Aide à la Décision pour le Développement Durable II – placement optimal d'éoliennes – Projet IPIPIP - 2007-2008

Sophie Demassey

Il s'agit de développer un logiciel d'aide à la décision pour l'implantation d'un parc éolien hétérogène de productivité optimale.

Le rendement[1] d'une éolienne dépend à la fois de sa puissance et de son emplacement[2]. Certains sites peuvent d'ailleurs être très appropriés à l'élévation d'une éolienne (un col de montagne, un bord de mer,...) mais interdits à certains modèles d'éoliennes (trop grandes, trop fragiles, trop bruyantes,...).

Nous disposons pour la création de notre parc, d'un ensemble d'éoliennes de types tous différents, et nous avons identifié un ensemble de sites propices à l'installation de ces éoliennes. L'étude préalable de la puissance et de la turbulence du vent a permis d'estimer l'énergie que produirait chaque éolienne, en megawattheures (MWh), en moyenne par an, en fonction de son modèle et de l'emplacement choisi. Ces valeurs sont données pour chaque modèle (colonne) et pour chaque site (ligne) dans la table 1. Dans cette table, une case vide (–) indique qu'un modèle d'éolienne ne peut être installé sur un site donné.

	modèles d'éoliennes							
	V90T	V90M	G5X	G8X	E-40	E-66	E-70	E-112
sites	2MW	3MW	850kW	2MW	500kW	1.5MW	2.3MW	6MW
Col du Chvatàl	8000	–	3400	8080	2000	6000	9200	24000
Puits Dantzig	4600	–	1955	4646	1150	3450	5290	13800
Lac Erdös	–	10500	2975	7070	1750	5250	8050	21000
Berge Claude	–	11100	3145	7474	1850	5550	8510	6000
Source de Tarjan	4000	–	1700	4040	1000	3000	4600	12000
Pic Dijkstra	–	–	3825	–	2250	6750	–	–
Plateau Bellman	6000	–	2550	6060	1500	4500	6900	18000
Coupe Min	4400	–	1870	4444	1100	3300	5060	–
Forêt de Cayley	4000	–	1700	4040	1000	3000	4600	–
Pic Floyd	–	–	3740	–	2200	6600	–	–
Pont de Könisberg	–	10800	3060	7272	1800	5400	–	–
Tour d'Euler	7000	–	2975	7070	1750	5250	8050	–
Chemin Hamilton	4400	–	1870	4444	1100	3300	5060	13200
Mer de Warshall	–	18100	3995	9494	2350	7050	10810	28200
Arête de Kuhn	8000	–	3400	8080	2000	6000	–	–
Poste tié-chinois	6400	–	2720	6464	1600	4800	7360	19200
Marche de Prim	–	11100	3145	7474	1850	5550	8510	–
Arbre de Steiner	4400	–	1870	4444	1100	3300	5060	13200

TAB. 1 – Énergie moyenne annuelle produite (en MWh) par une éolienne en fonction du modèle et du site.

[1] Le rendement net comprend notamment aussi les coûts d'installation et de maintenance de l'éolienne et de son réseau électrique, mais on n'en tiendra pas compte ici.

[2] Une puissance nominale de 1MW (megawatt) signifie que l'éolienne produit 1MWh (megawattheures) d'énergie par heure lorsqu'elle atteint sa performance maximale. Afin de calculer sa production annuelle moyenne d'énergie, il faut connaître la distribution des vitesses du vent sur le site d'installation. Une éolienne de 1MW fonctionnant en moyenne 2300 heures par an à sa puissance maximale sur un site donné, produira 2300MWh d'énergie.

1 Le projet

L'objectif de notre logiciel est de **déterminer un emplacement possible pour chaque éolienne**, avec **au plus une éolienne par site**, de façon à **maximiser l'énergie totale produite**. Il s'agit d'un problème classique d'*optimisation combinatoire*, pour lequel un algorithme de résolution de complexité temporelle polynomiale (il est donc « rapide ») est connu.

Le logiciel à développer doit être :

- opérationnel et efficace : vous implémenterez et testerez l'algorithme évoqué ci-dessus à partir de la description complète qui vous sera fournie (vous aurez, dans une première étape, à en comprendre le fonctionnement) ;
- réutilisable : vous implémenterez une interface générique permettant à l'utilisateur d'entrer des données autres que celles de la table 1. Vous devrez assurer le « bon » fonctionnement du logiciel dans tous les cas ;
- fonctionnel : vous fournirez un affichage lisible de la solution trouvée.

2 Question subsidiaire

Le gestionnaire du parc souhaiterait agrandir son parc en achetant de nouvelles éoliennes, parmi les modèles décrits ci-dessus, à installer sur les sites encore libres. En confrontant les données de la table 1 et la solution trouvée, le logiciel devra être en mesure de proposer un plan d'achat de nouvelles éoliennes, permettant d'augmenter la productivité du parc. De combien l'énergie totale produite serait augmentée ? Résolvez au moyen de votre logiciel, ce nouveau problème augmenté (on dispose maintenant de plusieurs éoliennes pour un même modèle). Quelle aurait été l'énergie totale produite maximale, si l'ensemble de ces éoliennes avait été disponible au moment de la conception du parc ?

3 Estimation de la difficulté du projet

Algorithmique : ****
Modélisation du problème : ***
Modélisation objet : *
Programmation : **

IPIPIP : Placement optimal d'éoliennes
– 1ère partie –

Sophie.Demassey@emn.fr, B210

Ce document contient les premières indications pour réaliser votre projet IPIPIP : commençons donc par un peu d'algorithmique, avant de se lancer dans le code !

D'ici le prochain document (d'ici 2 à 3 semaines), tâchez de :
– comprendre l'exemple et les étapes de l'algorithme décrits ci-dessous
– rédiger l'algorithme

N'hésitez pas à poser toutes vos questions sur le forum pour partager ces informations. Au besoin, nous pourrons bien-sûr convenir de rendez-vous physiques.

1 Recherche d'un placement maximum

Avant de s'attaquer au problème général, i.e. placer les éoliennes de façon à produire le maximum d'énergie, on va commencer par un problème plus facile à résoudre : **placer le nombre maximum d'éoliennes**. En effet, il se peut que les éoliennes disponibles ne puissent toutes être placées simultanément : par exemple, si le nombre d'éoliennes est supérieur au nombre d'emplacements, ou si deux éoliennes n'ont qu'un seul, le même, emplacement possible.

Dans ce problème simplifié, la quantité d'énergie produite par les éoliennes n'est pas considérée. En revanche, on retiendra, pour chaque éolienne, les seuls sites possibles d'implantation.

Dans cette section, nous décrivons un algorithme qui assure de trouver un placement maximum des éoliennes : le principe de l'algorithme et le déroulement sur un exemple (1.1) puis les grandes étapes de l'algorithme (1.2). Il vous est demandé, à partir de ces informations, d'écrire l'algorithme dans le langage vu en cours, de l'implémenter, puis de le tester sur différents problèmes.

1.1 Exemple

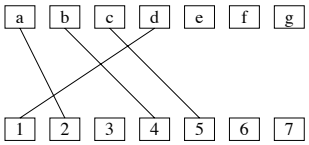
On dispose de 7 éoliennes (notées a, b, c, d, e, f, g) et de 7 emplacements (numérotés de 1 à 7). Chaque éolienne ne peut être placée que sur un sous-ensemble de sites :
a sur $\{2, 3, 4, 6\}$, b sur $\{2, 4, 5, 6\}$, c sur $\{2, 5\}$, d sur $\{1, 3, 5\}$, e sur $\{2, 7\}$, f sur $\{5, 7\}$, g sur $\{2, 5, 7\}$.

On cherche un *placement p valide* (c.a.d. tel que chaque site est occupé par au plus une éolienne) de *cardinalité maximale*.
Exemple : Le placement de a sur 2 et de b sur 4 est un placement valide, noté $p = (a2, b4)$, et de cardinalité 2. Les éoliennes c, d, e, f, g et les sites $1, 3, 5, 6, 7$ sont *libres* dans le placement p.

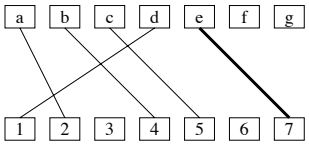
L'algorithme débute avec un premier placement valide (non maximum a priori.) À chaque itération de l'algorithme, on cherche à rajouter une éolienne libre, quitte à modifier l'emplacement des éoliennes déjà placées. À chaque itération donc, un nouveau placement valide est calculé, et ce placement contient une éolienne de plus que le placement de l'itération précédente. À une itération donnée, si aucune éolienne ne peut plus être ajoutée au dernier placement calculé, alors il est prouvé que ce placement est maximal et l'algorithme s'arrête.

Ce principe est illustré sur notre exemple :

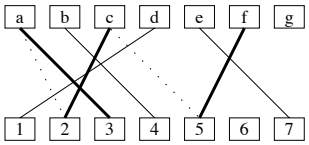
Initialisation : on calcule un premier placement valide, par exemple : $p = (a2, b4, c5, d1)$. Les éoliennes e, f, g sont encore libres.



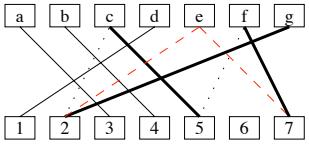
Itération 1 : on cherche à ajouter une éolienne libre au placement p.
On commence par évaluer l'éolienne e. e dispose de deux sites possibles : le site 2 est occupé par l'éolienne a, en revanche le site 7 est libre. On sait donc augmenter le placement p sans le « déranger » : le placement valide est maintenant $p = (a2, b4, c5, d1, e7)$.



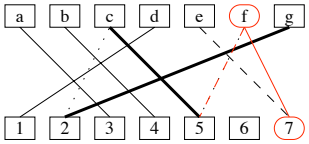
Itération 2 : on cherche à ajouter une éolienne libre (parmi f ou g) au placement p.
On commence par évaluer f. Tous les sites possibles pour f sont occupés : 5 par c et 7 par e. Afin de placer f, on sera donc amené à déplacer l'une des éoliennes c ou e, lui trouver à son tour un nouvel emplacement, et ainsi de suite... jusqu'à ce que l'on trouve un emplacement libre. Ici, par exemple, on essaie de placer f en 5 : l'éolienne c doit donc être déplacée. Tous les sites possibles pour c sont occupés (2 par a) ou réservés (5 par f). On essaie de placer c en 2 et donc de déplacer a. Or a dispose du site libre 3. On a donc trouvé à augmenter le placement en lui ajoutant l'éolienne f (sur le site réservé 5) et en déplaçant les éoliennes c et a (sur les sites réservés 2 et 3). Le nouveau placement est $(a3, b4, c2, d1, e7, f5)$.



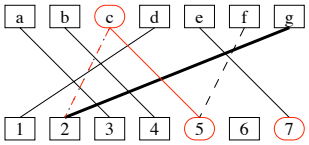
Itération 3 : on cherche à ajouter une éolienne libre (reste g) au placement p.
Tous les sites possibles pour g sont occupés $c2, f5, e7$. On teste $g2$ et on déplace c. On teste $c5$ et on déplace f. On teste $f7$ et on déplace e. Les seuls sites possibles pour e sont 2 (réservé à g) et 7 (réservé à f). Il est donc impossible de continuer au risque de boucler indéfiniment.



On revient alors progressivement sur nos décisions de réservation ($f7$ puis $c5$ puis $g2$), pour tester de nouvelles alternatives : On supprime la réservation $f7$ et on cherche pour f un autre déplacement possible. f ne peut être déplacée, ni en 7 (qu'on vient d'interdire), ni en 5 (réservé à c).

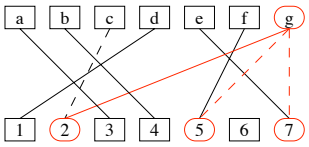


On supprime donc la réservation précédente, $c5$, et on cherche un nouveau site pour c. Encore une fois, il est impossible de déplacer c en 2, la seule alternative, car le site 2 est réservé à g.



On supprime donc la réservation précédente, $g2$, et on cherche un nouveau site pour g : 5 ou 7. Plutôt

que de tester ces deux alternatives, nous pouvons utiliser la propriété suivante (nous ne la prouverons pas ici) : comme les sites 5 et 7 ont déjà été réservés précédemment et comme aucune augmentation n'a encore été trouvée, alors aucune augmentation ne peut être obtenue non plus à partir de ces deux alternatives.



Ainsi, il n'est pas possible d'augmenter le placement p avec l'éolienne g. Autrement dit : placer g implique de libérer au moins l'une des éoliennes de p.

À cette étape, toutes les éoliennes libres ont été testées, sans succès. Le placement $p = (a3, b4, c2, d1, e7, f5)$ est donc maximal : au plus 6 éoliennes peuvent être placées simultanément.

1.2 Étapes de l'algorithme

Nous détaillons ici les différentes étapes de l'algorithme, que vous devrez formaliser, puis implémenter.

Il faut, pour commencer, définir un objet simple (tableau, liste,...) ou composé `p` modélisant le placement valide qui sera augmenté à chaque itération de l'algorithme. *À vous de le définir !*

Cet algorithme nécessite, par ailleurs, de définir deux objets *marqueurs* pour mémoriser le travail déjà fait et que l'on ne veut/doit pas refaire :

- un marqueur `test[x]` de type `boolean` pour chaque éolienne `x`, indiquant si une éolienne a ou non déjà été testée lors d'une itération (voir étape 1).
- un marqueur `res[y]` pour chaque site `y`, indiquant si le site a déjà été réservé par une éolienne, au moment de l'augmentation du placement (voir étapes 2-4). Afin de reconstruire le placement augmenté (voir étape 3), on ne mémorisera pas seulement un booléen, mais directement le numéro de l'éolienne. Si un site n'a pas encore été réservé, son marqueur sera initialisé, par exemple, à `-1`.

Initialisation. On construit un premier placement valide `p`.
À vous de jouer, vous avez le choix ! Vous pouvez par exemple commencer l'algorithme par le placement vide (aucune éolienne n'est placée) qui est naturellement valide. Vous pouvez aussi générer facilement d'autres placements valides.
On initialise le marquage des éoliennes (`test[x]=false` pour toute éolienne `x`) ; puis on passe à l'**étape 1**.

Étape 1. On cherche une éolienne libre non marquée.
Si aucune n'existe : **STOP** (le placement `p` est de cardinalité maximale).
Sinon, on choisit une telle éolienne `X` : on la marque (`test[X]=true`) ; on réinitialise les marqueurs des sites (`res[y]=-1` pour tout site `y`) ; puis on passe à l'**étape 2** avec `x=X`.

Étape 2. *Entrée : l'éolienne* `x` *que l'on cherche à (dé)placer.*
On choisit parmi les sites possibles de `x`, un site libre.
Si un tel site existe, disons `y`, on mémorise l'éolienne qui lui est réservée (`res[y]=x`) ; puis on passe à l'**étape 3**.
Sinon, on cherche un site possible de `x`, déjà occupé donc, mais non réservé `res[y]=-1`.
Si un tel site n'existe pas, alors on passe à l'**étape 4**.
Sinon, on en choisit un, disons `y`, on mémorise l'éolienne qui lui est nouvellement réservée (`res[y]=x`) ; on note `x` l'éolienne qui occupe le site `y` dans le placement actuel `p`, puis on passe à l'**étape 2**.

Étape 3. *Entrée : le site* `y`, *libre dans* `p` *que l'on va occuper.*
On augmente l'ancien placement `p`, en lui ajoutant l'éolienne `X` de l'étape 1, connaissant : le site `y` qui sera nouvellement implanté et le marquage `res`.
En vous aidant de l'exemple ci-dessus, voyez comment le placement `p` *peut être reconstruit (augmenté) à partir*

de ces seules informations : `y` *et* `res`.
On réinitialise le marquage des éoliennes (`test[x]=false`) ; puis on passe à l'**étape 1**.

Étape 4. *Entrée : l'éolienne* `x` *qui ne peut être déplacée.*
Si l'éolienne `x` est libre dans le placement `p` (i.e. `x` est l'éolienne `X` de l'étape 1) ; on passe à l'**étape 1**.
Sinon, on supprime la dernière réservation choisie : soit `y` le site occupé par `x` dans le placement `p`, on note `x=res[y]`, puis on passe à l'**étape 2**.

IPIPIP : Placement optimal d'éoliennes
– 2nde partie –

Sophie.Demassey@emn.fr, B210

Ce document contient la seconde indication pour réaliser votre projet IPIPIP : une description littérale de l'algorithme qui permet de calculer un placement des éoliennes de rendement énergétique maximal.

2 Recherche d'un placement de productivité maximale

L'algorithme vu précédemment permet de placer un **nombre maximal** d'éoliennes, mais il ne garantit pas que le placement ainsi calculé produise le **maximum d'énergie** possible. D'ailleurs, un placement de productivité maximale ne contient pas nécessairement le nombre maximum d'éoliennes :
Exemple : Soient 2 éoliennes a et b et 2 sites 1 et 2 tels que : placer a en 1 rapporte 100, placer b en 1 rapporte 300, placer b en 2 rapporte 100 et placer a en 2 est interdit. Alors le placement $p_1 = (a1, b2)$ est de cardinalité maximum 2 et de productivité 200. Tandis que le placement $p_2 = (b1)$ est de productivité maximale 300 mais sa cardinalité (1) n'est pas maximum.

Nous allons donc considérer un second algorithme dans le but de calculer un placement de productivité maximale. Cet algorithme ressemble au précédent : partant d'un placement valide (le placement vide), le placement est augmenté d'une éolienne (et d'un site) à chaque itération. Mais à la différence de l'algorithme précédent, où il suffisait de trouver une augmentation possible à chaque itération (étapes 2 à 4), il est nécessaire ici de trouver, parmi l'ensemble des augmentations possibles, celle qui augmentera le plus la productivité du placement.

Pour cela, nous avons besoin de mémoriser de nouveaux marqueurs :

- des marqueurs `test[x]` de type `boolean` pour chaque éolienne `x`, indiquant si une éolienne a ou non déjà été testée lors d'une itération ;
- des marqueurs `res[x]` (resp. `res[y]`) de type `int` pour chaque éolienne `x` (resp. pour chaque site `y`), indiquant le numéro du site (resp. de l'éolienne) qui lui est réservé. Si une éolienne (resp. un site) n'est pas réservée, son marqueur vaut `-1` ;
- des marqueurs `dual[x]` et `dual[y]` de type `double` pour chaque éolienne `x` et pour chaque site `y` ;[1]
- un marqueur `delta[y]` de type `double` pour chaque site `y`.

Remarque d'implémentation : *Les crochets `[]` ici participent à la notation de l'algorithme : ils ne désignent pas un tableau java. Par exemple, attention à la confusion entre `res[x]` et `res[y]` ! En effet, si vous implémentez ces marqueurs par un unique tableau `res`, et si les éoliennes et les sites sont chacuns numérotés à partir de `0` (ou de `1`), alors `res[0]` représentera à la fois le marqueur de la première éolienne et celui du premier site.*

On note `w[x][y]` la productivité que rapporte le placement de l'éolienne `x` sur le site `y`. Si un site `y` n'est pas possible pour une éolienne `x`, on pose `w[x][y]=0`. On a besoin également de considérer une valeur `double` constante `DMAX` arbitrairement grande.[2]

[1] Les marqueurs `dual` indiquent combien il est profitable d'intégrer une éolienne ou un site au placement courant.
[2] Pour l'implémentation en Java : chercher dans l'API de la classe `Double`, la valeur constante que vous pourriez utiliser : http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Double.html

2.1 Étapes de l'algorithme

Initialisation. Le premier placement valide p est **le placement vide**.
On calcule `W=max{w[x][y]}` pour toute éolienne `x` et tout site `y`. On initialise les marqueurs :
- pour chaque éolienne `x` : `res[x]=-1`, `dual[x]=W`, `test[x]=false`;
- pour chaque site `y` : `res[y]=-1`, `dual[y]=0`, `delta[y]=DMAX`.

On passe à l'**étape 1**.

Étape 1. On cherche une éolienne `x` non marquée (`test[x]==false`) telle que : ou bien `x` est libre dans le placement `p`, ou bien `x` est déjà placée dans `p` **et** réservée (`res[x] !=-1`). Si une telle éolienne n'existe pas, alors on passe à l'**étape 4**. Sinon, on marque l'éolienne trouvée (`test[x]=true`) ; puis on passe à l'**étape 2**.

Étape 2. *Entrée : l'éolienne* `x` *que l'on cherche à placer : on met à jour les marqueurs* `delta`.
On parcourt l'ensemble des sites `y` tels que : *(1)* `y` est un site possible pour `x`, *(2)* `y` n'est pas occupé par `x` dans le placement `p`, **et** *(3)* `delta[y]>dual[x]+dual[y]-w[x][y]`.

On va mettre à jour les marqueurs `delta[y]` pour chacun de ces sites jusqu'à ce que l'on rencontre un site `Y` qui soit libre dans le placement `p` et tel que `delta[Y]` soit nul. Dans ce cas précis, on passe alors directement à l'**étape 3** avec `y=Y`. Autrement, quand tous les mises à jour ont été effectuées et qu'aucun site `Y` n'a été rencontré, alors on repasse à l'**étape 1**. La procédure de mise à jour pour un site `y` est décrite ci-après.

Mise à jour à l'étape 2. *Entrée : un site* `y` *répondant aux critères de l'étape 2 pour une éolienne* `x` *donnée.*
- on diminue le marqueur `delta[y]=dual[x]+dual[y]-w[x][y]` ;
- on réserve le site `y` à l'éolienne `x` (`res[y]=x`) ;
- si `delta[y]==0` :
  - si `y` est occupée par une éolienne, disons `x'`, dans le placement courant `p`, alors on pose `res[x']=y` ;
  - sinon (`Y=y`), on a trouvé un nouveau meilleur placement incluant le site `y` : on sort de l'étape 2 puis on passe à l'**étape 3** ;

Étape 3. *Entrée : le site* `Y`, *libre dans* `p` *que l'on va occuper.*
On augmente l'ancien placement `p`, en lui ajoutant le site `Y` trouvé à l'étape 2. Le nouveau placement `p'` est construit, comme suit, en « remontant » le marquage `res` depuis `Y` :
(o) on pose `y=Y`, `p'=p`
(i) on pose `x=res[y]`
(ii) on rajoute dans `p'` le placement de l'éolienne `x` sur le site `y`
(iii) si `x` était libre dans l'ancien placement `p`, alors **STOP** (la construction du nouveau placement `p'` est terminée). Sinon, on pose `y` égal à l'ancien site occupé par `x` et on retourne à *(i)*.

Une fois, le nouveau placement construit, on réinitialise le marquage de toutes les éoliennes `x` et de tous les sites `y` (`test[x]=false` et `delta[y]=DMAX`), et on annule toutes les réservations (`res[x]=-1`, `res[y]=-1`) ; puis on passe à l'**étape 1**.

Étape 4. *Aucune éolienne ne peut être déplacée : on met à jour les marqueurs* `dual` *et* `delta`.
On calcule `d1=min{dual[x]}` pour toute éolienne `x` ; on calcule `d2=min{delta[y]}` pour tout site `y` tel que `delta[y]>0` ; On calcule `d=min{d1,d2}`. On met à jour les marqueurs, comme suit :
- pour toute éolienne `x` qui est libre dans `p` ou bien qui est réservée (`res[x] !=-1`) : `dual[x]=dual[x]-d;`
- pour tout site `y` tel que `delta[y]==0` : `dual[y]=dual[y]+d;`
- pour tout site `y` tel que `delta[y]>0` : `delta[y]=delta[y]-d;`

Si `d==d1` : **STOP** (le placement `p` est de productivité maximale).
Sinon, après cette mise à jour, il existe nécessairement de nouveaux sites `y` tels que `delta[y]==0`. Pour chacun de ces sites, on procède comme à l'étape 2 : dès que l'on rencontre un site, disons `Y`, qui soit libre dans le placement `p`, alors on sort prématurément de l'étape 4 pour passer à l'**étape 3**. Autrement, chaque site `y` considéré est actuellement occupé dans `p`, disons par une éolienne `x'`, et on pose alors `res[x']=y`. Quand tous les sites sont ont été considérés, on repasse à l'**étape 1**.

2.2 Tests et debuggage

On peut procéder de différentes manières pour s'assurer qu'un programme informatique est correct ou, dans le cas contraire, pour déceler les erreurs afin de les corriger.

Idéalement, on construit un programme petit à petit, méthode par méthode, en vérifiant que chaque nouvel élément implémenté s'exécute correctement. Ici, c'est un peu plus compliqué car l'algorithme forme un tout et on peut difficilement tester le fonctionnement de chaque étape indépendamment du reste. En revanche, une fois l'algorithme « brut » implémenté et testé, vous pourrez procéder à des améliorations de la lisibilité du code (emploi d'accesseurs, nommage compréhensible des méthodes et des variables, affichages et interfaces, etc.)

Avant cela, pour vérifier le code de l'algorithme, le mieux est de le **tester sur différents ensembles de données**, et de vérifier que chacun d'entre vous obtient bien les mêmes résultats, c'est à dire la même valeur de productivité maximale : créez des exemples simples de données, que vous pourrez vous échanger via le forum de Campus par exemple, et comparez vos résultats.

Par ailleurs, en cas de dysfonctionnement de votre programme, pensez à utiliser des **affichages pour déceler les parties erronées du code** : affichez par exemple, l'état du nouveau placement à chaque itération de l'algorithme (étape 3). Vous pouvez également faire un affichage au début de chaque méthode pour vérifier que le programme suit bien les étapes décrites dans l'algorithme, afficher l'état des variables et des marqueurs, etc.

Enfin, l'algorithme présent possède quelques **propriétés que l'on peut exploiter pour le test**. En effet à chaque itération, le placement construit à l'étape 3 vérifie obligatoirement les conditions suivantes :

- si une éolienne `x` occupe un site `y` dans le placement alors `dual[x]+dual[y]==w[x][y];`
- si un site `y` est libre dans le placement alors `dual[y]` est nul ;
- la somme des marqueurs `dual[x]` et `dual[y]`, sur toutes les éoliennes `x` et tous les sites `y`, est supérieure à la valeur de la productivité du placement.

Vous pouvez ainsi implémenter une méthode de test chargée de vérifier que toutes ces conditions sont réunies, après chaque augmentation du placement. Par ailleurs, le dernier placement construit (de productivité maximal donc) possède des propriétés supplémentaires que vous pouvez également vérifier à la toute fin du programme :

- si une éolienne `x` n'appartient pas au placement alors `dual[x]` est nul ;
- la somme des marqueurs `dual[x]` et `dual[y]`, sur toutes les éoliennes `x` et tous les sites `y`, est égale à la valeur de la productivité du placement.

Et si après ça, votre programme ne fonctionne toujours pas... pensez à demander de l'aide :
forum Campus, `sophie.demassey@emn.fr` ou bureau B210.