du satellites (notées REC). Pour toutes ces acquisitions, on précise dans le fichier .dat son ID, la date de fin d'acquisition (imposée à 0 pour les acquisitions dites REC) ainsi que sa durée d'upload (donnée par le volume divisée par la vitesse d'upload).

La seconde composante caractérise les fenêtres de download et permet de décrire les moments auxquels le satellite peut download les acquisitions. Pour ce, on identifie chaque fenêtre par un ID. On donne de plus la date à laquelle la fenêtre s'ouvre ainsi que la date à laquelle la fenêtre se ferme.

— Ordonnancement des acquisitions à download par ordre chronologique

Après quelques essais, on peut remarquer que le code Opl permettant d'optimiser la download des prises de vue se fait plus simplement lorsque les acquisitions sont rangées dans l'ordre chronologique. Il a donc été décidé dans la suite de conserver cette ordre chronologique qui n'affecte en rien le code du reste du

projet.

— **Suppression des acquisitions non désirées**

Certaines acquisition sont effectuées après la date de fin de la dernière fenêtre de download. Ces acquisitions ne peuvent donc pas être download au cours de cette période. Pour cette raison, nous ne mettons pas dans le fichier .dat les acquisitions dont la date de fin est après la date de fin de la dernière fenêtre de download.

3.2.2 Création du code *OPL* pour l'optimisation du download

Notre nouveau code OPL a pour but d'optimiser le nombre de fenêtres de vidage utilisées, et de générer le DownloadPlan sous le même format que celui produit par la classe *BadDownloadPlanner* du projet Java, dans le but d'être lisible par *PlanViewer*. Nous allons détailler les éléments de ce fichier OPL :

— **Variables de décision**

La première variable de décision est une matrice de booléens indexée sur les fenêtres de vidage et sur les acquisitions. Elle indique quelle fenêtre sera utilisée pour vider chaque pour chaque acquisition. Cette matrice aura, à la fin de l'optimisation, un unique "1" par ligne. En effet, chaque acquisition est vidée, car nous nous sommes assurés de retirer celles trop tardives pour être vidées (cf partie 3.2.1), et nous ne pouvons pas vider deux fois la même acquisition.

La deuxième est un tableau de flottants portant sur chaque acquisition, exprimant la date de début de vidage de chacune d'entre elles.

— **Fonction objectif**

La fonction objectif est simplement le nombre de fenêtres de vidage non utilisées, que nous cherchons à minimiser.

— **Contraintes**

Les contraintes mises en places sont les suivantes :

1. *Toutes les acquisitions sont vidées après leur date d'acquisition*

2. *"First-in first-out"*

De deux acquisitions, celle prise en premier sera vidée en premier, et la deuxième doit être au moins une durée de vidage plus loin. Cette condition nous assure que deux acquisitions ne pourront pas être vidées en même temps.

3. *Cohérence acquisition - fenêtre*

Si telle acquisition est vidée dans telle fenêtre, il est nécessaire que la date de vidage soit postérieure à la date de début de la fenêtre, et antérieure à la date de fin moins la durée de vidage. Pour cette dernière contrainte, un Big-M a été utilisé. Nous avons cherché à exprimer ce big-M en fonction des variables, mais nous avons échoué, et nous nous sommes donc contentés d'une grande valeur. Cela ne pose pas trop de problème, car cette optimisation est relativement rapide.

4. *Volume maximal des fenêtres*

La durée totale de vidage de toutes les acquisitions assignées à une fenêtre doit être inférieure à la durée de ladite fenêtre.

En l'état, cette optimisation fonctionne, mais présente quelques défauts. En particulier, la solution doit assigner une fenêtre à chaque acquisition. Ce problème s'est tout d'abord manifesté car la dernière acquisition prise sur une période est souvent postérieure à la fin de la dernière fenêtre de vidage. Nous avons toutefois résolu ce problème en retirant ces acquisitions des fichiers .dat et en les traitant comme enregistrées pour la prochaine période. En revanche, on peut imaginer un autre cas problématique dans lequel on prendrait beaucoup d'acquisitions tout en ayant qu'une petite fenêtre à disposition. Dans cette situation, notre OPL ne trouverait pas de solutions. Nous avions initialement estimé cette situation peu probable, mais si nous en avions eu le temps nous aurions cherché à modifier le fichier .mod pour résoudre ce problème.

3.3 Visualisation des résultats

La partie visualisation des résultats sous forme de graphique nous a paru essentielle afin de connaître très rapidement les résultats ainsi que le respect des contraintes.

3.3.1 Outil de visualisation des acquisitions effectuées

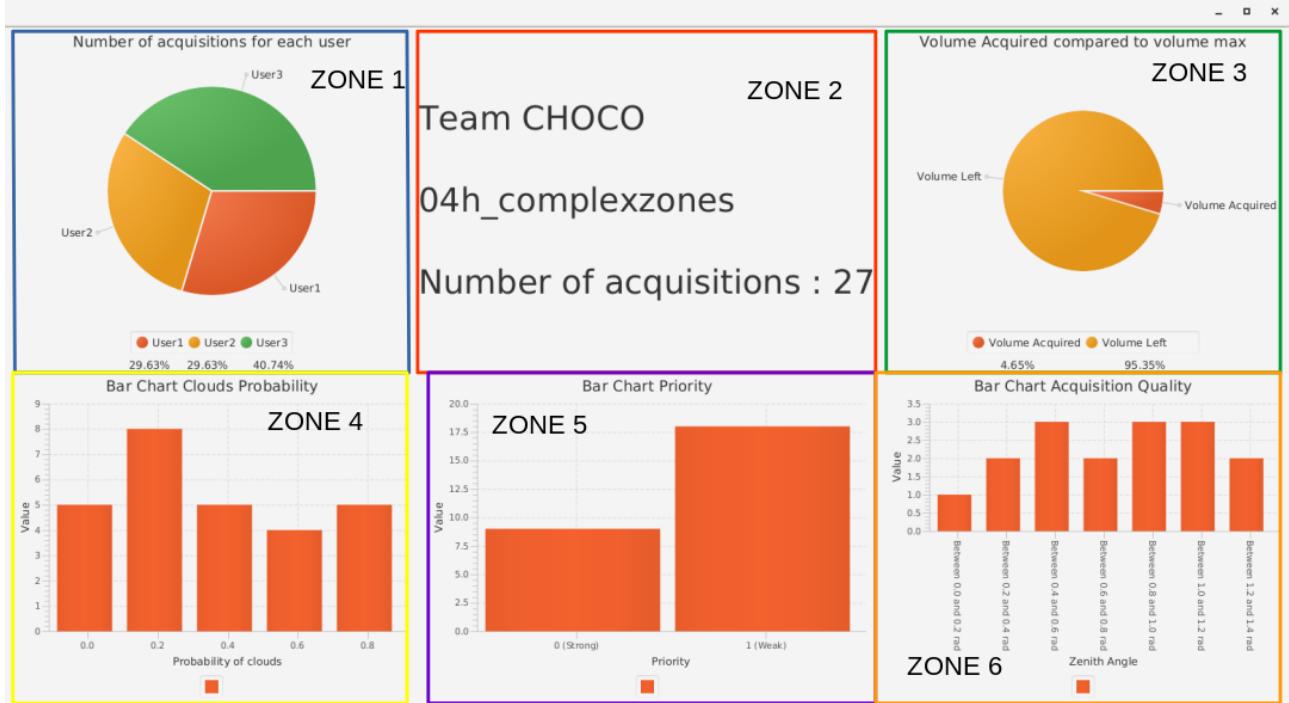


FIGURE 3.3 – Caption

- **Zone 1 :** Le graphique présenté compare le partage des utilisateurs en terme de nombre d'acquisitions. On rappelle que l'utilisateur 1 demande 50%, le 2 30% et le 3 20%.
- **Zone 2 :** Dans ce cadre, nous rappelons le nom de l'équipe. Puis, le code JAVA récupère le cas qui a été optimisé (ici le cas complexe 4h). Enfin, le nombre d'acquisitions est donné pour un résumé rapide.
- **Zone 3 :** Ce graphique camembert présente le volume de données acquis par rapport au volume total possible. Cependant, nous avons enlevé les doublons dans l'optimisation, mais ils sont toujours présents dans le volume total possible.
- **Zone 4 :** Ce diagramme en barres présente la répartition des acquisitions en fonction de la probabilité de nuage. Il est utilisé pour valider notre critère.
- **Zone 5 :** Ce diagramme en barres présente la répartition des acquisitions en fonction de la priorité.
- **Zone 6 :** Ce diagramme en barres présente la répartition des acquisitions en fonction de l'angle de zenith.

Cet outil de visualisation a été développé pour nous permettre de valider nos critères d'optimisation. En effet, nous avons implémenté des poids pour chaque critère dans notre fonction objectif. De ce fait, l'utilisateur a la possibilité de choisir quel type d'image il souhaite, que ce soit des images ayant une forte qualité, ou des images où il est sur de ne pas avoir de nuages. Pour obtenir le type d'image souhaité, l'utilisateur a uniquement à modifier les poids attribués à chaque paramètres de l'optimisation. Ainsi, un poids fort pour un certain paramètre le rendra plus influent au moment du choix des acquisitions à effectuer.

3.3.2 Qualité de la photo

L'un des paramètres sur lequel l'utilisateur peut influer en fixant un poids plus ou fort est la qualité de la photo. Dans le code original, l'optimisation ne prenait pas en compte l'angle de zenith, qui correspond à la qualité. Dorénavant, en mettant le poids de la qualité au maximum sur le nouveau code, on s'attend à avoir beaucoup d'acquisitions avec des angles de zénith proche de zéro.

C'est effectivement ce que l'on observe sur la figure 3.4. Dans le cadre de l'ancien code (a), on observe qu'il n'y a pas de règle quant à la répartition des images en terme d'angle. Cependant, en fixant un poids élevé avec notre code (b), nous présentons bien plus de photos présentant un angle au zénith faible et la majorité des

acquisitions présente alors un angle entre 0 et 0.2 rads.

L'influence de ce poids se voit aisément sur les photos satellites de cette même figure. Une photo avec un angle élevé (c) va être légèrement de biais, la photo va alors faire ressortir les reliefs mais une partie des éléments ne sera pas visible à cause de ces mêmes reliefs. La photo avec un angle nul, elle, permet d'avoir une vue satellite vue du dessus.

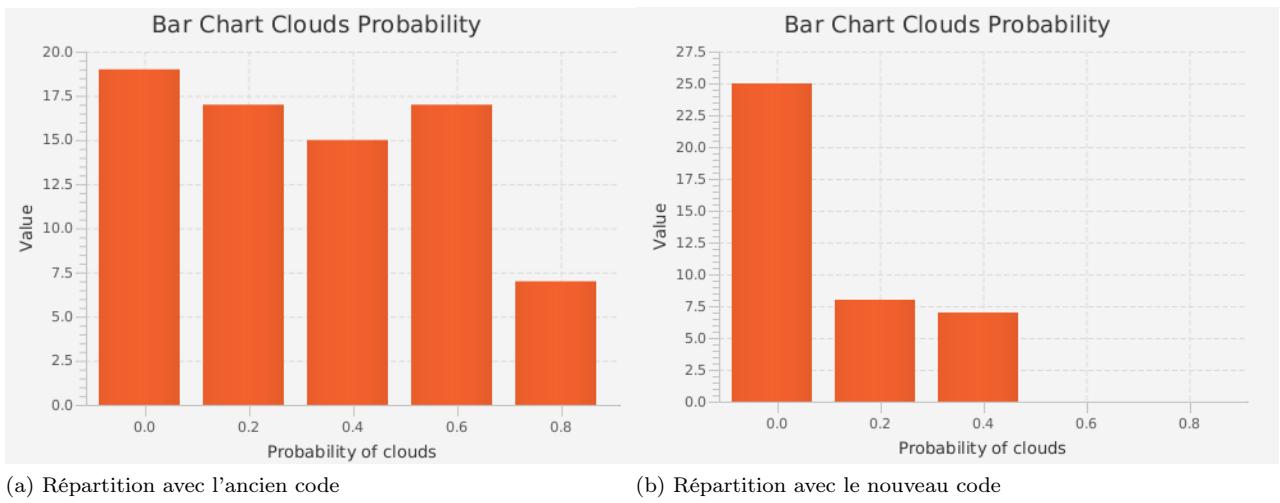


FIGURE 3.4 – Validation de la qualité dans notre critère

3.3.3 Présence de nuages dans la photo

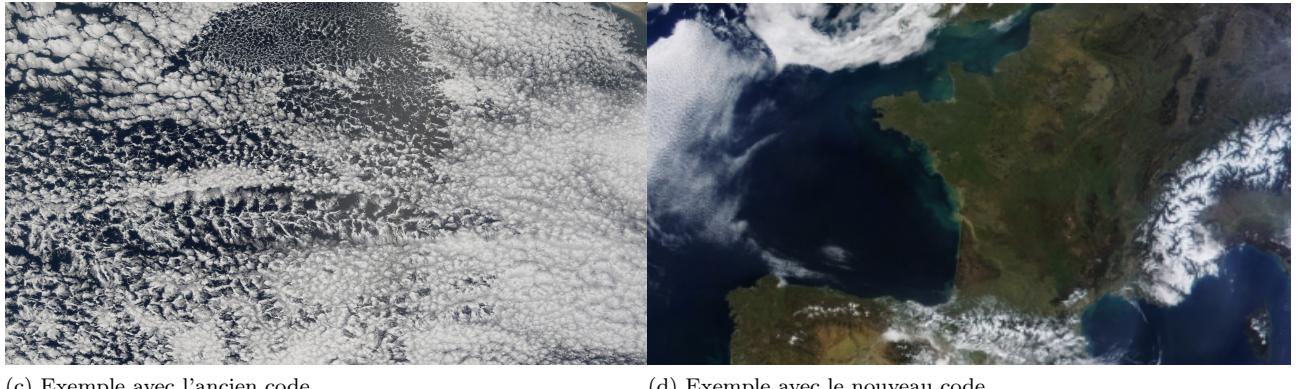
Un autre paramètre sur lequel l'utilisateur peut influer en mettant un poids plus ou moins fort est la probabilité de nuages dans les acquisitions demandées. De même que pour l'angle au zénith, le code d'origine ne permettait pas de contrôler la quantité de nuages debout. En mettant un poids fort sur ce paramètre, on récupère alors des images avec une faible présence en nuage.

On observe sur la figure 3.5 l'effet de la pondération de ce paramètre. Dans le cas du code d'origine (a), on a une répartition aléatoire les probabilités de nuages. Cependant, dans le cas d'une pondération forte de ce paramètre (b), il ne demeure majoritairement que les acquisitions présentant une faible probabilité de nuages.



(a) Répartition avec l'ancien code

(b) Répartition avec le nouveau code



(c) Exemple avec l'ancien code

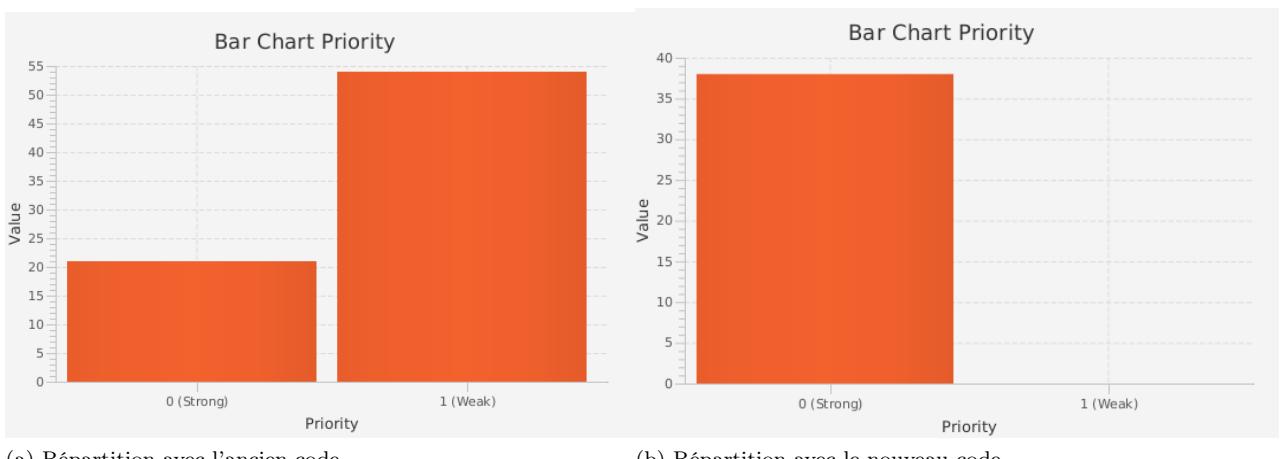
(d) Exemple avec le nouveau code

FIGURE 3.5 – Validation de la probabilité de nuage dans notre critère

3.3.4 Gestion de la priorité dans les acquisitions

Les acquisitions peuvent être prioritaires ou non. Ainsi, une acquisition de priorité 0 est importante alors que une acquisition de priorité 1 l'est moins. En mettant un poids fort sur ce paramètre, l'utilisateur peut alors décider de n'avoir que ou presque que acquisitions de priorité élevée.

On peut observer sur la figure 3.6 le comportement de l'optimisation lorsque l'on change les poids. On observe ainsi que dans le code original (a), la gestion des priorités n'était pas faite. On se retrouvait donc avec de nombreuses acquisitions non-prioritaires. Au contraire, dans le cas d'une forte pondération de ce paramètre, on n'obtient que des acquisitions présentant une priorité importante.



(a) Répartition avec l'ancien code

(b) Répartition avec le nouveau code

FIGURE 3.6 – Validation de la probabilité de nuage dans notre critère

3.3.5 Validation des quotas par utilisateur

Nous avons décidé de contraindre le partage par utilisateur. En effet, nous pensons que cela doit être vérifié pour chaque cas d'optimisation. Pour vérifier que nos contraintes fonctionnent, nous avons mis en place un diagramme permettant de visualiser cette contrainte. Comme vu sur la figure 3.7, à 10% près les quotas sont bien respectés.

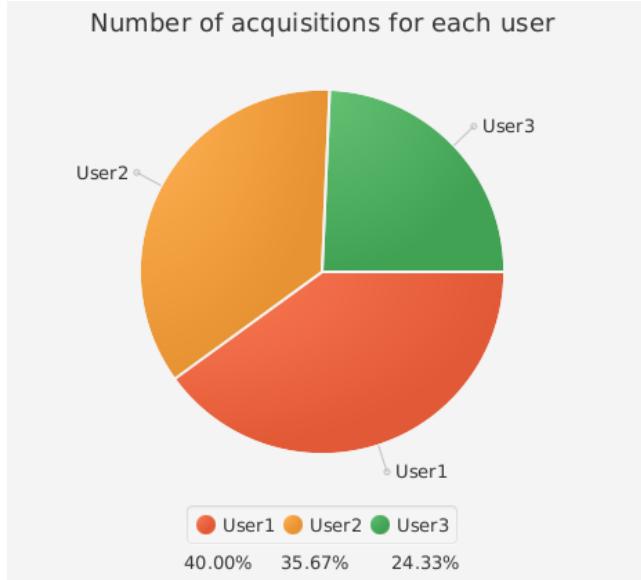


FIGURE 3.7 – Diagramme des quotas par utilisateur

3.4 Automatisation du process à l'aide d'un script BASH

Comme expliqué précédemment, le script bash développé permet d'automatiser le processus et donc toute la chaîne d'optimisation. Il a été écrit en bash, car nous travauillons sous Linux. La redondance des activités nécessaires telles que les copier/coller, les lancements des configurations d'exécutions nous ont paru être une source d'erreurs facilement faisable après de nombreuses itérations. Le script automatise toutes ces tâches et est facilement modifiable, puisqu'il permet de choisir le cas à traiter ainsi que le temps maximum d'exécution de l'optimisation par exemple. La liste des tâches effectuées par le script est détaillée ci-dessous.

Script bash :

- Lancer le script
- Le script demande à l'utilisateur la configuration à lancer (simple ou complex zones avec 4h, 12h ou 24h)
- Ecriture dans la classe *Params* de la configuration puis compilation de cette classe
- Appel de la classe *AcquisitionPlanner* et création d'un fichier **.dat**
- Le script demande à l'utilisateur le temps maximum alloué pour l'optimisation des acquisitions, car l'optimisation peut être très longue (cf section 3.1)
- Ecriture du temps de calcul dans le fichier **.mod**
- Exécution du modèle opl avec le fichier **.dat** précédemment copié coucou moussier
- Une fois fini, ou le temps maximum atteint, copie des fichiers dans le dossier *output* du code JAVA
- Appel de la classe *DownloadPlanner* avec les fichiers texte solutions de l'acquisition
- Création d'un fichier **.dat** pour l'optimisation du déchargement, puis copie de ce fichier dans le dossier OPL
- Exécution du modèle OPL pour le déchargement des données
- Copie des solutions dans le dossier *output* du code JAVA
- Exécution de notre classe *Analytics* (cf section 3.3) et de la classe *PlanViewer* pour observer la solution complète

4 Analyse des résultats

4.1 Analyse des performances de l'acquisition

4.1.1 Analyse des algorithmes

Le temps de calcul est une caractéristique essentiel pour mener à bien les optimisations. Le projet original ne terminant pas même pour le cas simple 4h, il fallait pour limiter cela instaurer une limite de temps de calcul. Pour éviter cela, la suppression de la matrice next a beaucoup aidé. Ainsi notre algorithme optimisé termine dans le pire cas (complexe 24h) en moins de 2 minutes avec les ordinateurs disponibles au centre informatique. Les temps de calculs comparés dans la table 4.1 sont les temps de calculs de l'algorithme fixant le startTime et utilisant la matrice next et ceux de l'algorithme fixant le startTime mais n'utilisant pas la matrice next. On s'aperçoit dès lors que les gains en temps de calcul en l'absence de matrice next sont très précieux pour traiter les cas complexes.

Cas	Avec la matrice next			Sans la matrice next		
	Temps de calcul	Fonction objectif	Nombre d'acquisitions	Temps de calcul	Fonction objectif	Nombre d'acquisitions
Simple 4h	< 10s	11.6	50	< 10s	11.6	50
Complexe 4h	80s	16.2	70	< 10s	16.2	70
Simple 12h	2300s	41.2	210	< 10s	41.2	210
Complexe 12h	✗	✗	✗	< 10s	66.7	300
Simple 24h	✗	✗	✗	< 20s	76.8	375
Complexe 24h	✗	✗	✗	< 60s	197.2	560

TABLE 4.1 – Tableau récapitulatif des résultats avec le startTime fixé à l'EarliestStartTime. Les poids utilisés sont 100 pour le volume, 40 pour les nuages, 40 pour la priorité et 20 pour la qualité

Avec le tableau 4.2 (simulé avec le cas complexe 4h) on s'aperçoit que la meilleure solution obtenue est avec un algorithme OPL utilisant la matrice next et fixant le startTime et un avec l'algorithme n'utilisant pas la matrice next. Les autres méthodes sont plus lentes et non pas le temps de trouver une solution optimale. Cependant si l'on prend des cas avec plus d'acquisitions comme le cas complexe 12h, seul l'algorithme n'utilisant pas la matrice next termine avant la limite de temps fixée à 10 minutes.

D'une part, les nombres de contraintes et d'itérations sont plus élevés et croissent plus rapidement lorsque les algorithmes utilisent la matrice next. L'évolution du nombre de contraintes et de variables en fonction du nombre d'acquisition candidates est donnée à la figure 4.1 pour la méthode utilisant la matrice next et celle ne l'utilisant pas. On constate que le nombre de contraintes et de variables explosent pour la méthode next. En effet, les contraintes et les variables sont environ autour de 15 millions pour 2000 acquisitions tandis que la méthode sans next frôle difficilement le million de contraintes et variables.

	BadPlanners	Avec next + startTime = dvar float+	Avec next + starTime = dvar int	Avec next + startTime = EarliestS- tarTime	CP	Sans next + startTime = EarliestS- tarTime
Nombre de contraintes	43 267	299 075	299 075	299 075	299 082	60 241
Nombre de variables	43 679 (43 472 binaires)	579 221 (438 984 binaires)	579 221 (438 984 binaires)	579 221 (438 984 binaires)	439 450	936 binaires
Itérations	506 264	706 425	721 324	554 871		92
Fonction objectif	15	5	9,1	16,2	2,5	16,2
Nombre d'acquisitions	15	30	45	70	20	70

TABLE 4.2 – Tableau récapitulatif des contraintes/variables/objectif pour chaque méthode dans le cas complexe 4h avec un temps maximum de résolution de 10 min. Les poids utilisés sont 100 pour le volume, 40 pour les nuages, 40 pour la priorité et 20 pour la qualité

	BadPlanners	Avec next + startTime = dvar float+	Avec next + starTime = dvar int	Avec next + startTime = EarliestS- tarTime	CP	Sans next + startTime = EarliestS- tarTime
Nombre de contraintes	1 695 207	5 793 326	5 793 326	5 793 326		987 860
Nombre de variables	1 697 807 (1 696 506 binaires)	9 932 292 (7 860 612 binaires)	9 932 292 (7 860 612 binaires)	9 932 292 (7 860 612 binaires)		3 964 binaires
Itérations	74 306	0	0	0		431
Fonction objectif	0	0	0	0	0	66,8
Nombre d'acquisitions	0	0	0	0	0	300

TABLE 4.3 – Tableau récapitulatif des contraintes/variables/objectif pour chaque méthode dans le cas complexe 12h avec un temps maximum de résolution de 10 min. Les poids utilisés sont 100 pour le volume, 40 pour les nuages, 40 pour la priorité et 20 pour la qualité

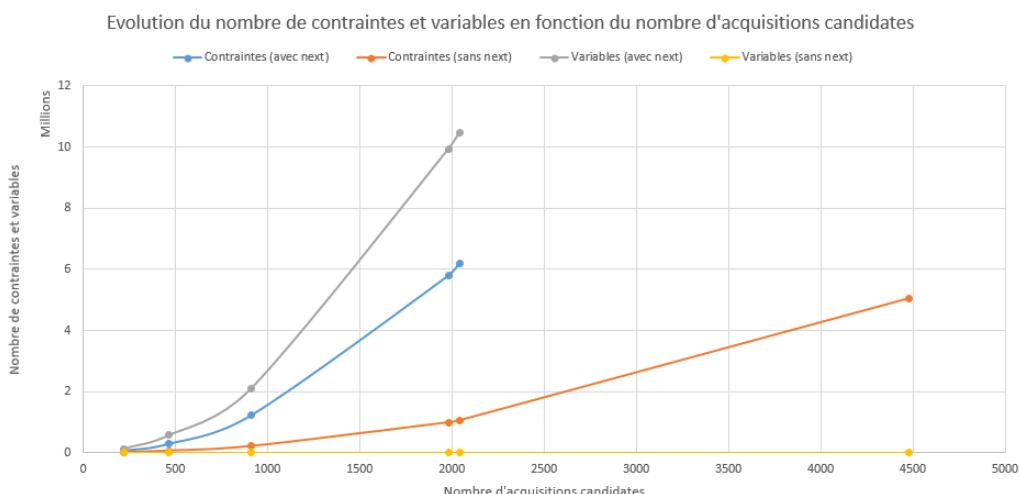


FIGURE 4.1 – Évolution des contraintes et variables en fonction du nombre d'acquisition candidates

Dans le cas où nous avons un temps limité pour traiter un grand nombre de d'acquisition possible alors la meilleure stratégie à adopter est de choisir l'algorithme fixant le startTime et n'implémentant pas la matrice next. Néanmoins si l'on dispose d'un supercalculateur ou de plus de temps, une meilleure optimisation serait obtenue en cherchant à optimiser le startTime avec la matrice next.

4.1.2 Analyse des acquisitions sélectionnées

L'analyse des caractéristiques des acquisitions choisies est importante pour déterminer la qualité globale des images et la satisfaction du client. La figure 4.2 montre les caractéristiques des images dans le cas complexe 4h choisies en calculant 10 min. On s'aperçoit tout d'abord que le nombre d'acquisition est relativement faible (27) alors que la fonction à optimiser est justement le nombre d'images sélectionnées. D'autre part, il n'y a pas de quotas respectés pour les différents utilisateurs. Le volume total des acquisitions est très faible (moins de 5%). De nombreuses images ont des probabilités de nuages très élevées (0,8) et la majorité des images sont d'ordre non prioritaires.

Sur la figure 4.3, on peut observer les effets de notre algorithme incluant le respect des quotas en terme de volume par utilisateur, respect de la qualité des images et de la maximisation du volume d'images acquises. Les effets de l'optimisation sont très nets et notre algorithme permet également de modifier les coefficients qui pondèrent la fonction objectif en fonction des besoins.



FIGURE 4.2 – Visualisation des caractéristiques des acquisitions candidates avec le projet original sur le cas complexe 4h pendant 10 min



FIGURE 4.3 – Visualisation des caractéristiques des acquisitions candidates avec le projet optimisé sur le cas complexe 4h

4.2 Analyse des performances du déchargement d’images

La partie de déchargement a été implémentée pour limiter le nombre de fenêtre utilisé. Le projet original utilisait quant à lui n’importe quelle station disponible pour décharger comme on peut le constater sur la figure 4.4. Sur la figure 4.5, les résultats de l’algorithme de déchargement implémenté montrent bien que le nombre de fenêtre utilisé est minimal. En l’occurrence, seuls deux fenêtres ont été utilisées contre 5 pour le projet initial (les fenêtres utilisées sont identifiées par un cercle). Cette optimisation permet de réduire le coup de déchargement de données en utilisant le moins de stations au sol possible. Quant à la durée de l’optimisation, tous les cas ont été traités en moins de 10s.

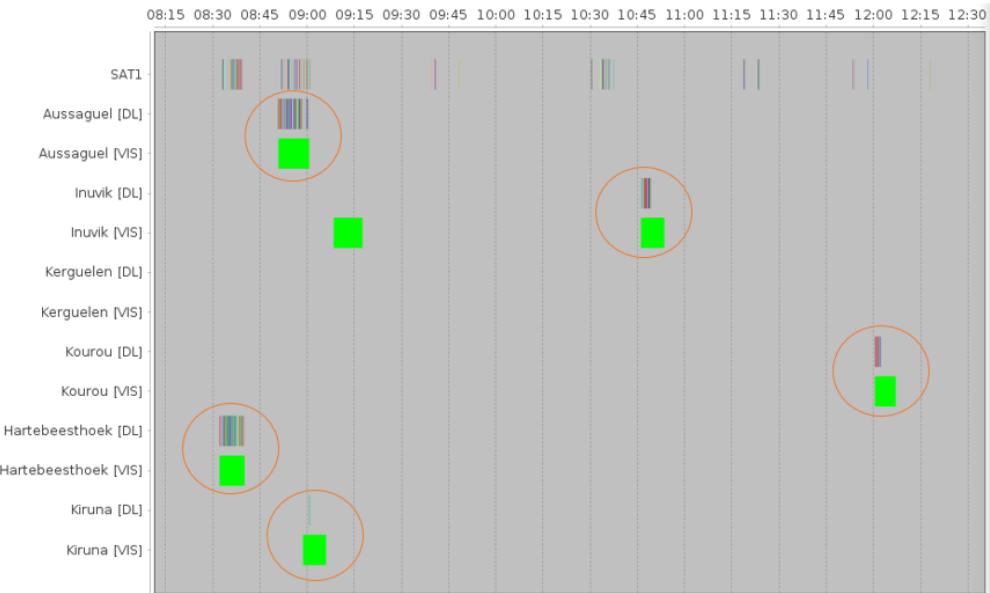


FIGURE 4.4 – Visualisation du processus d’acquisition et déchargement avec le projet original

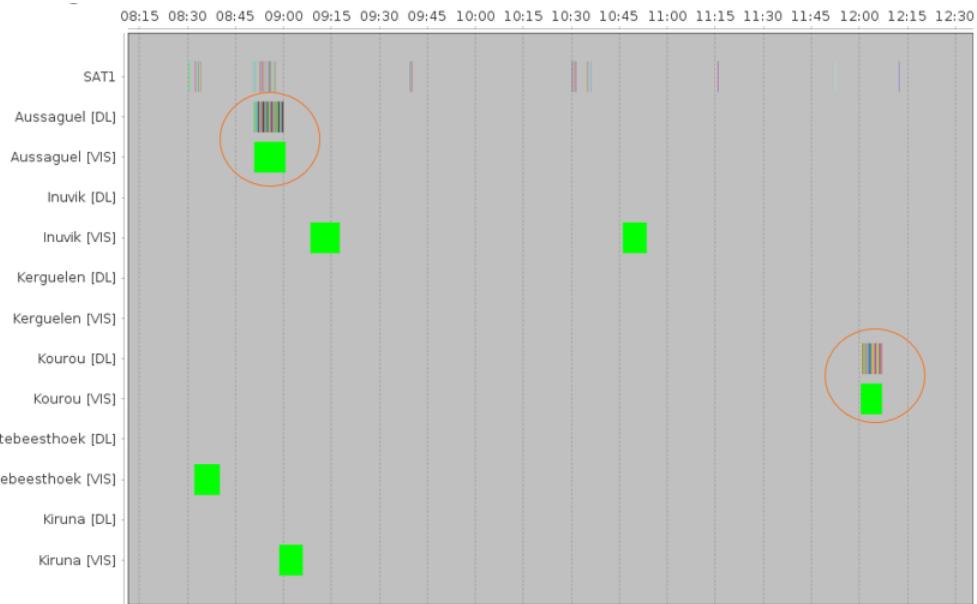


FIGURE 4.5 – Visualisation du processus d’acquisition et déchargement avec le projet optimisé

4.3 Améliorations possibles et pistes de réflexion supplémentaires

Pour la partie acquisition

Pour la partie déchargement d’images, notre algorithme ne converge pas dans le cas complexe 24h car il a été codé de sorte à ce qu’il considère que toutes les fenêtres sont suffisantes pour décharger toutes les images acquises. Ce qui n’est pas le cas en réalité. Ainsi, il faudrait améliorer le code pour permettre le déchargement du maximum d’images tout de même.

Pour la partie acquisition, pour pouvoir effectuer les simulations, nous avons fixé le startTime à l’earliestStartTime ce qui est une grande approximation. Elle permet de faire tourner le code très rapidement au détriment de cas possibles. En se penchant sur les heuristiques et sur d’autres façons de supprimer la matrice next sans fixer le startTime, nous pourrions accélérer le code tout en restant réaliste et plus précis. Nous pourrions aussi investiguer les résultats des optimisations en fonction de la nature des startTime (float ou int).

Pour la partie BASH, il serait intéressant d’inclure une interface graphique plus ergonomique et de permettre à l’utilisateur de choisir les poids des différents coefficients intervenants dans la fonction objectif.