# Algorithme d'identification de contraintes incompatibles pour les problèmes d'optimisation : application à un projet énergétique

**Lou Morriet*[1,2], Benoit Delinchant[1], Gilles Debizet[2], Frédéric Wurtz[1]**

**[1] Univ. Grenoble Alpes, CNRS, Grenoble INP**, G2Elab, 38000 Grenoble, France**
**Bâtiment GreEn-ER, 21 Avenue des Martyrs, CS 90624, 38031 GRENOBLE CEDEX 1**
**[2] Univ. Grenoble Alpes, CNRS, Sciences Po Grenoble***, Pacte, 38000 Grenoble, France**
**Cité des territoires, 14 Avenue Marie Reynoard, 38100 Grenoble**
***lou.morriet@grenoble-inp.fr**

*RESUME. La conception des projets énergétiques évolue vers une conception multi-acteurs. Les parties prenantes doivent alors s'accorder sur le projet, et plus particulièrement le modèle du projet, afin de permettre sa mise en place. Nous entendons par modèle, le modèle physique ainsi que les contraintes et les objectifs. Les contraintes jouent un rôle essentiel dans la définition des projets et peuvent même rendre un problème « infaisable », i.e. sans solution, s'il y a des contraintes incompatibles. C'est pourquoi il est important de pouvoir identifier ce type de contraintes. Les acteurs pourront alors s'accorder pour relaxer des contraintes afin de trouver des solutions au problème. Nous avons développé en open source et en python l'algorithme LPFICS d'identification de contraintes incompatibles pour les solveurs MILP. Dans cet article, cet algorithme est comparé à l'algorithme proposé par le solveur propriétaire Gurobi.*

*MOTS-CLÉS : identification de contraintes incompatibles, optimisation MILP, aide à la négociation*

*ABSTRACT. Energy project design becomes multi-stakeholders design. Stakeholders need thus to find an agreement on the project and more specifically on the model of the project. We consider a model as the physical model but with its constraints and objectives. Constraints are highly important in the design of the project as they can deal to an "infeasible" problem without solutions, if there are incompatible constraints. That is why it is important to identify these constraints. Stakeholders could thus relax some constraint to find a feasible solution. We developed an open source algorithm LPFICS in Python to identify incompatible constraints for MILP solvers. This article presents the algorithm and a comparison with the proprietary solver Gurobi.*

*KEYWORDS : Incompatible constraint identification, MILP optimisation, negotiation support algorithm*

## 1. INTRODUCTION

L'atténuation des émissions vis-à-vis du changement climatique nous amène à développer de plus en plus de systèmes énergétiques locaux bas-carbone. **Dans ce cadre on observe une augmentation des projets multi-parties prenantes dans la conception des projets** (IAU IdF et Vaisman 2018). De nombreux outils d'aide à la conception ont été développés pour répondre aux contraintes des systèmes énergétiques locaux bas-carbone et notamment l'intermittence des sources de production (Pajot et al. 2019). Cependant, les outils développés présupposent qu'un acteur unique assure non seulement la conception du système mais aussi décide de son fonctionnement ; ce qui n'est majoritairement pas le cas si le projet est multi-parties prenantes. **La multiplicité et l'influence forte des choix pour le modèle rendent la phase de modélisation cruciale pour les acteurs.** Crozier confirme cette hypothèse en expliquant que les négociations d'un problème portent « sur la définition du problème (…). Car les



** Institute of Engineering Univ. Grenoble Alpes
*** School of Political Studies Univ. Grenoble Alpes

partenaires savent trop bien qu'une fois cette définition imposée, l'orientation de la décision aura déjà été très fortement structurée. » (Crozier et Friedberg 2014, 364). **Il nous semble alors nécessaire d'aider les acteurs à s'accorder sur la définition modèle et notamment en phase de pré-étude,** phase où les inconnues sont nombreuses et les choix déterminants pour le projet. **Nous considérons que le modèle ne se restreint pas au modèle physique mais inclus également les contraintes et les objectifs**. La définition des objectifs est souvent mise en avant dans la conception des projets, et notamment avec le développement de travaux sur l'optimisation multicritère (Recht 2016; Raad 2017). Cependant, les contraintes peuvent compromettre le développement d'un projet, a fortiori dans un contexte multi-parties prenantes. En effet, en phase de pré-étude, les acteurs ont peu de connaissances sur ce qu'il est possible de faire ou non. Chaque acteur aura tendance à préserver ses intérêts en émettant des contraintes fortes, qui additionnées aux contraintes des autres acteurs, rendent le projet sans solution (Morriet, Debizet, et Wurtz 2019). **Il est donc nécessaire de pouvoir identifier les contraintes qui rendent le problème infaisable, comme nous le proposons dans cet article**. Les acteurs pourront alors s'accorder pour relaxer des contraintes afin de trouver des solutions au problème.

Une première partie présente rapidement des outils qui nous semblent adaptés au développement de systèmes énergétiques locaux bas-carbone multi-parties prenantes en phase de pré-étude. Une deuxième partie présente des méthodes d'identification de contraintes incompatibles, que nous avons ensuite appliquées, dans une dernière partie, à la modélisation d'un projet énergétique.

## 2. OUTILS POUR UNE NEGOCIATION EFFICACE AUTOUR DES MODELES DE PROJETS ENERGETIQUES

### 2.1. LE RECOURS A UN META-MODELEUR MILP ORIENTE OBJET ET OPEN-SOURCE POUR MODELISER DES SYSTEMES ENERGETIQUES LOCAUX BAS-CARBONE : OMEGALPES

**Afin que les parties prenantes puissent s'accorder sur le modèle, il est nécessaire que le modèle soit compréhensible.** L'architecture dirigée par les modèles (Kleppe, Warmer, et Bast 2003) vise ainsi à différentier le modèle construit par les acteurs, décrit en langage de haut niveau compréhensible, du modèle adapté aux solveurs décrit en langage bas niveau et moins compréhensible. La programmation orientée objet permet d'utiliser une sémantique proche de celle utilisée par les parties prenantes pour décrire la réalité technique (quel sous-système ?) et sociale (qui est responsable ?). De plus, **il est nécessaire de pouvoir modéliser et modifier le modèle rapidement et plusieurs fois pour évaluer les propositions soulevées lors des négociations entre les parties.** Il est pour cela utile de définir une bibliothèque de briques de modèles préconstruites et modifiables, permettant de construire le modèle. L'héritage proposé par la programmation orientée objet rend la modélisation et modification des briques flexible et agile**. Les outils répondant aux caractéristiques présentées ci-dessus sont appelés méta-modeleurs.**

**Par ailleurs, le modèle devant être modifié à de nombreuses reprises il est nécessaire de pouvoir résoudre le problème rapidement, malgré les nombreuses variables et contraintes du modèle.** Ce nombre important est lié à la fois à la modélisation du système en phase d'exploitation, nécessaire en cas d'intermittence, et à la phase de pré-étude, étape où de nombreuses valeurs ne sont pas encore fixées. Une recherche de solutions par essai-erreur sur la base de nombreuses simulations manuelles ne permet alors pas d'identifier une solution optimisée rapidement. **C'est pourquoi les outils d'optimisation semblent appropriés.** L'optimisation linéaire nous semble également la plus pertinente car elle permet

une résolution rapide pour un grand nombre de variables et contraintes. De plus, pour une modélisation plus proche de la réalité, par exemple l'état éteint ou allumé d'un système énergétique, il semble nécessaire de choisir une modélisation Mixed Integer Linear Programming (MILP) (Pajot 2019). Cette modélisation s'écrit sous la forme de contraintes et d'objectifs appliquées à des variables :

$$\begin{aligned} &Minimiser\ c^T\ x\ +\ d^T\ y \\ &Selon:\ Ax\ +\ By\ \leq\ b \\ &x_{min}\ \leq\ x\ \leq\ x_{max}\ ,\ y \in \{0,1\}^m \end{aligned} \quad \textbf{(1)}$$

*Avec* :

$c \in IR^n,\ d \in IR^m,\ b \in IR^p$ vecteurs de données

$x \in IR^n,\ y \in \{0,1\}^m$ vecteurs de variables

$x_{min} \in IR^n,\ x_{max} \in IR^n$ vecteurs des limites basses et hautes des variables

$A \in IR^{p*n}\ and\ B \in IR^{p*m}$ matrices définissant les contraintes associées aux variables

Enfin, dans un contexte multi-parties prenantes, il est nécessaire d'avoir recours à des outils et méthodes open-source. En effet, cela permet d'assurer une transparence sur la modélisation vis-à-vis des parties prenantes du projet, qui pourront potentiellement faire des vérifications.

Plusieurs méta-modeleurs répondent aux critères ci-dessus sont identifiés dans (Pajot et al. 2019). Développé en Python, **OMEGAlpes** propose des unités énergétiques préconstruites intégrant des paramètres, contraintes et objectifs prédéfinis. Il propose également une modélisation « acteur » avec des contraintes et objectifs associés, selon si l'acteur applique une règle ou s'il est opérateur d'un système énergétique (Morriet, Debizet, et Wurtz 2019). C'est pourquoi nous l'avons choisi.

### 2.2. LE RECOURS AU PROJETEUR PULP ET AU SOLVEUR CBC

Pour passer de la modélisation objet, langage de haut niveau et compréhensible, au langage adapté au solveur, langage de bas niveau, il est nécessaire d'utiliser un « projeteur ». Pyomo (Hart, Watson, et Woodruff 2011) et **PuLP** (Mitchell 2019) sont deux projeteurs adaptés à une modélisation MILP. Ils sont open-source et développés en Python. OMEGAlpes se base sur le projeteur PuLP.

De nombreux solveurs open-sources ont été développés pour les problèmes MILP : CBC (COIN-OR Foundation 2020), GLPK (Free Software Foundation 2012) ou encore LpSolve (Peter et Eikland 2020). Bien que PuLP puisse utiliser plusieurs solveurs grâce à la création d'un fichier .lp contenant le problème décrit en langage bas niveau, **CBC** étant le solveur directement intégré à PuLP, nous l'utilisons.

## 3. ALGORITHME D'IDENTIFICATION DE CONTRAINTES INCOMPATIBLES ET CATEGORISATION DES CONTRAINTES

### 3.1. ALGORITHMES D'IDENTIFICATION DE CONTRAINTES INCOMPATIBLES

Une première méthode existe pour les solveurs conservant les pré-valeurs des variables dans le cas où le modèle est sans solution. Dans le cas d'un problème sur-contraint, il existe au moins une variable dont la valeur ne peut pas respecter plusieurs contraintes. Les contraintes incompatibles peuvent ainsi être identifiées. Cette méthode ne peut pas s'appliquer à tous les solveurs, car ils ne conservent pas tous des pré-valeurs pour les variables. En effet, ce n'est pas le cas du solveur CBC.

Gurobi propose la méthode computeIIS(), IIS signifiant « Irreducible Inconsistent Subsystem ». Le sous-système IIS, identifié par cette méthode, est un ensemble de contraintes et de variables tel que : le

problème est un système infaisable, et si l'on retire une des contraintes de ce sous-système, il devient faisable (Gurobi Optimization, Inc. 2017, 458-59). Cependant, du fait du caractère propriétaire de l'outil, la méthode n'est pas disponible en open-source. Une méthode similaire est présentée théoriquement sous le nom de QuickXplain par Junker (Junker 2004) et Rodler (Rodler 2020) et a été développée, sous le nom de Ruby-Cbc, dans le langage Ruby par Guillaume Verger sous licence MIT (gverger 2016). La méthode Ruby-Cbc n'était pas compatible avec nos développements car elle intègre directement le solveur CBC, ce qui ne permettait pas de faire la jonction avec PuLP et les autres solveurs. Nous avons donc choisi de traduire l'algorithme en langage Python. Afin de respecter la licence MIT, notre développement reprend cette licence ainsi que le nom de l'auteur. **La méthode développée, nommée LPFICS (Linear Problems : Find Infeasible Constraint Sets), est disponible en open-source**[1]**.** Voici les étapes dans les grandes lignes :

1. Diviser les contraintes du modèle en deux groupes égaux (possiblement à une contrainte près)
2. Optimiser le modèle avec le premier groupe, s'il existe une solution avec le premier groupe, optimiser le deuxième groupe.
    a. Si un des deux groupes n'a pas de solution identifiée recommencer à partir de l'étape 1 jusqu'à trouver un problème insolvable à deux contraintes.
    b. Si les deux groupes ont tous les deux une solution, changer l'ordre des contraintes du groupe précédent et recommencer à l'étape 1.
    c. Si aucun sous-groupe à deux contraintes n'a pas pu être identifié, récupérer le groupe précédent et recommencer à l'étape 1 en cherchant un groupe de contraintes à une variable de plus jusqu'à identifier le plus petit sous-groupe.

Il est à noter que, comme pour la méthode proposée dans Gurobi, le modèle initial peut avoir plusieurs sous-systèmes de type IIS mais seul un IIS est identifié. Il est possible qu'une fois les contraintes identifiées relevées, le modèle reste sur-contraint. Il faut alors réitérer le processus.

### 3.2. DISTINGUER LES CONTRAINTES DE DEFINITION, LES CONTRAINTES TECHNIQUES ET CELLES LIEES AUX ACTEURS

Toutes les contraintes ne sont pas destinées à être négociées. Il peut donc y avoir une **hiérarchisation entre les contraintes** (Junker 2004) : **celles qui sont négociables, celles qui le sont moins et celles qui ne le sont pas**. Dans le cadre des projets énergétiques, nous proposons trois catégories avec des exemples intégrés dans le méta-modeleur OMEGAlpes[2] :

- les **contraintes de définition et physiques (DefinitionConstraint (DCte))** qui ne sont pas négociables. Nous entendons par contrainte de définition une relation mathématique imposée (par la physique par exemple) reliant les variables entre elles. Il peut s'agir de calculer l'énergie totale (e_tot) comme le produit entre la somme des puissances (p) à chaque pas de temps et le pas de temps (contrainte : calc_e_tot, cf. (2)).

$$e_{tot} = \sum_{t} p[t] * \Delta T \quad \textbf{(2)}$$

[1] https://gricad-gitlab.univ-grenoble-alpes.fr/omegalpes/lpfics
[2] https://gricad-gitlab.univ-grenoble-alpes.fr/omegalpes/omegalpes

- les **contraintes techniques (TechnicalConstraint (TCte))** qui peuvent être négociées par les acteurs et qui sont à différencier des contraintes liées aux acteurs. Il peut s'agir d'une centrale de production dont la mise en route doit respecter une rampe d'allumage. La puissance de démarrage augmente alors respectant la rampe de consigne (contrainte : set_max_ramp_down).
- les **contraintes choisies par les acteurs (ActorConstraint (ACte))** qui ne sont pas liées à la technique et uniquement au choix des acteurs. Il peut s'agir d'une utilisation d'un appareil restreinte selon des plages horaires (contrainte : daily_dynamic_constraint).

Il convient de différentier les contraintes techniques des contraintes liées aux acteurs pour une plus grande pertinence et efficacité dans les discussions. Il est à noter que les contraintes techniques ou liées aux acteurs ne sont pas toujours négociables mais cela donne une première base de discussion.

**LPFICS a été complété pour permettre de séparer l'ensemble des contraintes en sous-groupes : définition, techniques, acteurs et autres** (dans le cas où certaines contraintes pourraient être ajoutées par le modélisateur sans typage spécifique). Deux autres algorithmes sont alors proposés. Le premier, intitulé *find_definition_and_actor_infeasible_constraints_set*, ne prend pas en compte les contraintes techniques et vise à identifier des contraintes incompatibles liées aux acteurs. Le second, intitulé *find_definition_and_technical_infeasible_constraints_set*, ne prend pas en compte les contraintes liées aux acteurs et vise à identifier des contraintes techniques incompatibles.

## 4. EXEMPLE D'APPLICATION

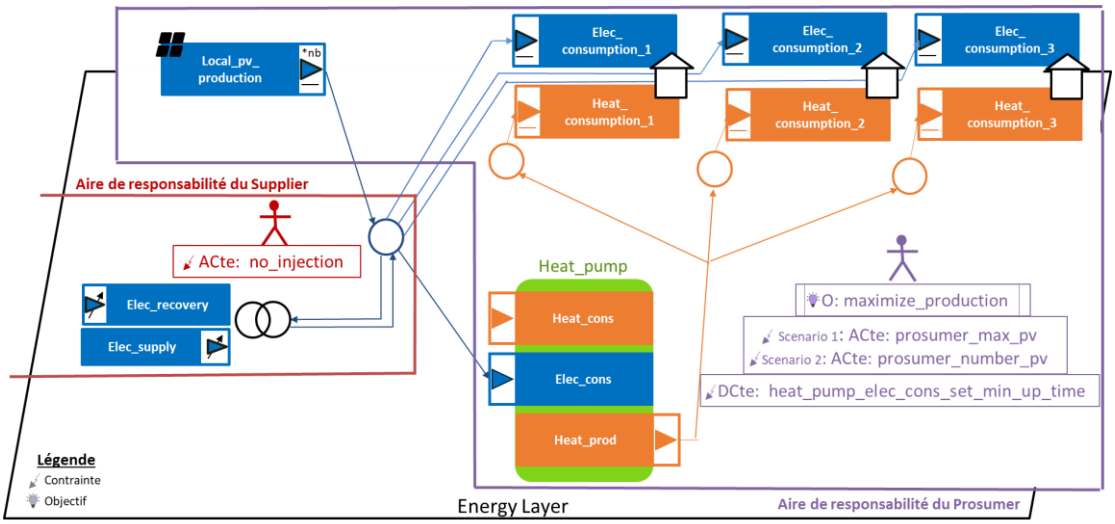


*Figure 1: Représentation graphique du modèle du projet | Source : Auteure*

Ces méthodes sont ici appliquées à un projet énergétique (cf. Figure 1, représentation graphique du modèle du projet). Le projet se porte sur trois maisons modélisées par ses consommations de chaleur (briques *Heat_consumption_1*, *Heat_consumption_2* et *Heat_consumption_3*) et électriques (briques *Elec_consumption_1*, *Elec_consumption_2* et *Elec_consumption_3*). Les maisons sont situées en bout de ligne d'un réseau électrique en zone rural ayant des difficultés d'approvisionnement. Les habitants décident de produire sur place de l'énergie à partir de panneaux photovoltaïques (brique *Local_pv_production*) installés sur les toitures des maisons et bâtiments annexes et souhaitent maximiser la production. Après une étude rapide de la place disponible, ils peuvent installer au maximum 50 kWc (scénario 1, contrainte : *prosumer_max_pv*) ou décident d'installer les 50 kWc

(scénario 2, contrainte : *prosumer_number_pv*). Le fournisseur ne souhaite pas racheter l'énergie produite (contrainte : *no_injection*). Les habitants ont également choisi d'installer une pompe à chaleur (brique *heat_pump*). Les différents systèmes sont reliés via le réseau électrique (brique *elec_node*). Concernant la chaleur, la pompe est reliée à un réseau de chauffage et au réseau d'eau chaude sanitaire des maisons. Les habitants décident de limiter l'allumage de la pompe à chaleur à une fois par heure pour allonger son cycle de vie (contrainte : *heat_pump_elec_cons_on_off_min*). L'étude se porte sur la semaine du 11 au 17 mai 2019, le chauffage est donc éteint. Le modèle est disponible en ligne sous forme de code python[3] et de notebook[4] sous le nom de *article_2020_IBPSA_constraint_identification*. Les données open-source utilisées dans le cadre de cet article y sont référencées.

### 4.1. COMPARAISON DE DEUX ALGORITHMES D'IDENTIFICATION : COMPUTEISS DE GUROBI ET LPFICS

Nous nous proposons de comparer les temps d'optimisation et d'identification des contraintes pour un même modèle infaisable selon les deux méthodologies LPFICS et ComputeISS, considérant les solveurs respectifs CBC et Gurobi. Il n'est pas possible d'utiliser Gurobi pour LPFICS car le solveur ne renvoie un statut « non résolu », qui ne correspond pas toujours au statut « infaisable ».

| Scénario | Outil utilisé | Temps de résolution | Temps d'identification des contraintes | Type de contrainte identifiée (pas de temps correspondants non indiqués) |
|---|---|---|---|---|
| **1** | **LPFICS solver CBC** | 1,48 s | 21,89 s | **DCte:** dwelling1_heat_node_power_balance<br>**DCte:** heat_pump_elec_cons_on_off_max<br>**DCte:** heat_pump_elec_cons_on_off_min<br>**TCte:** heat_pump_elec_cons_set_min_up_time<br>**DCte:** heat_pump_elec_cons_def_start_up<br>**DCte:** heat_pump_conversion |
| **1** | **ComputeISS solver Gurobi** | 0,49 s | 2,73 s | dwelling1_heat_node_power_balance<br>dwelling3_heat_node_power_balance<br>heat_pump_elec_cons_on_off_max<br>heat_pump_elec_cons_on_off_min<br>heat_pump_elec_cons_set_min_up_time<br>heat_pump_elec_cons_def_start_up<br>heat_pump_conversion<br>heat_pump_power_flow |

*Tableau 1: Comparaison des méthodologies selon les solveurs*

Dans le cadre du scenario 1, composé de 7740 contraintes et 4381 variables (2028 continues et 2353 discrètes), on observe que les deux méthodes identifient le même type de contraintes, excepté *« heat_pump_power_flow »* qui n'est identifiée que par Gurobi mais qui est liée aux autres contraintes de la pompe à chaleur identifiées dans les deux cas. Ceci peut s'expliquer par le fait que les algorithmes identifient des lots de contraintes et non pas toutes les contraintes incompatibles. On observe de moins bonnes performances pour LPFICS. Une première explication serait le temps de résolution du solveur, qui est meilleur pour Gurobi, car l'algorithme fait de nombreuses résolutions. Une deuxième explication pourrait être un prétraitement des contraintes réalisé par Gurobi mais aucune information n'a été trouvé dans la documentation de Gurobi. LPFICS permet d'identifier que les acteurs devraient discuter de la contrainte technique concernant le temps minimal de fonctionnement de la pompe à chaleur.

[3] https://gricad-gitlab.univ-grenoble-alpes.fr/omegalpes/omegalpes_examples/-/tree/master/article_case_study
[4] https://gricad-gitlab.univ-grenoble-alpes.fr/omegalpes/omegalpes_notebooks

### 4.2. IDENTIFIER LES CONTRAINTES NEGOCIABLES EN PREMIER

Comme nous l'avons dit précédemment, il nous semble nécessaire d'amener les parties prenantes à identifier les contraintes techniques et les contraintes d'acteur. Pour cela, nous nous appuierons sur la hiérarchie présentée précédemment : contraintes physiques et de définition (contraintes non négociables), contraintes techniques et contraintes liées aux acteurs (contraintes négociables).

| Contraintes souhaitées | Temps d'identification des contraintes | Nombre de résolution du modèle | Contraintes identifiées |
|---|---|---|---|
| **Algorithme basique** | 15,55 s | 66 | 1 technique et 10 de définition (cf. tableau 1 ligne 2) |
| **Définition et acteur** | 19,34 s | 62 | 3 acteurs et 4 de définition |
| **Définition et technique** | 16,29 s | 64 | 2 techniques et 10 de définition |

*Tableau 2: Validation de l'identification des contraintes selon leur type*

Les exemples réalisés sur le cas d'étude permettent de montrer que l'algorithme est cohérent dans l'identification des contraintes. Il faudrait chercher à diminuer le nombre de résolutions du modèle et pour cela, une piste serait de trier de manière pertinente les contraintes.

## 5. CONCLUSION ET PERSPECTIVES

La conception des projets énergétiques évolue vers une conception multi-acteurs. Les parties prenantes doivent alors s'accorder sur le modèle du projet constitué du modèle physique ainsi que les contraintes et les objectifs liés. Les méta-modeleurs, et notamment le modeleur OMEGAlpes, permettent une modélisation facile et compréhensible du système énergétique afin que le modèle soit support de négotiation. Cependant, les parties prenantes imposant différentes contraintes induisent souvent des problèmes sans solution. Afin de pouvoir identifier ces contraintes incompatibles dans le cadre de l'utilisation d'OMEGAlpes ; nous avons développé l'algorithme d'identification de contraintes LPFICS en Python et en open-source. LPFICS identifie un lot de contraintes incompatibles. Il peut être plus largement utilisé pour des problèmes MILP. Nous avons ensuite comparé, via l'étude d'un projet énergétique fictif, ses performances à une résolution de problème basé sur l'algorithme et le solveur propriétaire Gurobi. Un pré-classement des contraintes pourrait permettre d'améliorer les performances.

**Une fois les contraintes identifiées, il pourrait être intéressant de relaxer ces contraintes afin de montrer quelles solutions seraient envisageables.** La relaxation de contraintes pourrait se faire via un système de pénalité, déjà disponible dans PuLP à partir de « contraintes élastiques » (page « Elastic Constraints »), ou via des fonctions de désirabilité basées sur la méthode flou (Robinson et al. 2019).

## 6. REMERCIEMENTS

*Ce travail a bénéficié du soutien du CDP Eco-SESA recevant des financements de l'Agence Nationale de la Recherche, au titre du programme « Investissements d'avenir » portant la référence ANR-15-IDEX-02, du soutien de l'Agence de l'Environnement et de la Maîtrise de l'Énergie, l'ADEME, dans le cadre du projet RETHINE et de la région Auvergne Rhône Alpes dans le cadre du projet OREBE.*

## 7. BIBLIOGRAPHIE

Crozier, Michel, et Erhard Friedberg. 1981. *L'acteur et le système les contraintes de l'action collective*. Paris: Éd. du Seuil.

Gurobi Optimization, Inc. 2017. « Gurobi Optimizer reference manual, version 7.5 ». http://www.gurobi.com/documentation/7.5/refman.pdf.

Hart, William E, Jean-Paul Watson, et David L. Woodruff. 2011. « Pyomo: modeling and solving mathematical programs in Python ». Mathematical Programming Computation 3 (3): 219-60.

Junker, Ulrich. 2004. « QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems », In: *AAAI-04.* San Jose. https://www.aaai.org/Papers/AAAI/2004/AAAI04-027.pdf

Kleppe, Anneke, Jos Warmer, et Wim Bast. 2003. MDA Explained: The Model Driven Architecture: Practice and Promise.

Morriet, Lou, Gilles Debizet, et Frédéric Wurtz. 2019. « Multi-Actor Modelling for MILP Energy Systems Optimisation: Application to Collective Self-Consumption », In *Building Simulation 2019*. Rome. https://hal.archives-ouvertes.fr/hal-02285965.

Pajot, Camille. 2019. « OMEGAlpes : Outil d'aide à la décision pour une planification énergétique multi-fluides optimale à l'échelle des quartiers ». Thèse de doctorat, Université Grenoble Alpes. http://www.theses.fr/s162247.

Pajot, Camille, Lou Morriet, Sacha Hodencq, Benoît Delinchant, Yves Maréchal, Frédéric Wurtz, et Vincent Reinbold. 2019. « An Optimization Modeler as an Efficient Tool for Design and Operation for City Energy Stakeholders and Decision Makers ». In *Building Simulation 2019*. Rome. https://hal.archives-ouvertes.fr/hal-02285954.

Raad, Abbass. 2017. « Co-simulation et optimisation multi-critères en conception de bâtiment, par approche d'interopérabilité de services ». Thèse de doctorat, Université Grenoble Alpes. http://www.theses.fr/2017GREAT103.

Recht, Thomas. 2016. « Étude de l'écoconception de maisons à énergie positive ». Thèse de doctorat, Université Paris sciences et lettres. https://pastel.archives-ouvertes.fr/tel-01545437.

Robinson, E, C.J. Hopfe, M. Emmerich, L. Yevseyeva, et J.A. Wright. 2019. « Applying Desirability Functions to preference modelling in low-energy building design optimization ». In *Building Simulation 2019*. Rome.

Rodler, Patrick. 2020. « Understanding the QuickXPlain Algorithm: Simple Explanation and Formal Proof ». *ArXiv:2001.01835*, janvier. http://arxiv.org/abs/2001.01835.

Vaisman, Louise, Ghislain Bourg, Benjamin Ploux, Marcela Reinoso, Claire Huberson, et Julie Rieg. 2018. *Les facteurs sociologiques de réussite des projets de transition énergétique*. Paris: IAU Île-de-France.

## 8. WEBOGRAPHIE ET LOGICIELS – PAR ORDRE D'APPARITION DANS LE TEXTE

J. S. Mitchell et S. A. Roy, 2020. PuLP 2.2 [Logiciel]. Python. In : *Github*. 23 novembre 2019. Consulté le 6 mars 2020, https://github.com/coin-or/pulp.

COIN-OR Foundation, 2020. COIN-OR Branch-and-Cut solver 2.10.5 [Logiciel]. C++. In: *GitHub*. Consulté le 15 novembre 2020, https://github.com/coin-or/Cbc.

Free Software Foundation, 2012. GLPK [logiciel]. In : *GLPK (GNU Linear Programming Kit)*. 23 juin 2012. Consulté le 6 mars 2020, https://www.gnu.org/software/glpk/.

Notebeart Peter et Kjell Eikland, 2020. Lpsolve 5.5.2.9 [logiciel]. In : *SourceForge*. 13 mars 2020. Consulté le 15 novembre 2020, https://sourceforge.net/projects/lpsolve/.

Verger, Guillaume, 2016. Ruby-Cbc [Logiciel]. Ruby. In : *Github*. Consulté le 15 novembre 2020, https://github.com/gverger/ruby-cbc.

Page « Elastic Constraints », PuLP 2.3, 2009. Consulté le 8 novembre 2020, https://coin-or.github.io/pulp/guides/how_to_elastic_constraints.html.