université de BORDEAUX

Collège Sciences et Technologies
Introduction à l'Ingénierie de l'Optimisation

Protection des fonds marins

Tom GILLET, Carlos PASCUAL

CMI Optim L1

2022–2023

Introduction à l'Ingénierie de l'Optimisation

Table des matières

1 Introduction

On s'intéresse ici à un problème de protection des fonds marins. Le projet a pour objectif d'étudier différentes versions de plans de protection d'espèces marines. On s'intéresse à la protection d'un ensemble d'espèces E. Le fond marin est modélisé de manière très simplifiée comme un ensemble de secteurs, qui correspondent à des cellules dans une grille en deux dimensions. Les décisions à prendre consistent à protéger un certain nombre de secteurs, de manière à ce qu'un nombre suffisant d'individus de chaque espèce soit localisé dans un secteur protégé. Lorsqu'un secteur est protégé, un certain nombre d'activités (pêche, tourisme, ...) sont interdites, de manière à préserver les espèces présentes.

Exemple illustratif

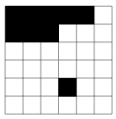

FIGURE 1 – Exemple de secteurs dans une grille 7x7 avec des zones terrestres(noir)

Les premières tentatives de conception de réserves reposaient sur des règles intuitives : estimation d'une valeur de conservation associée à une zone donnée (Helliwell, 1967 ; Tubbs et Blackwood, 1971 ; Goldsmith, 1975 ; Wright, 1977), puis classement des zones en fonction de leurs valeurs (Tans, 1974 ; Gehlbach, 1975 ; Rabe et Savage, 1979), et enfin enrichissement du processus par des approches de classification itératives pour pallier le manque de complémentarité des réserves (Kirkpatrick, 1983 ; Margules et al., 1988 ; Pressey et Nicholls, 1989). Cependant, il faudra attendre jusqu'à ce qu' Adrien Brunel et Jérémy Omer étudient en détail l'optimisation de la protection des fonds marins.

Dans ce rapport, nous présentons des algorithmes approchés pour résoudre le problème. Ces algorithmes sont comparés expérimentalement entre eux, chacun partant d'une hypothèse de base différente qui sera explicitée. Le but pour toutes les versions du problème considérées est de minimiser le nombre de secteurs protégés. Le rapport est organisé de la manière suivante. La section 2 présente le problème en détail. La

section 3 présente trois algorithmes de résolution de ce problème. La section 4 présente les résultats des tests effectués sur le jeu de données fourni ainsi que leur analyse. La section 5 est dédiée à une conclusion sur le projet.

2 Formalisation du problème

On s'intéresse à la protection d'un ensemble d'espèces E. Le fond marin est modélisé de manière très simplifiée comme un ensemble de secteurs, qui correspondent à des cellules dans une grille en deux dimensions. On note S l'ensemble des secteurs considérés. Dans chaque secteur $s \in S$, on dispose d'une estimation $n_e s$ du nombre d'individus de l'espèce e dans le secteur s. Certains secteurs correspondent à des zones terrestres, elles ne contiennent aucun individu. Pour chaque espèce $e \in E$, on considère qu'elle suffisamment protégée si la somme du nombre d'invididus de l'espèce e dans les secteurs protégés est supérieure à une valeur $p_e \in N$ donnée en paramètre du problème. Les secteurs sont représentés dans une grille de taille (W $\times$ H). Ils sont numérotés de 0 à (W x H) - 1. Les lignes et colonnes sont respectivement numérotées de 0 à H - 1 et 0 à W - 1. Le secteur d'indice i a pour coordonnées (E(i/W), i mod W). Inversement, le secteur de coordonnées (x, y) a pour indice y x W + x.

Dans tout ce qui suit, une solution sera représentée par une liste de secteurs protégés.

3 Algorithmes de résolution

3.1 Première hypothèse : secteurs libres

Dans la première version du problème, on peut sélectionner n'importe quel ensemble de secteurs. La seule contrainte est que l'on doit protéger suffisamment d'individus.

Dans cette section, nous présentons deux heuristiques pour résoudre le problème décrit dans la section 2. La première heuristique consiste à construire la solution sans tenir compte de la minimisation du nombre de secteurs protégés. L' heuristique suivante est gloutonne et ambitieuse ; dans le sens où la construction de la solution se fait en comparant tous les rectangles possibles.

3.1.1 Heuristique idiote

L'algorithme idiot consiste à protéger tous les secteurs s (hormis, évidemment, les secteurs "terres"). L'algorithme s'arrête donc après les avoirs tous "cochés". Dans cet algorithme, la solution est représentée sous la forme d'un coloriage de tous les secteurs s.

3.1.2 Heuristique de la somme

Il s'agit maintenant d'un algorithme constructif. L'idée est la suivante : pour chaque secteur non terre de l'ensemble S considéré au départ, nous calculons la somme de tous les poissons protégés $n_e s$ par ce secteur indépendamment des espèces e On itère alors ce processus pour tous les secteurs. Une fois cette étape terminée, on trie les secteurs par odre croissant selon la somme de poisson qu'il protège. On prend ensuite les cases de la plus grande somme à la plus petite en ajoutant pour chaque espèce la quantité que la case protège jusqu'à ce que la somme du nombre d'individus de chaque espèce e soit supérieure ou égale à la valeur p_e donnée en paramètre du problème.

```
n = 0
test = TestQt(Taille,TabAux[0][0],T,Espece,nbEsp,testEspece)
while test == False :
    solution = solution + [TabAux[n][0]]
    #print(n,TabAux[n])
    n = n + 1
    test = TestQt(Taille,TabAux[n][0],T,Espece,nbEsp,testEspece)
solution = solution + [TabAux[n][0]]
#print("Objectif :",Espece)
#print("Nombre protégé",testEspece,"")
return solution
```

FIGURE 2 – Programmation de l'algorithme 1 de résolution selon l'heuristique de la somme

Exemple illustratif

3.1.3 Heuristique de la somme 2

On a maintenant une version améliorée, optimisée de l'algorithme antérieur. En effet, dès qu'une espèce est déjà protégée dans sa totalité, celle-ci n'est plus prise en compte dans les conditions d'itérations de l'algorithme. Cependant, cette version n'est pas toujours plus efficace que sa coéquipière, comme nous le verrons lors des expérimentations.

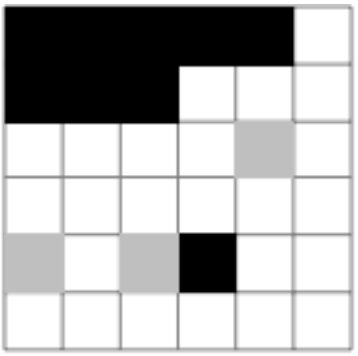

FIGURE 3 – Solution possible pour l'hypothèse 1. Les secteurs protégés sont représentés en gris. On protège trois secteurs

```python
def algo2(T,Taille,Espece,nbEsp):
    NonExclu=[0]*nbEsp
    for i in range(len(NonExclu)):
        NonExclu[i] = i+1
    solution = []
    TabAux = [0]*(Taille**2)
    testEspece = [0]*nbEsp

    while TestToutProteger(NonExclu) == False:
        nbCase = 0
        for i in range(Taille):
            for j in range(Taille):
                if SecteurAExclure(i,j,solution,Taille) == True:
                    ValeurSomme = 0
                else:
                    ValeurSomme = sommeEspece(T[i][j],NonExclu)
                TabAux[nbCase] = [j*Taille+i,ValeurSomme]
                nbCase = nbCase + 1
        TabAux = trifusion(TabAux)

        n = 0
        test = TestQt(Taille,TabAux[0][0],T,Espece,testEspece,NonExclu)
        while test == False :
            solution = solution + [TabAux[n][0]]
            n = n + 1
            test = TestQt(Taille,TabAux[n][0],T,Espece,testEspece,NonExclu)
        solution = solution + [TabAux[n][0]]
    return solution
```

FIGURE 4 – Programmation de l'algorithme 1 de résolution selon l'heuristique de la somme 2

3.2 Deuxième hypothèse : secteurs carrés

On va maintenant chercher à se rapprocher d'une solution réaliste en introduisant une notion de connexité. On considère qu'une zone est un ensemble de secteurs protégés qui se touchent. Sous l'hypothèse 2, chaque zone doit être un rectangle, deux zones ne peuvent pas se toucher, et on ne doit pas utiliser plus de K rectangles (K un entier donné en paramètre). Une zone rectangulaire ne peut pas recouvrir un secteur terrestre.

3.2.1 Heuristique du meilleur rectangle

Pour cette deuxième hypothèse, on va supposer que le nombre K passé en paramètre est compris entre 2 et 7 (inclus). Nous allons comparer tous les rectangles possibles entre eux afin de créer un classement basé sur des "scores", et on prendra donc les K rectangles non connexes ayant le meilleur score comme solution. Ce score est basé sur le fait que, selon le nombre K de rectangles dont on dispose, le "meilleur rectangle" doit protéger une certaine part du total des espèces qui dépend du nombre K. Le "deuxième meilleur" rectangle doit protéger une part en plus petite car il est, au maximum aussi bon que le premier, donc sa part à protéger doit être innférieur. *e.* La première étape consiste alors à regarder toutes les combinaisons possibles. Pour cela, nous allons partir du premier secteur s non terre et nous allons créer tous les rectangles dont lesquels ce secteur est innclus. Puis, nous enlèverons ce dernier de la liste de secteurs possibles. On recommence avec le secteur qui est à côté. Si une des construction créée à partir d'un des secteurs suivants existait déjà, elle n'est évidemment pas prise en compte (elle serait alors faussement comptait deux fois). Une fois ce processus terminé, on va regarder parmi la liste de tous les rectangles qui viennent d'être crées celui qui, avec le moins de secteurs, protège la part imposée : il s'agit du rectangle ayant le meilleur score. On enlève ce dernier de la liste des rectangles. On regarde à nouveau le rectangle qui, avec le moins de secteurs, protège la deuxième part imposée. On itère ce processus avec les K parts imposées ; en vérifiant à chaque étape que les rectangles ne soient pas connexes.

Exemple illustratif

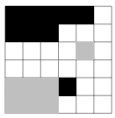

FIGURE 5 – Solution possible pour l'hypothèse 2. On protège 7 secteurs à l'aide de deux zones rectangulaires

3.3 Le minorant

Pour ce projet, il nous est également utile d'avoir une référence, un point de comparaison : c'est le rôle du minorant. Celui-ci est obtenu en imaginant qu'une seule espèce doit être protégée ; ce qui simplifie énormément le résultat. Ainsi, dans la plupart des cas, la valeur du minorant est inatteignable mais est indicative, et nous montre la cohérence des résultats obtenus.

4 Expérimentations

Nous avons implémenté nos algorithmes dans le langage Python. Nous avons testé les algorithmes présentés à la section 3 sur le jeu de données fourni pour le projet. Ce jeu de données comporte 343 cas test présentant des situations différentes : des archipels, des détroits, des iles, des grandes iles ou encore des péninsules. Les tests ont été lancés sur un ordinateur possédant un processeur Intel Core i7-12700H CPU @ 2.30GHz× 8, 15,7 Go de mémoire vive et ayant comme système d'exploitation Windows 11 22H2. Le premier but de ces expérimentations est de comparer les solutions retournées par les trois heuristiques.

Pour chaque donnée, nous comparons la valeur de la solution obtenue à la valeur de la meilleure solution obtenue par les trois heuristiques. Nous ne présentons que les résultats agrégés dans cette section. Les résultats détaillés de nos tests sont fournis dans l'annexe A. Le tableau 1 présente un résumé des résultats

de nos expérimentations. Il montre pour chaque méthode le temps (en moyenne) CPU (Colonne *Temps CPU*) et le nombre de données pour lesquelles la méthode a retourné la meilleure solution parmi les trois méthodes (Colonne *#Données MS*). Dans le tableau ci-dessous, le temps CPU est rapporté en secondes. Pour l'heuristique du meilleur rectangle, le cas k=4 (valeur considérée comme intermédiaire) est celui pris en compte.

```python
def Minorant(Taille,nbEsp,Espece,TableauLecture):
    indiceMax = 0
    max = 0
    for i in range(nbEsp):
        if max<Espece[i]:
            max = Espece[i]
            indiceMax = i
    TabTrie = [0]*(Taille**2)
    indice = 0
    for x in range(Taille):
        for y in range(Taille):
            TabTrie[indice] = TableauLecture[x][y][2+indiceMax]
            indice = indice +1
    TabTrie = sorted(TabTrie,reverse=True)
    sommeEspece = 0
    sommeSecteur = 0
    for z in TabTrie:
        sommeEspece = sommeEspece + z
        sommeSecteur = sommeSecteur +1
        if sommeEspece > Espece[indiceMax]:
            return sommeSecteur
```

FIGURE 6 – Programmation de l'algorithme du minorant

TABLE 1 – Comparaison des différentes méthodes

Algorithme	Temps CPU (ms)	#Données MS
Heuristique de la Somme	0,0725	1/343
Heuristique de la Somme 2	0,7506	323/343
Heuristique du meilleur Rectangle	0,0316	19/343

L'heuristique idiote est la plus rapide à s'exécuter, elle fournit cependant des solutions ayant une valeur beaucoup trop grande et qui n'est pas intéressante, voilà-pourquoi elle n'apparaît pas sur nos figures. L'heuristique de la somme 2 retourne dans la grande majorité de cas (94%) la meilleure solution mais le temps de calcul moyen est bien supérieur (entre 10 et 25 fois plus en moyenne).

Pour l'heuristique du meilleure rectangle le temps moyen semble ici très faible car seulement de petites instance ont été testées (Seulement les 10x10 et quelques 30x30), car contrairement aux deux algorithmes pour les secteurs libres la complexité n'est pas logarithmique. Il ne s'agit cependant pas des meilleures instances à tester car on observe que de nombreuses instances n'ont pas de solutions avec cet algorithme, notamment pour k = 6. Cela peut s'expliquer par le fait que les rectangles ne peuvent pas être collés et donc plus l'instance est petite plus les zones restreintes autour des rectangles prennent un pourcentage important des rectangles disponibles.

Une deuxième manière d'évaluer la qualité des solutions produites par les heuristiques est d'évaluer la distance entre les solutions retournées et la solution optimale. Pour quantifier l'écart à la solution optimale, nous avons implémenté le calcul du minorant de la section 3.3.

La Table 2 présente un résumé de l'écart relatif observé entre la solution retournée par chaque méthode et le minorant. Chaque ligne correspond à une heuristique. On rapporte pour chacune d'elles la moyenne de la valeur $(z - m)/z$, où z est la valeur de la solution retournée par l'algorithme, et m la valeur du

minorant. Nous rapportons aussi l'écart-type. Pour cela, nous calculons l'écart relatif (appelé *gap*) vis à vis de la meilleure solution de la façon suivante : $(z - z^{Best})/z$, où z est la valeur de la solution retournée par l'algorithme, et z^{Best} est la valeur de la meilleure solution calculée durant nos tests.

TABLE 2 – Comparaison des gaps entre la valeur de la solution calculée et le minorant

Algorithme	Gap Minorant	
	Moyenne	Ecart type
Heuristique de la Somme	59,00%	18,77%
Heuristique de la Somme	52,35%	0,00%
Heuristique du meilleur Rectangle	69,26%	71,14%

Nous voyons que l'écart est élevé quelle que soit la méthode utilisée. La meilleure méthode (plus proche voisin) atteint un gap moyen de 72%, ce qui est très important.

En conclusion sur les résultats numériques, si on dispose d'un temps de calcul de plusieurs secondes, on utilisera en priorité l'heuristique MI, qui fournit les meilleurs résultats. Si on veut une solution "en temps réel", on favorisera l'heuristique PPV, qui s'exécute de manière très rapide.

5 Conclusion

Dans ce projet, nous avons développé trois heuristiques pour fournir des solutions au problème de protection des fonds marins. Nous avons aussi montré qu'il était possible de calculer un minorant pour la valeur optimale du problème.

Nos expérimentations montrent que sur les données test fournies, la deuxième heuristique de la somme est plus performante en moyenne que la première heuristique de la somme. C'est un résultat qui n'est pas très surprenant car dans le deuxième algorithme on essaie de ne pas protéger plus qu'il le faut une espèce alors que le premier ne prend pas en compte ce paramètre. La deuxième heuristique de la somme permet de résoudre ce problème en recalculant une somme sans prendre en compte les espèces déjà protégées.

Comme l'on pouvait s'en douter, l'heuristique idiote donne quant à elle de très mauvais résultats (elle choisie tout simplement tous les secteurs). Ceci justifie l'utilisation d'heuristiques plus perfectionnées,telles que l'heuristique de la somme ou du meilleur rectangle pour trouver des solutions acceptables.

Le gap avec le minorant est très élevé pour toutes les méthodes. Cela peut avoir deux interprétations (pouvant se cumuler) : le minorant calculé et/ou les solutions retournés par nos méthodes se situent loin de la valeur optimale. La première interprétation est vraisemblable car la méthode de calcul du minorant reste basique et oublie le reste des espèces, diminuant fortement le nombre de secteurs nécessaires. La deuxième interprétation semble très plausible également, étant donné que les algorithmes sont proposés par des élèves de L1, dont les méthodes et connaissances sont encore limités et pas optimales .

Personnellement, ce projet nous a permis de développer nos compétences dans le langage Python, d'apprendre à passer de données CSV à des données exécutables dans des langages de programmation, à lier entre eux plusieurs codes et à passer d'une idée mathématique de résolution du problème à sa programmation. Nous avons progressé également dans la mise en place d'un formalisme pour la résolution d'un problème complexe ainsi que la manipulation du langage LaTeX, qui permet de fournir des documents plus organisés, interactifs, visuels et professionnels. La démarche de test nous a cependant permis d'évaluer la pertinence de nos algorithmes, leurs défauts, leurs points forts, leur marge d'amélioration et leur acuité en termes de complexité.

Une suite possible pour ce projet serait par exemple de relâcher la contrainte indiquant que les zones doivent être rectangulaires et imposer uniquement qu'elles soient connexes.

Références

[1] Adrien Brunel , Juliette Davret , Brice Trouillet, Nicolas Bez, Julie Salvetat, Antoine Gicquel, Sophie Lanco Bertrand. *Opening the black box of decision support tools in marine spatial planning : shedding*

light into reserve site selection algorithms for a balanced empowerment of stakeholders. Sciences de l'environnement.

[2] Marie Bonnin, Sophie Lanco Bertrand. *Chapitre 15. Forces et faiblesses des outils d'aide à la décision.* OpenEdition Books.

[3] Francçois Clautiaux (2023), Introduction à l'ingénieurie de l'optimisation.

A Annexe : détail des résultats

Pour cette annexe, nous avons décider de choisir 50 instances significatives parmi toutes celles proposées pour le projet, pour éviter une surabondance d'informations. Nous avons pris un échantillon ayant des instances de taille et type (île, archipel, péninsule etc) différents.

Table 3 – Résultats détaillés

Donnée	Valeur minorant	Heuristique de la Somme		Heuristique du meilleur Rectangle	
		Solution	Temps CPU (s)	Solution	Temps CPU (s)
a280	2375	**2808**	0	3157	8
berlin52	6073	22205	0	**8980**	0
bier127	92069	393989	0	135737	2
eil101	551	2062	0	803	1
eil51	372	1308	0	511	0
eil76	463	1969	0	642	1
gil262	1799	26298	0	3208	7
kroA100	15979	191387	0	27807	1
kroA150	20211	287844	0	33633	2
kroA200	21641	373938	0	35859	4
kroB100	16219	157190	0	29158	1
kroB150	19470	273239	0	34499	2
kroB200	22289	327456	0	36980	4
kroC100	15509	183466	0	26227	1
kroD100	15610	170990	0	26947	1
kroE100	15954	188351	0	27460	1
lin105	8568	36480	0	20356	1
lin318	26057	119872	0	54019	10
nrw1379	47889	712343	0	68964	188
pr1002	203159	349403	0	331103	149
pr107	21566	62752	0	46680	1
pr124	37828	98941	0	69297	2
pr136	80600	287028	0	120769	2
pr144	18604	93526	0	**61652**	2
pr152	40016	160980	0	**85699**	2
pr226	47693	110417	0	**94683**	8
pr2392	303944	**378032**	1	461170	556
pr264	32304	77977	0	**58023**	6
pr299	37993	83506	0	59890	9
pr439	73109	270646	0	131281	19
pr76	75098	150781	0	153462	1
rat195	1999	4030	0	2752	4
rat575	5701	12934	0	8605	31
rat783	7048	72134	0	11054	60
rat99	1042	2124	0	1554	1
st70	490	3410	0	830	1
ts225	112500	276540	0	**152493**	5
tsp225	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5
tsp226	3274	10349	0	5030	5