

# Formation toolbox pour le design et controle des micro-reseaux

Les méthodes de décision « optimale »

Corentin Boënnec

Groupe Genesys, Laplace, Université de Toulouse

17 mars 2025

# Rappels sur l'optimisation

## Cadre général

Un problème d'optimisation c'est :









- ▶ Des variables de décisions ;
- ▶ Un domaine de définition pour ces variables ( $\mathbb{R}$ ,  $\mathbb{R}^+$ ,  $\mathbb{N}$ , 0,1...);
- ▶ Des contraintes, fonction de ces variables.
- ▶ Un ou plusieurs objectifs, fonction de ces variables, à minimiser ou maximiser.

## Complexité des problèmes

Selon la forme des fonctions de contrainte et objectifs les méthodes de résolution disponibles sont différentes :

- ▶ Les méthodes de résolutions exactes (programmation mathématique, B&B, programmation dynamique ...). Elles garantissent l'optimalité de la solution retournée dans le cadre de la modélisation du problème.
- ▶ Les méthodes approchées (Métaheuristiques). Elle ne fournissent aucune garantie d'optimalité, cependant elles permettent d'adresser des modélisations plus complexes si le nombre de variables n'est pas trop élevé (de l'ordre de la centaine).

# Les catégories en programmation mathématique

	fonction-coût non-convexe		fonction-coût convexe		
Fonction-coût non-différentiable					
Fonction-coût différentiable					
		pseudo-convexe (unimodale)		quadratique	linéaire

## Classes de problèmes (source wikipédia) :

- ▶ L'optimisation linéaire étudie le cas où la fonction objectif et les contraintes sont linéaire. (**LP**)
- ▶ L'optimisation quadratique étudie le cas où la fonction objectif est de forme quadratique avec contraintes linéaires. (**QP**)
- ▶ L'optimisation non linéaire étudie le cas général dans lequel l'objectif ou les contraintes (ou les deux) contiennent des parties non linéaires, éventuellement non-convexes. (**NLP**)
- ▶ L'ensemble de ses classes de problème peut intégrer des variables entières (toute ou une partie) transformant ces problème en (Mixed Integer ...)

# Différents paradigmes d'optimisation (non exhaustifs)

## La programmation mathématique

- ▶ Exploite les propriétés mathématiques du problème pour proposer des méthodes de résolution dont l'optimalité est prouvée.
- ▶ Nécessite une formulation (modélisation) particulière et possiblement contraignante du problème.

## Métaheuristiques

- ▶ Explore l'espace de décisions en cherchant à améliorer sa/ses solution(s). Aucune garantie d'optimalité n'est possible.
- ▶ Nécessite seulement de pouvoir évaluer les solutions.

# Exemple de résolution par Prog. Math.

## Définition de l'espace des solution

L'ensemble de solutions admissibles (qui respectent les contraintes du problème) d'un problème borné dont les contraintes sont exprimées sous la forme d'équations linéaires et affines est un polytope convexe. Il est possible de montrer que la meilleure solution appartient à l'un de ses sommets.

## Résolution

Le simplexe parcourt les sommets (algébriquement), glissant d'arêtes en arêtes en vue d'améliorer la valeur de la fonction objectif. L'algorithme s'arrête dès lors qu'aucune arête partant du sommet actuel ne permet d'améliorer la fonction objectif.

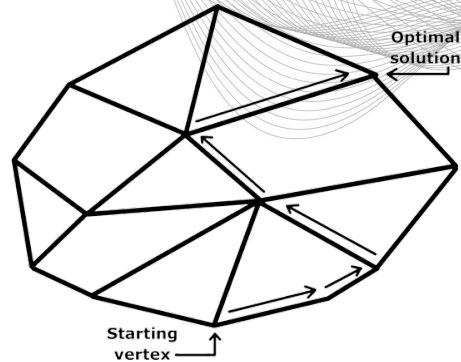


Figure – Exemple de résolution par le simplexe

## Note :

La méthode de résolution n'est possible que parce que le problème possède une forme particulière.

# Exemple de résolution par métaheuristiques

## Définition de l'espace des solution

L'ensemble de solutions admissibles n'est pas explicitement défini, on déclare simplement des bornes pour chacune des  $n$  variables de décisions définissant ainsi un  $n$ -orthotope (rectangle en dimension  $n$ ).

## Résolution

Durant un nombre d'itération, un temps donné, ou jusqu'à convergence, des solutions sont sélectionnées puis évaluées. La façon d'explorer l'espace des solutions dépend de la méthode utilisée.

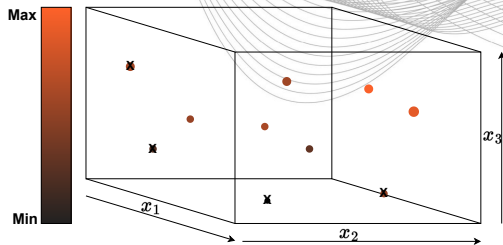
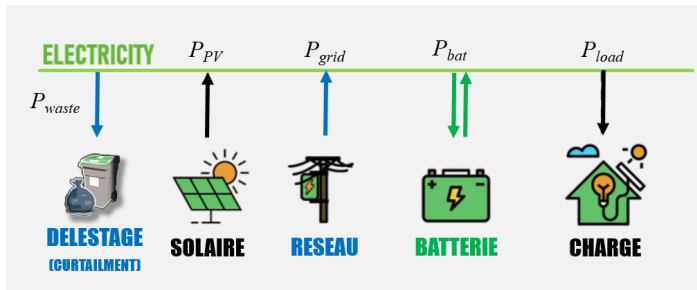


Figure – Exemple de résolution par métaheuristique

## Note :

Ces méthodes ne peuvent pas garantir d'avoir atteint la meilleure solution.

# Mise en situation



## Contexte :

On cherche à satisfaire une charge énergétique à chaque instant.

## Objectifs :

Parmi les ensembles de variables de décision permettant de satisfaire la charge, trouver celui qui optimise le/les critère de notre choix (prix, auto-consommation ...)

# Mise en situation

## Exemple avec l'opération du réseau simple

- ▶ Les variables de décisions sont un vecteur de longueur  $H$  représentant les flux de puissance de la batterie à chaque pas de temps ( $H$  pas de temps) ;
- ▶ Leur domaine de définition est  $\mathbb{R}^H$ . (Ce domaine peut être borné) ;
- ▶ Les contraintes sont les équilibres des flux de puissance ;
- ▶ L'objectif est par exemple la minimisation du coût d'opération du réseau ;



# Mise en situation 1

## Ajout au modèle 1

En réalité, le modèle doit inclure d'autres contraintes permettant de représenter le comportement de la batterie. Celle-ci possède une capacité et des bornes au sein desquelles elle peut être opérée. De plus, un lien entre ses états successifs d'heure en heure doit être formulé.

On peut rapidement considérer un modèle trivial de rendement de la batterie avec  $\eta = 0.9$ . On définit  $SoC_0 = 0.5$  l'état initial de charge,  $E^{rated}$  la capacité nominale de la batterie (ici fixée et constante) et  $\Delta_h$  l'intervalle entre 2 pas de temps (ici 1 heure).

$$SoC_{h+1} = SoC_h - \frac{P_h^{bat} \cdot \eta \cdot \Delta_h}{E^{rated}} \quad \forall h \in \{0 \dots H-1\} \quad (1)$$

$$0 \leq SoC_h \leq 1 \quad \forall h \in \{1 \dots H\} \quad (2)$$

Les équations de ce modèle sont toutes linéaires et les variables continues, on obtient alors un modèle **LP** (Linear Programming) qu'on pourra résoudre à l'aide du solveur *Cbc*.

# Mise en situation 2

## Ajout au modèle 2

Imaginons désormais que pour estimer le coût d'opération, on ajoute la notion de dépassement de la puissance réseau. Ce qui donc en cas de dépassement d'une valeur seuil  $P^{seuil}$  pour un pas de temps implique un coût  $c^{dépassement}$ .

On va alors avoir besoin d'une variable binaire  $\delta^{dépassement}$  pour chaque pas de temps afin de contrôler si la puissance demandée au réseau dépasse le seuil imposé.

Si l'on considère  $c_h^{elec}$  le coût de l'électricité acheté sur le réseau à l'heure  $h$ , le coût d'opération initialement calculé comme suit :

$$\sum_{h=1}^H P_h^{grid} \cdot c_h^{elec} \quad (3)$$

devient alors :

$$\sum_{h=1}^H P_h^{grid} \cdot c_h^{elec} + \delta^{dépassement} \cdot c^{dépassement} \quad (4)$$

Les équations de ce modèle sont toutes linéaires et les variables continues ou entières, on obtient alors un modèles **MILP** (Mixed Integer Linear Programming) qu'on pourra résoudre à l'aide du solveur *Cbc*.

Pour plus d'information quant à la gestion de la condition de dépassement voir la correction du TP2 et les contraintes relatives à la variables `m2[:dépassement]`

# Mise en situation 3

## Ajout au modèle 3

Oublions désormais cette histoire de dépassement et imaginons plutôt ici un rendement sous la forme

$$\eta = f(P) = aP + b$$

$$SoC_{h+1} = SoC_h - \frac{P_h^{bat} \cdot (aP_h^{bat} + b) \cdot \Delta_h}{E^{rated}} \quad \forall h \in \{1 \dots H - 1\} \quad (5)$$

$$0 \geq SoC_h \geq 1 \quad \forall h \in \{1 \dots H\} \quad (6)$$

Ainsi, on obtient un modèle avec des variables de décisions au carré et des variables continues, on parle de modèle **NLP** (Non Linear Programming) car ce n'est pas la fonction objectif qui est quadratique, mais des contraintes du problème.

# Mise en situation 4

## Ajout au modèle 4

Imaginons maintenant que la capacité de la batterie soit fonction de son utilisation passée à cause de son vieillissement mesuré par un indicateur *State of Health* :  $SoH_h = f(P_{1:h}^{bat})$ . On a alors :

$$SoC_{h+1} = SoC_h - \frac{P_h^{bat} \cdot \eta \cdot \Delta_h}{E^{rated} \cdot SoH_h} \quad \forall h \in \{1 \dots H - 1\} \quad (7)$$

Si on remplace  $SoH_h$  par son expression en fonction des  $P^{bat}$  de 1 à h on obtient un modèle avec des variables de décisions qui divisent d'autres variables de décisions. Cela donne un modèle **NLP** (Non Linear Programming).

# Mise en situation 5

## Ajout au modèle 5

Imaginons enfin que la capacité de la batterie soit fonction de son utilisation passée à cause de son vieillissement mesuré par un indicateur *State of Health* :  $SoH_h = f(P_{bat,1:h})$ . Mais que le calcul de ce SoH n'ai pas de forme fermée.

## définition : Forme fermée

Impossibilité de représenter son expression mathématique à l'aide d'une formule analytique finie, impliquant des opérations élémentaires telles que l'addition, la soustraction, la multiplication, la division, et les fonctions usuelles comme les exponentielles, les logarithmes, et les trigonométriques.

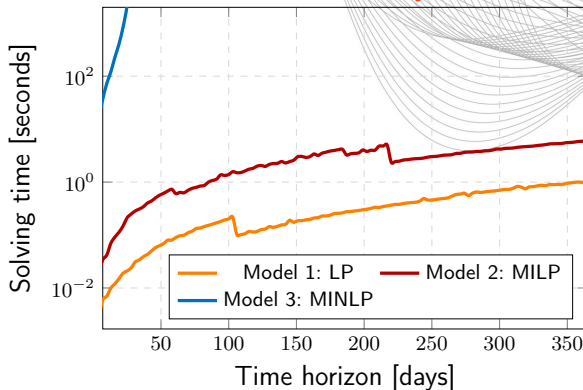
## Problématique et solution

On ne peut alors pas écrire ce modèle sous forme d'équation et la programmation mathématique ne peut donc adresser ce problème. Dès lors, il existe d'autres méthodes pour ce type de cas, notamment les méthodes à base de règle (RB).

D'autres méthodes acceptent tout type de modèle, car elle fonctionne via une évaluation successive de solution. Dès lors qu'il est possible de vérifier qu'une solution est réalisable et d'évaluer sa/ses valeur(s) objectif(s), cette classe de méthode est applicable.

# Impact de la complexité du modèle sur le temps de résolution

Pour illustrer l'explosion du temps de calcul voici un graphique montrant l'évolution du temps de résolution avec le Solver Gurobi pour les 3 première mises en situation. (Le modèle 3 inclut également les dépassements et est donc un problème **MINLP**).



## Note :

On notera que les temps de résolution sont donnés pour une solution optimale, mais qu'il est possible (généralement) de donner une limite de temps au solver pour retourner sa meilleure solution trouvée. On perd ainsi la garantie d'optimalité, mais on devient maître du temps d'exécution.

# Le problème complet

## Le deuxième niveau

En réalité, ce problème a 2 niveaux. Un niveau englobant de décision (le problème de design) dont l'évaluation est un problème d'optimisation (le problème de contrôle).

Selon les modèles utilisés, la résolution de ce type de problème peut impliquer une résolution très (voir beaucoup trop) longue.

## Les incertitudes

De plus, l'évaluation du problème est soumise à de fortes incertitudes, en effet, le contrôle optimal suppose par exemple :

- ▶ Le *perfect foresight* : c'ad que l'on connaît les variables imposées sur tout l'horizon temporel. Ce qui fait sur-performer le réseau. En effet, ce niveau d'optimalité est impraticable.
- ▶ Que le modèle est une parfaite description de la réalité, car la solution n'est optimal que dans le cadre décrit par le modèle. Ce qui n'est qu'une abstraction plus ou moins précise de la réalité.
- ▶ A cela s'ajoute l'incertitude en simplifiant certains aspects pour rendre la résolution plus rapide ou simplement possible (exemple : travailler sur une année représentative).
- ▶ ...

# Méthodes de résolution

## Choix des méthodes

Pour adresser le problème, il faut donc avoir conscience des hypothèses choisies et des limitations qu'elles apportent pour choisir une méthode d'optimisation en conscience.

- ▶ On peut soit utiliser des méthodes exactes dans une réalité qui en général sera plus déformée.
- ▶ Soit utiliser des méthodes approchées dans une réalité souvent un peu moins déformée, mais sans garanti d'optimalité.

## Conséquences implicites

- ▶ Le choix d'une méthode de programmation mathématique pour le design implique que le contrôle sera lui aussi résolu via la programmation mathématique.
- ▶ Le choix d'une modélisation complexe devra être compensé soit par des simplifications sur d'autres aspects soit par une méthode de résolution approchée.



# Les méthodes de résolution étudiées

## Le design

On étudiera durant la séance de TP deux méthodes principales :

- ▶ Les métaheuristiques
- ▶ La programmation mathématique

On s'intéressera ici principalement au design par métaheuristique via l'algorithme du clearing. On apercevra aussi la possibilité d'utiliser un algorithme multi-objectif.

## Le contrôle

- ▶ On étudiera le contrôle optimal afin de prendre en main l'outil puissant qu'est la programmation mathématique.
- ▶ On se concentrera sur les *Rule Base* qui sont certes sous optimales mais réalistes dans leur approche.

# TP2.1 - Introduction à la programmation mathématique

## Objectifs :

Découvrir la programmation mathématique et de percevoir les tenants et les aboutissants associés à ces méthodes.

Il ne s'agit pas de devenir spécialiste, mais d'être en mesure de déchiffrer ce qui s'y passe et de comprendre les difficultés associées.

## Exercices :

Les pages suivantes sont une suite d'exercices réalisant une progression vers la compréhension d'un modèle de programmation mathématique représentant le contrôle optimal du réseau.

# Exercice 1 : Assemblage de vélos

Dans une usine, une équipe d'ouvriers assemble des vélos cargos et des vélos standards : les vélos cargos (C), à raison de 100 modèles en 6 heures, et les vélos standards (S), à raison de 100 modèles en 5 heures.

- ▶ Chaque semaine, l'équipe fournit au maximum 60 heures de travail.
- ▶ Tous les véhicules sont ensuite garés sur un parking qui est vidé chaque week-end, et dont la surface fait  $1500m^2$ . Un vélo cargo C occupe  $2.5m^2$ , tandis qu'un vélo standard S occupe  $1m^2$ .
- ▶ De plus, il ne faut pas assembler plus de 700 vélos cargos C par semaine, car les ressources nécessaires aux batteries sont limitées. En revanche, les vélos standards S, ne sont pas limités par l'approvisionnement en ressources.
- ▶ Enfin, la marge (différence entre le prix de vente et le coût de production) vaut 700€ pour un vélo cargo et 300€ pour un vélo standard S.

L'usine souhaite savoir comment répartir le travail entre les deux modèles de vélos pour que la marge totale soit la plus grande possible.

## Instructions

- ▶ Définissez les décisions à prendre ainsi que leur domaine de définition.
- ▶ Comment les décisions sont-elles contraintes ? Écrivez ces contraintes.
- ▶ Comment peut-on exprimer la marge en fonction des variables de décisions ?

## Exercice 2 : Affectation de tâches

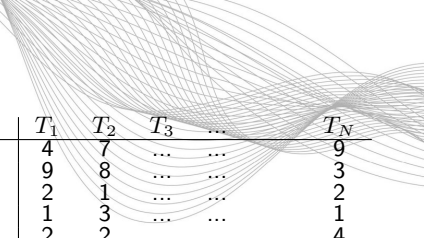
Une manageuse doit préparer le planning de son équipe de  $N$  personnes. Durant la journée, il y a  $N$  tâches à effectuer.

Chaque tâche doit être affectée exactement une fois et chaque personne doit effectuer exactement une tâche.

Chaque membre de l'équipe a fait part de ses préférences quant aux différentes tâches, qui se traduit par un score de préférence noté sur 10, comme illustré en figure 3.

Ainsi,  $c(i, j)$  correspond au score de préférence de la personne  $P_i$  pour la tâche  $T_j$ .

La manageuse souhaite trouver la meilleure affectation possible.



	$T_1$	$T_2$	$T_3$	...	$T_N$
$P_1$	4	7	...	...	9
$P_2$	9	8	...	...	3
$P_3$	2	1	...	...	2
...	1	3	...	...	1
...	2	2	...	...	4
$P_N$	4	3	...	...	2

Figure – Exemple de matrice de préférences

### Instructions

- ▶ Définissez les décisions à prendre, précisez si elles appartiennent à  $\mathbb{R}$ ,  $\mathbb{N}$ ,  $\{0,1\}$ ...
- ▶ Comment les décisions sont-elles contraintes ? Écrivez ces contraintes.
- ▶ Comment exprimer le contentement total de vis à vis de l'affectation des tâches ?
- ▶ Établissez un problème MILP pour modéliser cette situation.

## Exercice 3 : modélisation et optimisation d'une journée.

### Variables de décisions

Les puissances d'entrée/sortie de la batterie.  $p_{ch}^i$  et  $p_{dch}^i$  avec  $i$  de 1 à 24. (les valeurs sont positives)

### Variables d'ajustement

Les puissances demandé et fourni au réseau.  $p_{in}^i$  et  $p_{out}^i$  avec  $i$  de 1 à 24. (les valeurs sont positives)

### Variable d'environnement

La charge électrique et la production du panneau solaire.  $p_{gen}^i$  et  $p_{load}^i$  avec  $i$  de 1 à 24. (les valeurs sont positives)

### Équilibre des puissances

$$p_{load}^i - p_{gen}^i + p_{ch}^i - p_{dch}^i - p_{in}^i + p_{out}^i == 0 \quad \forall i \in \{1..24\}$$

## Exercice 3 : modélisation et optimisation d'une journée.

### Variables d'état

On va pour la batterie ajouter une variable d'état dont le rôle est le contrôle de la charge de la batterie.  $SoC^i$  avec  $i$  de 1 à 25 ( $=24+1$ ).

### Contraintes sur la dynamique des états

$$SoC^{i+1} == SoC^i \cdot (1 - \eta_{self}) - \frac{(p_{dch}/\eta - p_{ch} \cdot \eta) \cdot \Delta h}{E_{rated}} \quad \forall i \in 1..24 \quad (8)$$

Avec :  $SoC^1 == 0.5$  et  $SoC^{25} \geq SoC^1$

### Contraintes sur les bornes

Comme on contrôle l'état de la batterie on peut formuler des contraintes sur ses bornes :  $0 \leq SoC^i \leq 1 \quad \forall i \in \{1..24\}$  On peut également contraindre la puissance de (dé)charge à ne pas dépasser certaines valeurs  $p_{ch}^i, p_{dch}^i \leq P_{max} \quad \forall i \in \{1..24\}$

# Initiation à JuMP

L'intérêt de formuler les problèmes de cette façon est qu'un solveur va pouvoir les adresser de façon efficace car ils ont été mis dans une forme qu'il comprend. Vous trouverez la documentation du package et la liste de compatibilité des solveurs aux adresses suivantes : Manuel JuMP , Compatibilités des solveurs

## Instructions

Utiliser la syntaxe de JuMP pour implémenter les 3 problèmes précédents. Le script TP2 contient des données pour que vous ayez simplement les modèles à écrire.

## note :

Vous pouvez utiliser les corrections pour tester des variations. N'hésitez pas à comparer vos propositions avec les solutions du TP.

## TP2.1 Comparaison de méthode

### Instructions

Comparer les résultats obtenus par le contrôle **LP** (modèle 1) et le contrôle **RB** sur 1 an.

### Note :

On rappelle que la programmation mathématique trouve une solution en ayant une parfaite connaissance du future (*perfect foresight*) tandis que le contrôleur RB dévoile les données au fur et à mesure.

Par conséquent, dans le cadre de ce qui est modélisé la programmation mathématique fournit une borne (optimiste) à ce qui est possible d'obtenir comme solution.



# TP 2.2 Le design

## Aperçu des méthodes

Les deux approches principales pour la prise des décisions de design sont :

- ▶ La programmation mathématique où les décisions de design sont introduites comme variable du modèle pour construire un plus grand modèle intégrant *Design* et *Contrôle*.
- ▶ Les métaheuristiques où les décisions de design sont prise dans une boucle extérieur et évaluée à l'aune d'un contrôleur. La prise de décisions de contrôle et l'évaluation du système global forment alors une boucle intérieure dont le rôle est de comparer les décisions de design prisent dans la boucle extérieure.

## Objectif

- ▶ Programmation Mathématique : comprendre à travers un cas simple la mise en place de la co-optimisation contrôle/design.
- ▶ Métaheuristique : Être en mesure de modifier la boucle d'évaluation interne permettant d'évaluer un design.

## TP 2.2 Implémentation en programmation mathématique

### Instructions

Reprendre le modèle du TP2.1 modèle 1

- ▶ Y ajouter des variables de décision (positives) pour le design  $E_{rated}$  et  $P_{PV}$ .
- ▶ Changer, (comme dans la correction), le SoC de la batterie pour qu'il soit défini de 0 à  $E_{rated}$  plutôt que de 0 à 1.
- ▶ Insérer les nouvelles variables dans les équations du modèle.
- ▶ Modifier la fonction objectif pour y ajouter le coût d'investissement dans la batterie et le panneau solaire.
- ▶ Exécuter le modèle sur 1 an. Faire varier les prix et observer les valeurs des coûts d'investissement.
- ▶ Réfléchir aux aspects principaux absents de la modélisation proposée.

# Résultats

## Commentaire :

Les éléments principaux permettant de corriger le problème de design sont :

- ▶ L'ajout d'une valeur dite de *salvage* qui consiste à attribuer une valeur aux éléments du réseau à la fin de l'horizon temporel.
- ▶ L'ajout d'un contrôle de l'état de santé des composants pour déterminer leur valeur de *salvage*.
- ▶ Un horizon temporel plus long peut aussi permettre d'amortir l'investissement bien que la valeur de *salvage* prend en compte cet aspect.

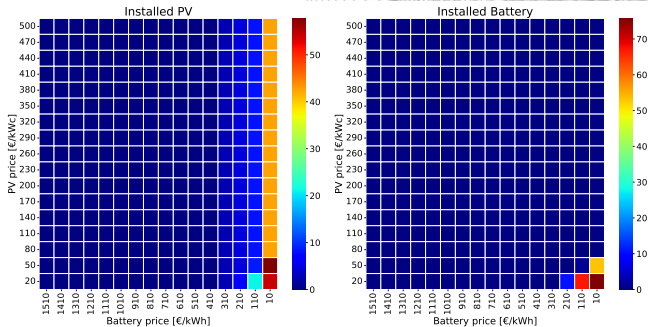


Figure — Résultat du design optimal en faisant varier le prix au kW pour les batterie et PV.

Ces résultats viennent du fait que l'on a oublié plusieurs éléments de modélisation. Il ne faut donc pas perdre de vue que les solutions optimales le sont dans le cadre que l'on a défini par notre modélisation.

# Détails sur les Metaheuristiques

## Le Clearing et ses caractéristiques

- ▶ Mono-objectif - Une seule valeur est utilisée pour comparer des solutions.
- ▶ Nichage dans l'espace de décisions - On cherche à conserver la diversité des solutions, on va pénaliser les solutions qui sont trop ressemblantes à une autre.

Il est toujours possible d'adresser plusieurs objectifs et contraintes via un algorithme mono-objectif en utilisant des poids. Plusieurs quantités peuvent être agglomérées via une somme pondérée. De la même façon, des pénalisations peuvent être appliquées pour inciter à suivre une contrainte.

## Le NSGAI et ses caractéristiques

- ▶ Multi-objectif - Plusieurs valeurs sont retournées lors de l'évaluation d'une solution et toutes sont prises en compte de façon équivalente et distincte.
- ▶ Nichage dans l'espace des objectifs - On cherche à conserver la diversité des solutions, on va pénaliser les solutions dont l'évaluation ressemble trop à une autre.

L'utilisation de plusieurs objectifs rend la désignation d'une solution comme solution optimale impossible. On va alors chercher à retourner l'ensemble des solutions dites non dominées. CàD dont on est pas capable de trouver une solution qui serait meilleure sur au moins un objectif sans être moins bonne sur aucun des autres objectifs. Cet ensemble de solutions s'appelle **Front de Pareto** et est composé de l'ensemble des solutions dites non dominées au sens de Pareto.

# Design par le Clearing

## Syntaxe

La syntaxe d'utilisation du design par la métaheuristique (voir ci dessous) est assez lourde car elle possède beaucoup de paramètres et options. Nous allons ici les décortiquer pour mieux comprendre.

```
designer = initialize_designer!(microgrid,  
    Metaheuristic(options = MetaheuristicOptions(;method = Clearing(nind = 50),  
        multithreads=false,  
        iterations = 10,  
        controller = RBC(options = RBCOptions(policy_selection = 103))),  
     $\omega$ , #scénario de donné.  
    ub_var, # Vecteur de borne supérieure des décisions  
    lb_var, #Borne inférieure  
    varID; # Dictionnaire pour lier décisions et composants  
    f_obj = fobj_cours) #Fonction d'évaluation des solutions (Optionnel : fobj2 par défaut)
```

## Paramètres de la fonction

- ▶ microgrid : Le micro réseau concerné ;
- ▶ Metaheuristic : Le designer (et ses options) ;
- ▶  $\omega$  : Le scénario de données ;
- ▶ Les paramètres suivants sont décrits dans les commentaires.

## Options de Metaheuristic

- ▶ method : Le Clearing et sa pop de `nind = 50` individus ;
- ▶ multithreads : L'activation du parallélisme ;
- ▶ iterations : Le nombre de générations ;
- ▶ controller : Le controller utilisé ;

# TP 2.2 prise en main des metaheuristiques

## Instructions 1 : Choix du controller

- ▶ Utiliser le designer pour dimensionner le réseau simple du TP1 ;
- ▶ Réaliser l'opération pour plusieurs RB ;
- ▶ Interpréter les résultats au regard du controller choisi.

## Instructions 2 : Choix de la fonction objectif

- ▶ Réaliser, sur la base de la fonction fourni, une fonction objectif de votre choix afin d'évaluer chaque design.
- ▶ Proposer une solution pour intégrer à cette fonction une contrainte. (par exemple sur l'autonomie du système)

## Note :

- ▶ L'algorithme maximise la fonction objectif. Si vous cherchez à minimiser  $Max(x) = -Min(-x)$ .
- ▶ Vous pouvez utiliser `metrics.cost.total` ou `metrics.npv.total` qui se construisent comme  $Capex + Opex - Salvage$ .
- ▶ Vous trouverez également la part de renouvelable dans `metrics.renewale_share`
- ▶ Pour plus de détails cherchez `Metrics` dans le mode *Help* de la console.

## TP 2.2 prise en main des metaheuristiques

### Rappel :

Le Clearing effectue un nombre d'itération défini par l'utilisateur, durant lesquelles il évalue `nind` solution sur `iterations` (auxquelles il faut ajouter l'initialisation de la population). Le nombre d'évaluation peut alors être obtenu comme :

$$N_{eval} = nind \times (iterations + 1) \quad (9)$$

### Instructions 3 : Analyse sur le temps d'exécution

- ▶ En reprenant les instructions permettant la simulation du microgrid au TP1, et à l'aide de la macro `@time`, faites une estimation du temps requis au design pour fournir une réponse en fonction des paramètres.
- ▶ Qu'en est-il si l'on utilise un controller faisant appel à la programmation mathématique ? `solve_time(m)` donne le temps de résolution d'un modèle m.