UNIVERSITÉ DE LIÈGE – FACULTÉ D'ARCHITECTURE

Optimisation en conception architecturale

Les alternatives aux algorithmes génétiques

Travail de fin d'études présenté par Thomas Dissaux en vue de l'obtention du grade de Master en Architecture

Sous la direction de : Sylvie Jancart

Année académique 2017-2018

Axe(s) de recherche : Culture Numérique

Remerciements

Je tiens à remercier toutes les personnes qui m'ont aidé dans la réalisation de ce mémoire.

Je souhaite remercier David Rutten et la communauté du forum Grasshopper, toujours très positifs dans leurs réponses et enthousiastes à l'idée de voir évoluer la plateforme.

Je souhaite également remercier Thomas Wortmann et Judyta Cichocka, figures de tête dans l'optimisation en architecture, qui m'ont fortement aidé dans l'avancement de mon travail à travers leur Workshop à Hambourg.

Je souhaite remercier mes lecteurs Olivier Henz (ULiège) et John Liotta (ULB) pour l'enthousiasme exprimé tout au long de ce travail.

Un grand merci à Adeline Stals pour toutes les relectures et les conseils apportés, non seulement dans le cadre de ce mémoire, mais également pour d'autres publications.

Et enfin, un tout grand merci à Sylvie Jancart sans qui rien n'aurait été possible. En dehors des interminables relectures et corrections, de la motivation, du support moral et intellectuel, je lui suis infiniment reconnaissant de m'avoir introduit au milieu du numérique en architecture et de m'avoir poussé, à travers workshops et publications, à m'investir dans ce domaine auquel j'ai pris goût et dans lequel j'aimerais évoluer dans le futur.

Table des matières

Introduction

Introduction générale

Aujourd'hui, l'optimisation arrive en phase finale des projets de grande envergure. Les ingénieurs sont alors chargés d'optimiser certains aspects tels que la structure, l'énergie, l'acoustique, etc. La tâche d'optimisation est pour l'instant, propre au travail de l'ingénieur et n'a jamais que très peu concerné l'architecte.

Par ailleurs ce genre de pratique est souvent réservé aux grands bureaux, aux moyens financiers importants, et pouvant se permettre de développer, en interne, des solutions appropriées à chaque projet. La réalité en Europe et en particulier en Wallonie est que plus de la moitié des bureaux est constituée de moins de cinq architectes intervenant la plupart du temps sur des projets à petit budget [Stals, 2016]. Les ressources économiques ainsi que le temps consacré à la formation continue sont fort limités et ne permettent pas un apprentissage approfondi des nouvelles approches numériques.

Ceci étant, on constate un certain engouement à la transposition de ces techniques d'optimisation et il apparaît petit à petit des solutions plus abordables dans le milieu de l'architecture. Jusqu'ici présentées de manière théorique, elles émergent dans la recherche de solutions pratique et plus abordables. Des architectes, tels que David Rutten [2011], Thomas Wortmann [2014] et Judyta Cichocka [2015], collaborent avec ingénieurs et mathématiciens afin de développer des pratiques numériques adaptées à leurs nécessités.

Une étude des différentes méthodes d'optimisation développées et proposées à ce jour et leurs comparaisons à travers des cas concrets est donc nécessaire. Cela permet, d'une part, de se faire une idée de la difficulté d'exécution de ces algorithmes et d'autre part, d'identifier les bénéfices qu'un architecte peut en tirer, y compris pour des travaux de petite échelle.

Définition du problème

Traditionnellement, le projet démarre par une phase de recherche. Celle-ci consiste en la réunion d'informations relatives au projet ainsi qu'à la réalisation des premières esquisses sous forme de dessins et maquettes. Elle a pour but de constituer une base de discussion permettant l'échange entre les différents acteurs. La contrainte principale à ce stade, est le temps. Une fois celui-ci écoulé, l'architecte se hâtera de retranscrire numériquement son travail sous les différents formats inhérents à chaque profession satellite. L'informatique ne participe alors en rien au développement du projet, elle devient même une contrainte importante.

Cela est dû, en partie, à l'avènement du CAD (Computer Assisted Design) en architecture ces dernières années. Le CAD n'a jamais été développé que comme outil de retranscription du dessin et non comme outil de recherche ou de développement [Terzidis, 2006]. L'architecte est alors soumis aux possibilités du logiciel ou à ce qu'il est lui-même capable de reproduire sur celui-ci.

> *"As most of the researchers in CAD were primarly concerned with the technicalities of converting design ideas into digital tools, none, if any, was also concerned with using those tools to actually design".*
>
> *Terzidis [2006]*

Pour la majorité des architectes, le travail est directement influencé par le logiciel qu'ils utilisent et pour lesquels se développe une forme d'addiction qui induit un certain workflow et une certaine logique [Rocker, M, 2006]. Ces architectes ont peu, voire aucune connaissance des algorithmes sous-jacents aux outils qu'ils emploient réduisant l'interaction à une manipulation de formes préprogrammées à travers une interface masquant toute complexité mathématique [Rocker, M, 2006]. Il est intéressant de concevoir que, de manière perverse, le simple fait de savoir qu'il faudra tout retranscrire numériquement pousse l'architecte à même se limiter dans son esquisse aux outils informatiques que propose la plateforme sur laquelle il travaille.

Il en découle une perte importante de finesse dans la définition de l'architecture par réduction souvent simpliste des logiciels de dessin. En plus de cette perte de résolution vient s'ajouter la mauvaise gestion de l'information, en effet, une source de perte importante est la transformation des données vers les différents formats propres aux nombreuses professions tels que l'ingénieur

AEC, l'ingénieur structure ou l'entrepreneur. Chaque poste nécessite en effet un format spécifique de document, ce qui impose à l'architecte de souvent devoir retranscrire plusieurs fois une même idée sous divers environnements exigeant chacun une certaine expertise.

But du travail

A l'heure du monde de l'information, l'informatique est devenue une nécessité omniprésente à laquelle se confronte constamment l'architecte. La quantité d'informations à traiter est souvent telle qu'il est nécessaire d'amorcer directement le projet de manière informatique [Grady, 2011]. Il est donc important de comprendre cet outil afin d'en faire un élément générateur.

Se pose alors la question d'émancipation. Etant donné sa présence inhérente aujourd'hui en architecture, comment intégrer l'informatique au développement du projet de sorte à ce qu'elle fonctionne pour et non contre l'architecte ?

Afin d'inverser ce rapport de servitude, il faudrait donc apprendre à communiquer avec la machine, lui parler dans un langage qu'elle comprend, c'est-à-dire, coder.

> *"By using scripting languages, designers can go beyond the mouse, transcending the factory-set imitations of current 3D software".*
>
> *Rocker [2006]*

M. Rocker fait le lien entre l'architecture et l'approche computationnelle en revenant aux origines du mot code. Il est dérivé du mot « codex » qui désignait ces tablettes de bois réalisées par les Romains ayant pour objectif de distribuer les règles et principes à respecter à travers leur empire. L'architecture est de la même manière codée, c'est-à-dire dirigée par un ensemble de règles, conventions et contraintes. Whitehead [2003] appuie indirectement ces propos et affirme que l'architecte pense en fait déjà de manière programmatique, il n'a néanmoins ni le temps ni l'envie d'apprendre à programmer. Il existe cependant des alternatives qui permettent aux architectes d'aller plus loin dans cette communication et cela à moindre effort.

La programmation visuelle permet de passer outre la complexité liée à la programmation textuelle. Elle est par conséquent particulièrement adaptée aux architectes et se développe d'ailleurs en périphérie d'un certain nombre de logiciels liés au design.

De par la gestion d'information quasi-totale induite par cette approche, il est possible de communiquer directement vers une série d'applications liées à l'ingénierie. Ces applications concernent notamment la simulation et l'optimisation [Turrin, 2011]. Celles-ci prennent petit à petit une place plus importante dans le développement du projet et apportent un nouveau degré de cohérence dans son évolution.

Cette approche se traduit aujourd'hui par une série de travaux d'envergure, souvent monumentale. Elle fût développée au sein de grands bureaux où architectes, ingénieurs et informaticiens travaillent côte à côte. Ayant appris à travailler ensemble dès les prémisses du projet, de nouveaux modes de travail se sont installés afin de répondre aux complexités actuelles du bâtiment. Beaucoup de termes sont liés à ces méthodes ; architecture paramétrique, architecture computationnelle, méthode générative, information design etc… La suite de ce travail permet entre autre de bien distinguer les nuances entre plusieurs de ces termes.

Quoiqu'il en soit, cette nouvelle démarche est responsable d'un réel engouement au sein de l'architecture et a alimenté le développement d'un certain nombre de logiciels liés à son évolution. Ces projets, comme par exemple Grasshopper3D, ou Dynamo, sont essentiellement open source, ce qui représente un aspect important à leur épanouissement. En effet, une partie de la communauté d'utilisateurs participe à son développement, l'ouverture vers d'autres logiciels est facilitée, et l'accès en est ouvert.

Ces nouveaux outils représentent une opportunité pour les plus petites firmes qui doivent faire face aux exigences et contraintes actuelles. Bien maitrisés, ils permettraient une pertinence accrue dans le développement de projet avec comme seul investissement leur apprentissage.

Ce travail a pour but de présenter cette approche qui représente, pour beaucoup, le futur de l'architecture, certains parlant même de passage vers un nouveau paradigme [Liddament, 1998, Terzidis, 2004, Barbisan 2013, Couwenberg, 2014, …] où l'architecte, sans devenir informaticien, prend le contrôle de cet outil qu'est l'ordinateur.

Ceci étant, il est important de garder à l'esprit l'idée de départ. Certes la programmation visuelle nous offre une maîtrise plus profonde de la géométrie, mais celle-ci reste elle-même dépendante à un système sous-jacent. On décrit en partie le langage utilisé en informatique par son niveau de profondeur : si l'interface du logiciel est de profondeur 0 il est dicté par un système de profondeur 1 lui-même réglé par un système de profondeur 2 et ainsi de suite jusqu'à atteindre le niveau le plus profond c'est-à-dire lorsque le software communique directement avec le

hardware. Le but n'est pas de convaincre l'architecte de maitriser toutes ces couches mais bien d'en être conscient et de s'arrêter là où c'est pertinent.

Le cas d'étude sera ici l'optimisation et en particulier le choix de l'algorithme. La gestion accrue de l'information relative à un projet permet l'optimisation de ses paramètres à travers la simulation. Hors à nouveau, peu de considération est mise sur le choix de l'algorithme responsable de cette optimisation. C'est donc de manière aveugle, au même titre que l'interface d'un logiciel de CAD, que l'architecte tend à l'utiliser.

Etat de l'art

« Aujourd'hui, néanmoins nous sommes passés outre le stade de la belle image qui se vend bien et « les démarches s'inscrivent dans une culture constructive plus rigoureuse ».

Picon A. & Razavi A., 2011

L'architecture paramétrique

Le groupe SmartGeometry définit le paramétrique comme l'outil permettant le passage entre le dessin de certaines formes par CAD à la création de relations géométriques [Aish, R. et al, 2006]. Le but du groupe SmartGeometry est de créer les fondations intellectuelles pour une manière de concevoir plus profonde. Le but n'est pas de changer l'approche intuitive de l'architecte mais de permettre la création et le contrôle de géométries plus complexes. La raison de cette recherche est que, d'après les auteurs, l'architecture de par son obsession du minimalisme par usage de forme orthogonales et plates aurait tendance à oublier sa maîtrise de la géométrie descriptive ou inversement à faire preuve d'impressionnisme par usage de formes trop complexes sans considération du programme. En dehors de leur complexité, ces nouvelles formes pourraient se retrouver être des solutions formelles efficaces dans le projet lorsqu'il s'agit de concilier de nombreuses variables. Hors, dans une méthode de travail linéaire traditionnelle, l'architecte dessine, retranscrit et se soucie ensuite des performances. La conséquence d'une telle habitude en architecture est qu'on a préféré faire en sorte de rendre ces éléments simplistes plus efficaces au lieu de s'étendre sur leurs bases géométriques.

C'est ici qu'intervient l'architecture dite algorithmique. Bien que ce terme soit réducteur, cette approche est souvent qualifiée d'architecture paramétrique. En effet un logiciel de BIM tel que Revit est paramétrique dans le sens où il est aisé de modifier un paramètre tel que l'épaisseur des murs ou ces composants sans interférer sur les relations géométriques prédéfinies avec le reste du bâtiment. L'approche étudiée ici est plutôt algorithmique. Le concepteur définit directement l'ensemble des relations géométriques du projet et ses différents paramètres. L'architecte n'est dès lors plus dépendant des principes de constructions encodés par le logiciel, ce qui engage à plus d'innovation.

Terzidis [2006] donne une définition du mot algorithme :

> *"Theoretically, an algorithm is the abstraction of a process and serves as a sequential pattern that leads towards the accomplishment of a desired task."*
>
> *Terzidis K.*

Sur base de cette définition, il offre 2 interprétations : la rationalisation de la pensée humaine, hors l'information qui compose l'algorithme ne reflète pas nécessairement la nature du résultat, et une expression interprétée de manière à ce que l'ordinateur la comprenne.

Dans ces 2 cas il en découle plusieurs avantages considérables. Le premier atout est celui d'une meilleure compréhension du projet. En effet, « encoder » le projet pousse la réflexion plus loin en terme de faisabilité promettant un degré supérieur de compréhension [Aish et al, 2006].

Un autre avantage est l'approche non linéaire du projet. Il n'est plus nécessaire de fixer les décisions prématurément au risque de voir son projet évoluer vers quelque chose qui irait à l'encontre de l'idée de départ.

Et enfin, dans une équipe multidisciplinaire, chaque acteur a besoin d'une représentation différente du projet, hors le paramétrique permet cela très facilement. Ayant accès à l'information, le portage vers d'autres formats est très efficace [Leitao, 2017].

La perte de donnée est une problématique récurrente en architecture et fait le sujet de plusieurs recherches. La programmation visuelle ne représente d'ailleurs pas encore la solution efficace pour certains qui prônent une approche programmatique scriptée plus traditionnelle. Mr Leitao par exemple, consacre une partie de son travail au développement d'un programme appelé Rosetta, un environnement de programmation écrite, qui tente de résoudre le problème de perte de données lors du portage d'information en limitant le nombre d'intermédiaires (voir fig. 1).

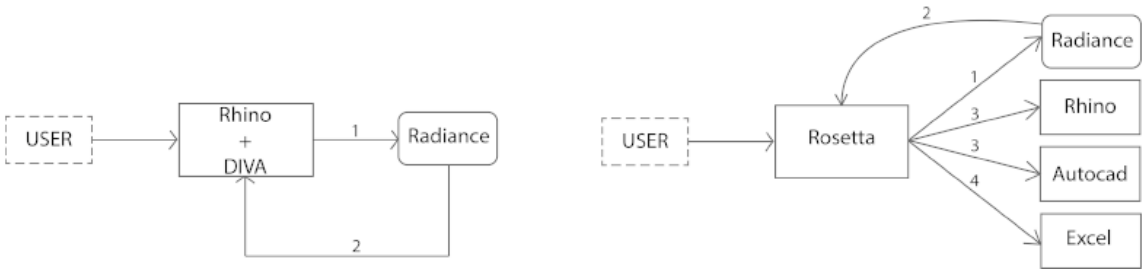


Fig.1 : Comparaison entre un workflow type et un workflow utilisant Rosetta

[Leitao, 2017]

Le BIM aujourd'hui possède une approche qui rejoint cette idée à travers le A-BIM (Algorithmic Based Building Information Modelling). Mais les informations sont en général emprisonnées dans le logiciel qui les sert, ce qui induit une perte d'information lors d'export et import de fichier mais aussi l'interaction entre acteurs se fait à travers le modèle généré et non le code qui le génère. Dans le même esprit que Leitao [2017] sur la portage d'information, Janssen *et al* [2017] promeuvent l'usage du VC (Visual Control). L'approche est semblable à celle de la plateforme GitHub, une plateforme de développement software partagé, mais dans le cadre du développement d'un projet d'architecture. Elle nécessite donc malheureusement une certaine expertise en informatique.

L'architecture paramétrique permet de lier facilement forme simulation et optimisation à travers une série de logiciels indépendants l'un de l'autre permettant plus de liberté.

Cette interface de programmation visuelle permet donc, en comparaison avec le logiciel de CAD, de débrider la recherche dès le départ. Elle permet, entre autre, de prendre parti de la puissance computationnelle de l'ordinateur dans un but d'alimenter cette recherche notamment par la simulation et l'optimisation.

Logiciels de développement d'algorithmes

La programmation visuelle se distingue de la programmation scriptée traditionnelle par l'usage de fonctions, préprogrammées ou non, apparaissant sous forme de nœuds qui acceptent des données en entrée et dont en ressortent un résultat défini par la fonction (voir fig. 2).

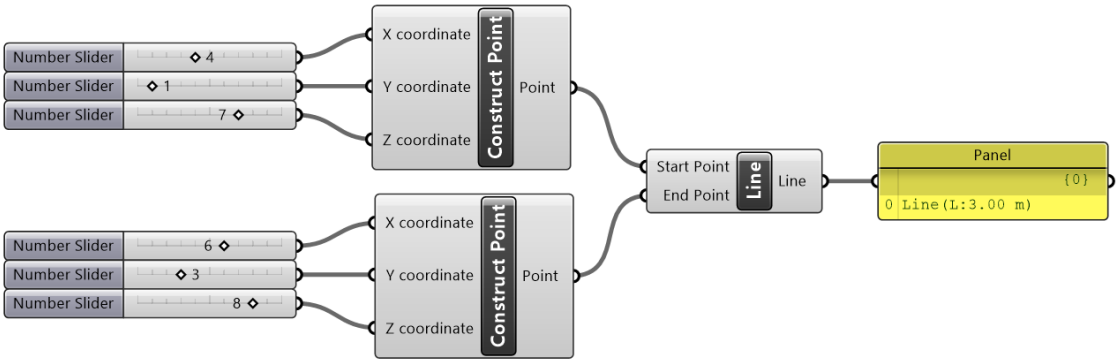


Fig. 2 : Exemple de programmation visuelle. 2 points sont définis par 3 paramètres que sont les valeurs de coordonnées x, y, z. Ces points sont utilisés comme paramètres d'entrée de la fonction « Line ». Il en ressort une ligne (avec indication sur sa longueur)

> *"A module in a dataflow programming language (the programming paradigm of graph based parametric schemata), defined by Wong and Sharp, is "a sequence of program instructions bounded by an entry and exit point".*
>
> *Burry & Burry [2011]*

Un ensemble de nœuds successifs forme une chaîne qui sera l'algorithme définissant le design à travers une série de paramètres déterminés. Les contraintes imposées à ces paramètres définissent le degré d'exploration de la forme. On parle alors de design paramétrique.

Cet algorithme définit en fait le concept de base pensé et réinterprété. Cette réinterprétation est limitée dans un logiciel de CAD mais pas, ou moins, dans le cas de la programmation visuelle.

C'est à travers cet environnement de programmation visuelle que l'utilisateur aura accès à une multitude de fonctions aidant à la simulation et à l'optimisation.

Alfaiate [2017] soulève néanmoins un désavantage pour la programmation visuelle; elle ne serait pas appropriée pour les projets plus complexes. Néanmoins, d'après nos sources, ce problème n'est cité que par cette auteure qui promeut une approche plus profonde de l'informatique par la programmation scriptée du projet d'architecture. L'intérêt de ce travail porte ici sur l'architecture à l'échelle de l'architecte plus commun qui s'occupe en général de projets d'envergure plus réduite telle que la maison unifamiliale.

Il faut également noter une chose, que ce soit pour les logiciels de CAD ou pour les environnements de programmation visuelles, ceux-ci influeront toujours à un certain degré le projet [Janssens, 2011 ; Leitao, 2012 ; Celani, 2012 ; Davis, 2012 ; Stals, 2016]

La suite du travail s'appuiera sur le Logiciel Grasshopper en raison de sa popularité en architecture [Cichocka et al, 2017]. Celui-ci fonctionne à travers le logiciel de modélisation Rhinoceros3d qui permet un feedback visuel (voir fig. 3).

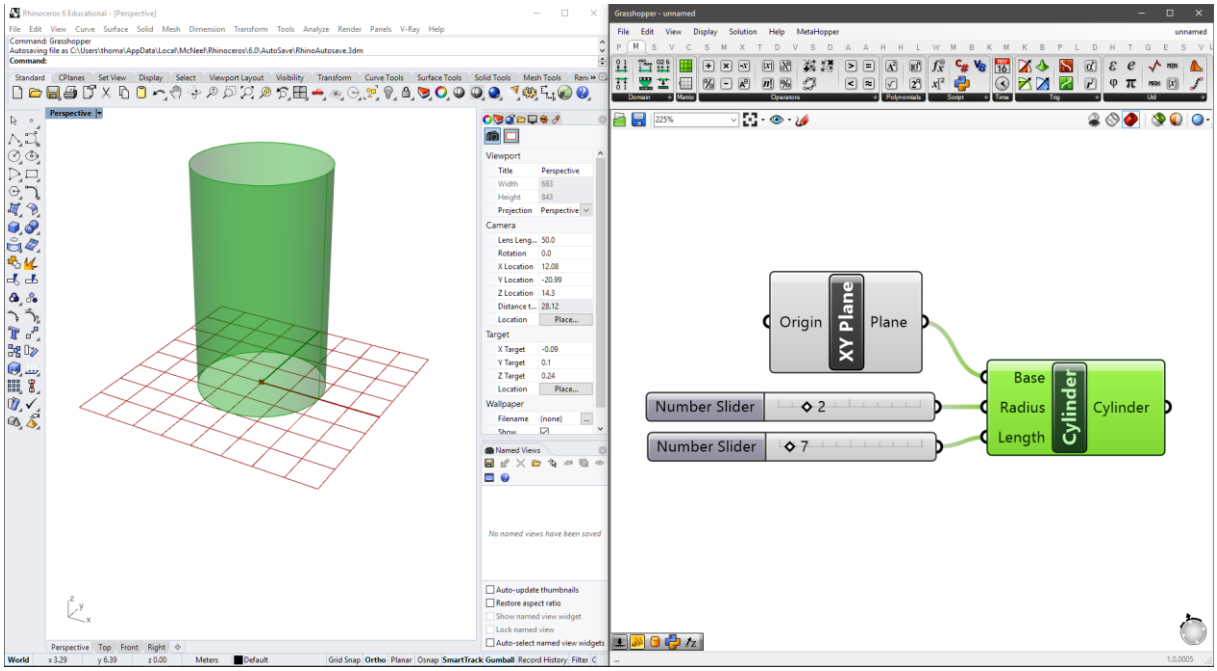


Fig. 3 : Retour visuel en direct de la définition paramétrique dans Rhinocéros.

Bien qu'il soit rattaché à la notion de paramétrique, David Rutten, son développeur, le définit comme un éditeur graphique d'algorithmes qui ne requière aucune connaissance en programmation mais une logique de pensée.

David Rutten, Architecte diplômé de TU-Delft, était frustré de ne pas pouvoir remonter dans son historique afin de modifier les données rentrées dans le logiciel Rhinoceros. Grasshopper, à l'origine appelé « Explicit History », avait pour but de donner l'opportunité à son utilisateur d'étaler le processus linéaire de modélisation de sorte à pouvoir revenir sur chaque étape et en modifier les paramètres tels que la valeur d'une courbure ou une hauteur d'extrusion. Cette approche ne laissa plus qu'au designer à définir un ensemble de relations géométriques aux paramètres variables, d'où le terme de paramétrique. Hors ce procédé de modélisation était déjà présent sur d'autres logiciels tels que 3DSMax, Revit ou Maya. L'avantage de Grasshopper a été d'évoluer au fil du temps vers un environnement de programmation visuelle complet et indépendant sans plus se limiter uniquement aux fonctions offertes par Rhinocéros.

Grasshopper se présente comme un canevas sur lequel on étale une série de fonctions ou « nœuds » que l'on connecte de sorte à les mettre en relations. Le désavantage majeur de l'approche séquentielle est l'absence de « loop » mais ce défaut peut être contourné de différentes manières, soit par des plug-ins, soit en passant par une programmation scriptée, ce que Grasshopper permet très facilement.

L'énorme avantage de Grasshopper comparé aux autres environnements de programmation visuelle est le fait qu'il soit gratuit et ouvert. Il est constamment nourri par des plug-ins de la communauté et sert de tremplin éducatif vers la programmation scriptée traditionnelle. C'est notamment sur base d'une série de plug-in, notamment de simulation et d'optimisation que se focalisera la suite de cette étude. Cela n'aurait pas été possible avec d'autres logiciels, de par leurs prix, leurs complexités parfois, le manque de communauté et le manque d'outils directement implémentés, ce que Cichocka *et al.* [2017] mettent en avant (voir fig. 4).

PRODUCER - SOFTWARE	VISUAL PROGRAM-MING/ PARAMET-RIC PLUG-IN	% USERS [12]	OPTIMIZATION PLUG -IN /OPTIMIZER	OPTIMIZATION ALGORITHMS	CATEGORY	AUTHOR AND DATE OF IMPLEMENTATION
Gehry Tech-nologies: Digital Project	n/a	3%	n/a	n/a	n/a	n/a
Bentley Sys-tems: Mictostation	Generative Components	0%	Prototype Design Evolution	GA	EC	Bentley Applied Rese-arch, 2011
Autodesk: Revit Archi-tecture	Dynamo (2011)	7%	Optimo (2014)	MOEAD	EC	Mohammad Rahmani Asl and Dr. Wei Yan, 2014
McNeel & As-sociates : Rhi-noceros3D	Grasshopper (2008)	90%	Galapagos	GA	EC	David Rutten, 2008
				SA	other[1]	David Rutten, 2011
			Goat	COBYLA,BOB YQA,Sbplx, DIRECT,CRS2	other[2]	Simon Flory, 2010-2015
			Opossum	Surrogate mod-els	Modelbased	Thomas Wortmann, 2016
			Octopus	SPEA- 2 and HypE	EC	Robert Vierlinger, 2013
Nemetschek North Ame-rica: Vectorworks	Marionette (2015)	<1%	n/a	n/a	n/a	n/a

Fig. 4 : Liste non exhaustive des logiciels de dessin proposant un plug-in de développement paramétrique [Cichocka, 2017]

La programmation scriptée traditionnelle dans Grasshopper est également possible, ce qui est un autre gros avantage. La plupart des API (Application Programming Interface) font appel à des langages propriétaires propres à chaque programme comme le Rhinoscript pour Rhinoceros MEL pour Maya MAXscript pour 3DSMAX ce qui représente un certain investissement en temps et pousse à fidéliser de manière perverse les utilisateurs investis. A coups de complaintes des utilisateurs, les géants de l'industrie comme Autodesk ont tout de même fini par implanter Python dans la plupart de leurs logiciels. Mais aucun n'offre autant de possibilités que Grasshopper qui permet d'emblée de coder en Python, C# ou Visual Basic (voir fig. 5).

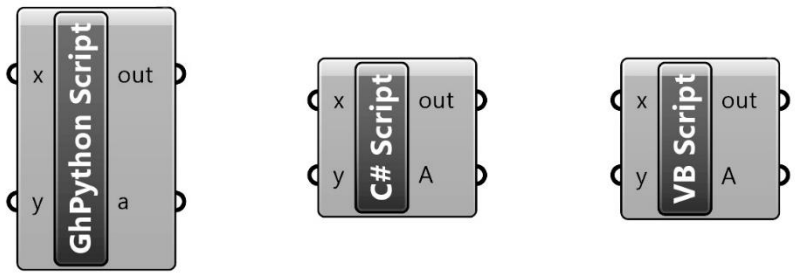


Fig. 5 : Nœud de développement de fonctions par script. Grasshopper propose l'usage de Python, de C# ou VisualBasic, 3 languages informatiques très populaires selon l'index TIOBE [https://www.tiobe.com/tiobe-index/, 2018].

Outils de support à la simulation

La simulation a pris une part essentielle dans le développement du projet d'architecture, elle est même devenue indispensable. D'après les directives européennes sur les performances énergétiques du bâtiment du 19 mai 2010 les objectifs seraient que d'ici le 31 décembre 2020, tous les nouveaux bâtiments soient à consommation d'énergie quasi nulle (31 décembre 2018 pour les bâtiments publics), cela nécessite une évaluation de performance du bâtiment avant construction par simulation. Dans ce contexte Attia *et al* [2013] argumentent le fait de repenser le design dès ses prémisses au lieu de régler ponctuellement chaque détail à posteriori. Le raisonnement se prête entièrement au workflow présenté ici. Le développement du projet à travers une plateforme telle que Grasshopper permet un retour direct par simulation sur les performances du bâtiment. Le retour en temps réel, aux prémices du projet, de valeurs indicatives telles que la performance énergétique, promet un développement cohérent du projet dès le départ. On évite ainsi l'évaluation à postériori du groupe d'ingénieurs qui dans une logique linéaire pourrait aller à l'encontre du projet et imposer des concessions qui seraient en opposition au concept de départ.

"En nous appuyant dès le départ sur l'outil numérique, nous voulions penser ce projet dans une totalité géométrique ; être toujours capable de le considérer dans l'ensemble de ses aspects plutôt que de se référer alternativement à des parties distinctes en plans, coupes et élévations".

Matt Grady [2011]

Cette approche ne promeut en rien l'obsolescence de l'ingénieur ou de l'architecte mais bien une collaboration plus étroite entre les différents acteurs dès la naissance du projet [Turrin M., 2011]. La plupart des logiciels de simulation utilisés aujourd'hui et repris par SouthZEB, une plateforme mise en place par l'Europe dans le cadre de ses objectifs, est d'ailleurs peu connue des architectes et nécessite une certaine expertise. Il existe néanmoins certains outils plus accessibles qui permettent aux architectes travaillant en petit comité de diriger la conception, certes de manière moins précise mais toujours dans cet esprit de cohérence entre le projet et ses objectifs.

Les outils de simulations disponibles touchent à énormément de domaines, que ce soit la structure, les performances énergétiques, l'acoustique mais également la production. Dans un contexte où l'architecte s'écarte des formes standard il est en effet indispensable de considérer le challenge constructif que cela implique.

Fig. 6 : Exemples de plug-ins populaires de simulation, en partant du coin supérieur gauche, Ladybug, Honeybee, Butterfly, Dragonfly, Kangourou, Karamba, Pigeon, Lark, Goldfish, Pachyderm, RhinoCAM, KUKA prc [www.food4rhino.com].

L'architecture traditionnelle ne tend à intégrer les critères évaluables que dans les phases avancées du projet. La plus large part des performances du bâtiment tient alors de l'intuition de l'architecte et contraint considérablement l'exploration sur base des résultats de simulation par après. Une telle approche tend donc à limiter le processus de conception. Une relation interdisciplinaire à la genèse du projet permet l'exploration de davantages de solutions et crée

un lien entre la forme et l'évaluation numérique des performances, réduisant ainsi l'investissement dans des projets peu performants [Turrin, 2011].

La simulation permet ce passage du design paramétrique à ce qui s'appelle l'architecture paramétrique par la prise en compte des contraintes propres aux bâtiment, de la même manière qu'un architecte qui passe de l'esquisse au projet à proprement dit.

De façon pratique, Grasshopper offre un panel d'outils de simulations qui touchent à beaucoup de secteurs mais comme il a été fait mention plus tôt, il reste un environnement de programmation ouvert et est capable, sous réserve d'une relative connaissance informatique, de faire appel aux données calculées sur d'autres logiciels.

La simulation permet donc un feedback dynamique lors du développement du projet et aide à le diriger de manière pertinente mais l'architecte doit tout de même faire preuve d'une certaine humilité face à la quantité d'informations et aux complexités physiques relatives au bâtiment. L'interprétation des données et l'avancée du projet reste à tout moment sous contrôle de l'architecte, hors, celui-ci pourrait extrapoler certaines données sur bases de quelques échantillons programmés manuellement. Ceci n'est pas efficace. Les relations entre les différents paramètres sont souvent complexes et ne permettent pas de simplification mentale. Lorsque le domaine contraint des paramètres s'étend il devient donc nécessaire de faire appel aux outils d'optimisation.

Outils de support à l'optimisation

Il existe en optimisation deux approches différentes : l'exploitation et l'exploration.

De manière générale l'approche par l'exploitation se focalise sur une solution et tente de la rendre la meilleure possible. Il est alors question d'une recherche locale poussée basée sur un candidat de solution choisi de manière arbitraire. Cette approche est extrêmement scientifique et propre au domaine de l'ingénierie. C'est elle qui est privilégiée aujourd'hui dans le workflow linéaire typique des bureaux d'architecture. Le projet est développé et chaque aspect du projet est envoyé vers un spécialiste qui rendra son verdict sur un type de châssis ou une section de poutre sans regard sur le projet en lui-même. Ce travail à posteriori est typique de l'approche linéaire des bureaux d'architecture.

En revanche, l'approche par exploration envisage une large gamme de candidats afin de déterminer une ou plusieurs solutions, la recherche est donc fondamentalement globale. Le terme d'exploration est d'autant mieux choisi qu'elle permet « l'exploration » d'un éventail de solutions, formelles, au projet. Cette exploration est essentielle en architecture car il existe énormément de paramètres non interprétables numériquement, tels que la culture, la religion, l'histoire, la qualité des espaces etc… Il est donc important de prendre du recul et de faire preuve de sensibilité face à la réponse mathématique que procure l'outil informatique. De plus l'objet de la simulation reste virtuel jusqu'à sa construction et ne sera donc jamais totalement représentatif de la construction finale dans son contexte réel.

L'optimisation va servir à orienter l'exploration des paramètres de notre modèle de sorte à se diriger vers une série de solutions pertinentes par rapport au problème posé. L'émergence de la forme peut être reçue de différentes manières : soit comme inspiration à un développement parallèle du projet et dans ce cas-là, le modèle paramétrique, sa simulation et son optimisation ne servent que de cas d'étude, soit l'architecte fait preuve d'un « laisser aller » dans sa conception et décide de s'appuyer directement sur les résultats reçus jugés favorables.

Une bonne analogie serait alors de voir l'ordinateur non plus comme un simple outil mais comme un collaborateur qui excelle en rapidité lorsqu'il s'agit de présenter des variantes, efficaces et pertinentes, du concept de départ [Cohen, 2011].

Ces principes de laisser aller et d'émergence de la forme sont sujets à controverse de par le manque de contrôle que cela implique. Picon [Terzidis, 2006] présente deux points de vue en terme d'exploration. Un contrôle total menant à une forme, certes recherchée mais paradoxalement aléatoire de par son incompréhension des processus sous-jacent, à l'image des Blobs. Et l'architecture comme résultat d'un processus établi mais possible qu'à travers le travail de l'outil informatique. C'est bien ce deuxième point auquel s'intéresse l'optimisation.

Lors d'une interview entre A. Picon et l'architecte américain P. S. Cohen [2011], ce dernier insiste sur l'importance de la détermination géométrique dans l'usage des techniques computationnelles. Il qualifie cette approche numérique d'alliance du contrôle et de l'imprévisibilité. C'est également pour cela que Peter Eisenman est définit par Picon comme un précurseur de l'architecture numérique, car après avoir programmé une série d'automatismes, il acceptait de perdre le contrôle un instant afin de voir naître la forme architecturale [Razavi, Picon, 2011]. Il est alors question d'architecture générative.

De cette émergence peut naître une certaine complexité formelle que l'on retrouve notamment dans certains projets de grande envergure. Cette complexité au premiers abords peut sembler aléatoire et inutilement complexe mais est le fruit d'un ensemble de règles très simples qui ne sont appréhendables qu'à travers l'outil informatique et ne peuvent être anticipées mentalement. Le mathématicien Wolfram [2002] désigne ce phénomène d'émergence géométrique comme étant une nouvelle science.

"[…] Humans today have become capable of exceeding their own intellect. Through the use of algorithms, computation, and advanced computer systems designers are able to extend their thoughts into a once unknown and unimaginable world of complexity"

Terzidis K. [2006]

Une étude sur l'usage de l'optimisation en architecture menée par Bradner et al. [2014] montre néanmoins que les architectes qui font usage de ces méthodes ont tendance à considérer le résultat de l'optimisation comme l'équivalent d'une esquisse de départ, pertinente, sur laquelle démarrer le projet. Ce principe de laisser-aller n'est donc pas partagé dans la pratique professionnelle courante.

L'optimisation est donc l'élément essentiel à la naissance de ce nouveau paradigme numérique en architecture car il va permettre la croissance d'une nouvelle collection formelle en architecture. Des formes non-appréhendables par le crayon et qui à l'aide de l'informatique prennent tout leur sens. Le piège bien sûr est de ne pas tomber dans l'excès, l'exubérance, c'est à dire l'acte purement artistique ou publicitaire sans regard sur la fonction [Alonso, H.D., 2010].

Accès aux outils de paramétrique

"The transformation towards computer-driven form based design in global practice has not reach its full potential yet. Partially because of the lack of computational education of architects, and confusing literature […]. Corporate architectural practices such as,[…],use the computer simply as an efficiency tool while continuing to develop design through traditional manual means, and prominent, avant-garde practices, such as Gehry, Morphosis, or Zaha Hadid, use the computer as a means of marketing and presentation, despite their unsubstantiated claims to the opposite.[…]"

Terzidis K. [2006]

Lors d'une interview de D. Davis par le groupe Designalyze [2015], ce dernier l'interroge sur la nécessité d'enseigner l'informatique de manière plus poussée, c'est-à-dire la programmation. Bien que Davis D. soit connu pour son implication dans l'architecture numérique, celui-ci répond par la négative estimant qu'il existe déjà énormément de chose à assimiler en architecture et qu'il ne faudrait pas forcer certains aspects spécifiques de la profession sur les étudiants, rester le plus général possible. A l'époque de l'internet, les étudiants intéressés se procurent de toute façon l'information par eux-mêmes. Un point important est néanmoins soulevé lors de la discussion ; l'influence que possèdent les industries de développement de software sur l'architecture est immense. Ce constat est d'ailleurs mis en avant lors de l'étude menée par A. Stals qui expose en Belgique les facteurs influents sur le projet en majorité dans les petites et moyennes entreprises (voir fig. 7). Il y a dès lors un devoir de prise de conscience sur l'utilisation de ces outils.

"It is the programmer that invented the tool and set out the workspace, capabilities, and limitations for the designer to work within."

Terzidis K.

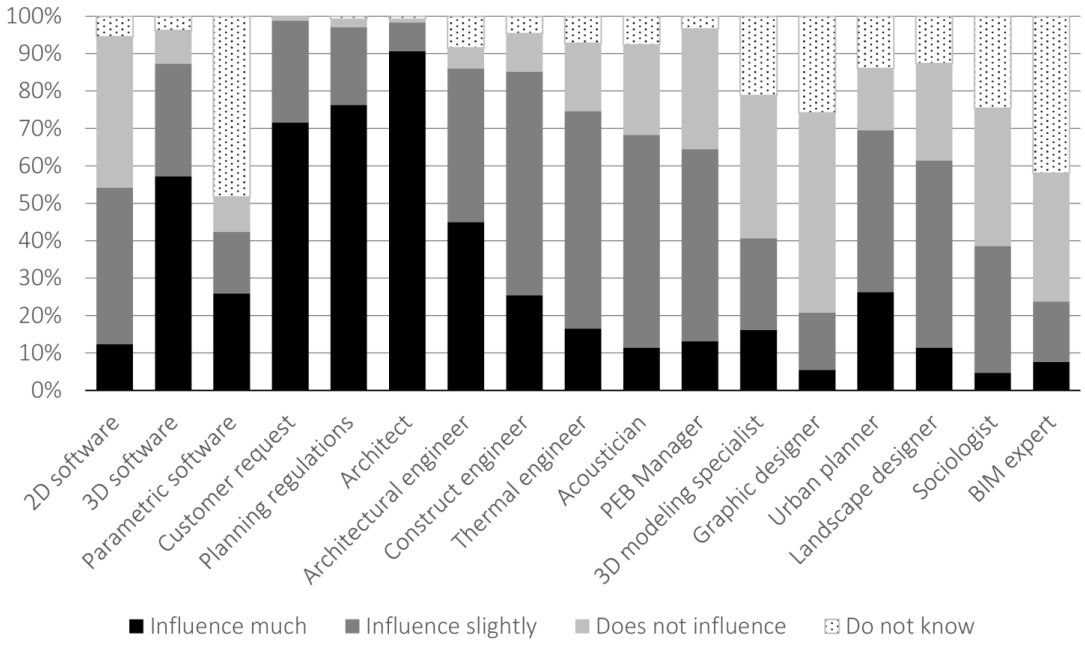


Fig. 7 : Facteurs influençant la production formelle en architecture en Belgique [Stals A., 2016]

Ce point de vue est repris par Girard [2017] plus catégorique : *« Nous pensons qu'il faut dynamiter la façon d'enseigner l'architecture en France »*. Ce propos est néanmoins tempéré en expliquant qu'il serait au moins indispensable pour un étudiant sortant d'avoir acquis les principes fondamentaux de la computation.

Gregg Lynn et Mark Foster Gage [2011] modèrent également les plus passionnés en expliquant que bien que coder permette d'aller au-delà des limites imposées par nos logiciels, il y a une limite où l'architecte doit s'arrêter d'être programmeur pour rester architecte.

Ces propos peuvent être mis en relation avec les résultats obtenus par J. Cichocka [2017] lors d'une enquête sur l'usage du paramétrique qui montre à travers un échantillon d'architectes, enthousiastes vis-à-vis de la paramétrique, un certain intérêt (87%), à ne pas devoir passer par la programmation traditionnelle scriptée dans leur travail.

La programmation visuelle semble donc être un bon compromis pour les architectes et une ligne de conduite qui coïncide avec les propos précédents.

Peu d'information concernant l'optimisation chez les architectes est actuellement disponible. Cela s'explique par le faible usage des outils paramétriques en général. En Belgique par exemple, la paramétrique, nous rapporte A. Stals, est très peu utilisé (voir fig. 8). On remarque néanmoins sur la figure x qu'environ la moitié de l'échantillon des architectes interrogés ne

sont pas au courant de l'existence des outils paramétriques et que seul 10% de ceux qui connaissent ces outils estiment qu'ils n'auraient aucune influence sur le projet (voir fig. 7).

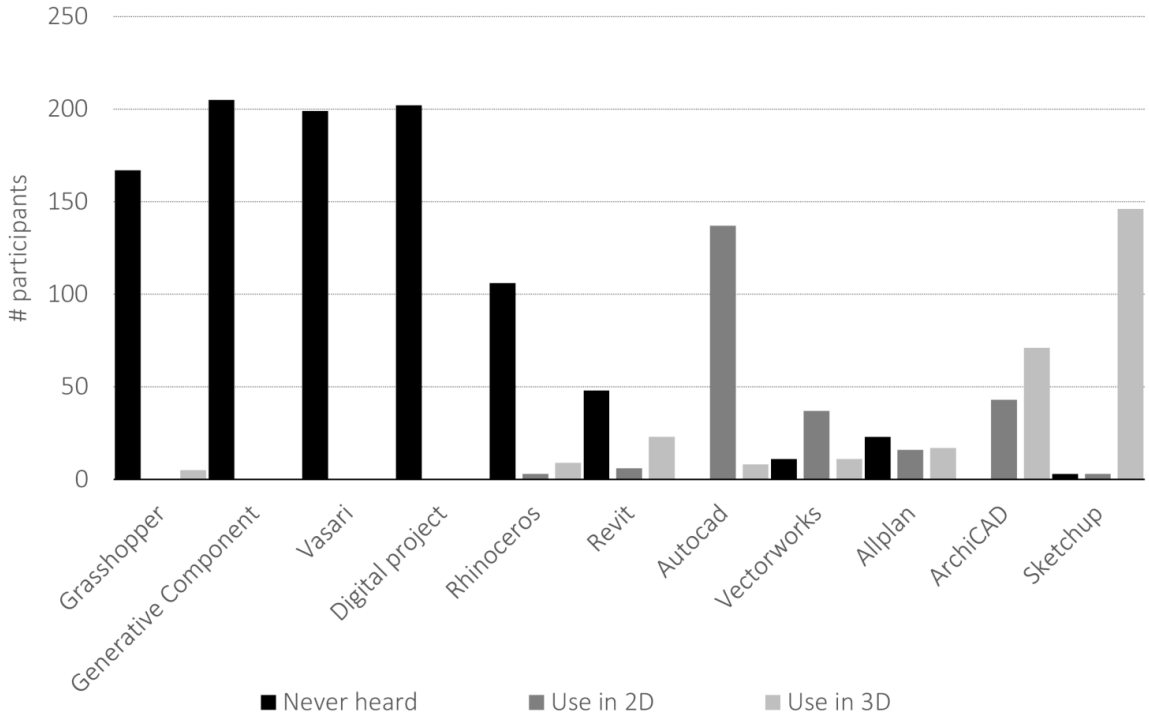


Fig. 8 : Usage des différents outils digitaux en Belgique. Les outils paramétriques étant Grasshopper, Generative

Bien que l'accès aux outils de paramétrique soit plus aisé aujourd'hui, il est évident que cette approche en est encore qu'au stade de maturation.

La majorité des outils de simulation et d'optimisation qui gravitent autour de ces plateformes de programmation visuelles en sont encore pour beaucoup au stade expérimental, et nécessitent souvent une certaine expertise. De plus l'élaboration d'une définition dans le but d'une optimisation n'est pas une chose triviale et nécessite un apprentissage personnel, ce qui ne favorise pas la motivation de la part des architectes à s'investir. Pour compléter le tout, même la communauté est encore partagée sur l'usage efficace de ces outils, un sujet qui représente d'ailleurs un vaste champ de recherche.

Une étude menée par Davis D [2012] par exemple, fait l'analyse de près de 600 définitions paramétriques issues du logiciel Grasshopper, offrant une vue sur la pratique du paramétrique en design et pointe du doigt quelques subtilités sur base de la longueur et la complexité des définitions étudiées. Plus de paramètres donne au concepteur plus de flexibilité mais chaque paramètre supplémentaire, comme mentionné précédemment, a un coût computationnel à prendre en compte. La plupart des définitions étudiées sont de petites tailles et ne représentent

en fait qu'une portion de chaine plus large, cela veut dire que les problèmes sont étudiés, en général, de façon ponctuelle, ne favorisant pas l'approche générative. De la même manière, un algorithme très long ou très complexe peut demander énormément d'efforts pour être compris, hors, il est également important de rappeler que l'un des avantages majeurs des méthodes computationnelles est la coopération.

D'autres s'intéressent à la compréhension et la mise en page de définitions paramétriques dans l'idée de collaboration entre acteurs. Davis & Burry [2011] démontrent les bénéfices du groupement dans la compréhension des algorithmes et Leitao [2012] discute sur les erreurs dues aux connections qui doivent se faire manuellement en programmation visuelle, le temps investit dans leurs maintenances et promeut une évolution vers la programmation textuelle.

En terme de production, il n'est pas rare d'apercevoir des exemples d'architecture paramétrique car ceux-ci sont en général des projets d'échelle gigantesque qui gagnent très vite le statut de monument ou d'attraction populaire. Hors les bureaux qui en sont responsables emploient des équipes multidisciplinaires travaillant ensemble dès la genèse du projet. Avec la popularité grandissante des Fablab, les productions à l'échelle du mobilier sont également devenues assez répendues. L'usage du paramétrique à l'échelle de l'habitation en revanche est très peu courante. La même enquête de A. Stals pointe dans son enquête que la majorité des firmes, en tout cas en Belgique, se compose de 1 à 5 personnes (voir fig. 9). Ces bureaux plus restreints assument à priori cette échelle de projet qui compose la grande majorité du paysage urbain, hors aucun d'entre eux ne fait usage du paramétrique.

Size of firms (number of people)	1 to 2	3 to 5	6 to 10	10 to 20
Percentage	41,0%	21,4%	10,6%	9,6%
Size of firms (number of people)	20 to 50	50 to 100	> 100	Other
Percentage	4,0%	4,0%	6,2%	3,1%

Fig. 9 : Distribution des tailles de bureaux d'architecture, en nombre d'employés, en Wallonie

Théorie

Introduction

L'optimisation, fait partie d'un ensemble comprenant également la définition paramétrique du projet et son évaluation par les différentes simulations. Le tout forme une boucle dont l'architecte est libre de sortir au moment jugé opportun (fig.10)

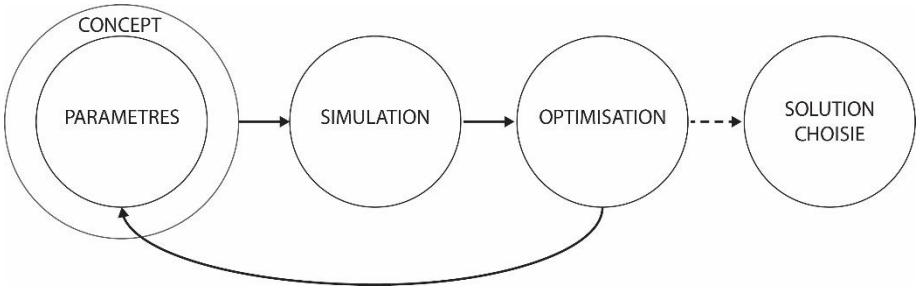


Fig. 10: Boucle d'itération des étapes d'un projet

Le solveur est l'outil responsable de l'optimisation. Il aura donc comme objectif de diriger l'exploration des paramètres vers de meilleures performances du projet. De manière théorique, le but d'une optimisation est soit de minimiser soit de maximiser une valeur, dans ce cas-ci, le résultat de la, ou des, simulations. Pour se faire le solveur, fait appel à un algorithme qui sera chargé de trouver les valeurs optimales pour chaque paramètre en moins de temps possible.

Mise en place de la définition paramétrique

Elle consiste en la définition de l'algorithme par programmation visuelle et de ses paramètres. Cette étape est cruciale et nécessite une expertise particulière quant aux connaissances relatives au processus. Effectivement, l'architecte tentera de minimiser le nombre de paramètres afin de faciliter la recherche. Les paramètres doivent également être définis de manière logique et non aléatoire de sorte à pouvoir extrapoler les résultats. La qualité du modèle de recherche déterminera en grande partie l'efficacité de l'exploration et le temps de simulation.

Le temps en optimisation est une denrée précieuse. Etant donné le temps que peut prendre une simulation, le choix de l'algorithme d'optimisation est lui aussi crucial [Wortmann, T., 2016]. Il dépendra en outre du type de problème et du nombre de paramètres. Ces aspects sont étudiés plus en profondeur à travers les différents cas pratiques.

Optimisation

L'optimisation consiste en la recherche d'un optimum pour une fonction donnée. Le Larousse [2000] procure la définition suivante pour la notion d'optimum: « état, degré de développement de quelque chose jugé le plus favorable au regard des circonstances données ».

Mathématiquement, l'optimisation consiste en la modélisation du problème par l'identification des variables de décision qui décrivent le système, et en la définition d'une fonction dites « objectif » qui décrit quantitativement l'état d'un système. L'état le plus favorable du système est appelé optimum et est obtenu par maximisation ou minimisation de cette fonction « objectif » (voir fig. 11). Les contraintes représentent les égalités ou inégalités que les variables doivent satisfaire afin d'être admissible [Blanchard, 2018].

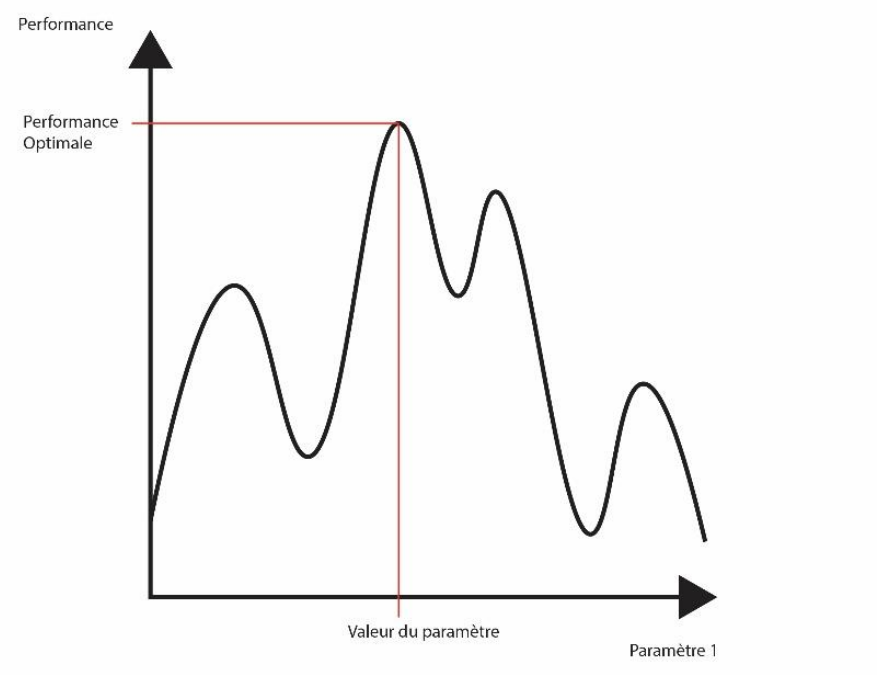


Fig. 11: Recherche de l'optimum pour une fonction donnée

Il arrive en optimisation que la relation mathématique qui lie les différents paramètres soit inconnue. On parle alors d'optimisation de type « boite noire ». Afin de trouver l'optimum il est alors nécessaire de faire le calcul ponctuel de certains ensembles de paramètres afin de diriger ce qu'on appelle l'exploration du dit « paysage de solutions ». L'optimisation en boîte noire est ce qui va intéresser l'architecte dans son exploration.

Dans un souci de compréhension, commençons par un problème à 1 dimension, c'est-à-dire un seul paramètre. Si la fonction était connue il serait facile de trouver l'optimum, hors elle ne l'est pas, il s'agit donc d'une recherche, « dans le noir », où chaque paramètre doit être évalué de façon empirique à travers une simulation.

Donc, concrètement la recherche de l'optimum consiste à rechercher les pics de cette fonction sans connaître la relation mathématique qui lie les paramètres au résultat de performance. Il faut donc choisir des valeurs de variable à tester et après un nombre déterminé de tests, définir les meilleures solutions (voir fig. 12).

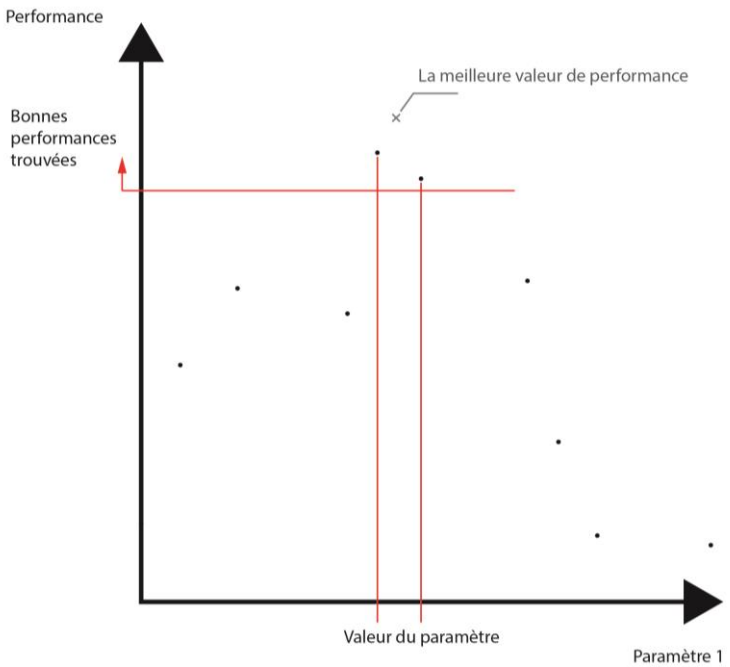


Fig. 12: Recherche de l'optimum en boîte noire. Les points sont choisis de manière aléatoire et évalués en terme de performance.

Réaliser une simulation pour l'ensemble des variables comprises dans le domaine restreint défini pour chaque paramètre signifie adopter une méthode dite de Force Brute et est hors de question de par le temps que cela prendrait. Il faut donc faire appel à un algorithme qui déterminera le chemin le plus court à prendre vers les pics d'une fonction qui nous est inconnue. Il est fait allusion dans la littérature [Rutten, 2013] d'exploration du paysage de solutions. Ces termes proviennent d'une simplification du problème en 3 dimensions ou le « paysage de solution » fait penser à une carte topographique. La figure 13, permet d'appréhender un tel exemple d'optimisation à 2 paramètres P1 et P2. A chaque duo de variables correspond une valeur z, résultat de la simulation S. L'ensemble des solutions compris dans le domaine des variables est alors représenté sous la forme d'un paysage topographique de solutions où les pics représentent les objectifs à atteindre. D. Rutten (2014) fait l'analogie entre cette recherche d'optimum et la recherche d'un sommet de montagne en conditions brumeuse où l'algorithme représente la stratégie de recherche.

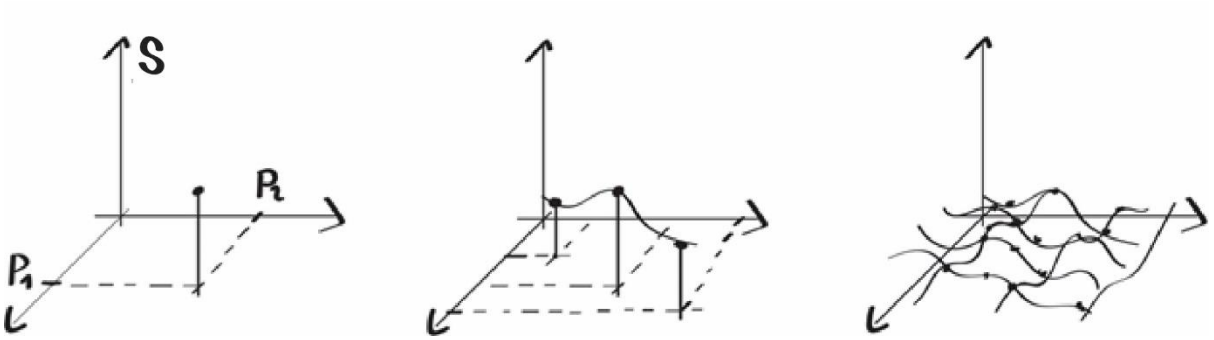


Fig.13 : Exemple d'optimisation à deux paramètres P1 et P2

Une série d'algorithmes spécifiques aux problèmes d'optimisation en boîte noire a été développée au fil des années et représente encore un domaine d'étude important en mathématique (voir fig. 14). Une compétition, la BBComp (Black Box Optimization Competition), a d'ailleurs lieu tous les ans dans le but d'alimenter la recherche à échelle mondiale. La figure 14 montre clairement les approches différentes que peuvent avoir les différents algorithmes pour un même problème. Chaque point blanc représente une simulation. Certains algorithmes tel que le DAKOTA/DIRECT ont une approche très mathématique et divisent de manière égale le paysage de solution, d'autres comme le DAKOTA/EA font usage de force brute et coûtent alors très cher en simulation, enfin certains comme le SID-PSM sont plus subtils dans leurs fonctionnements et parviennent à une solution très rapidement.

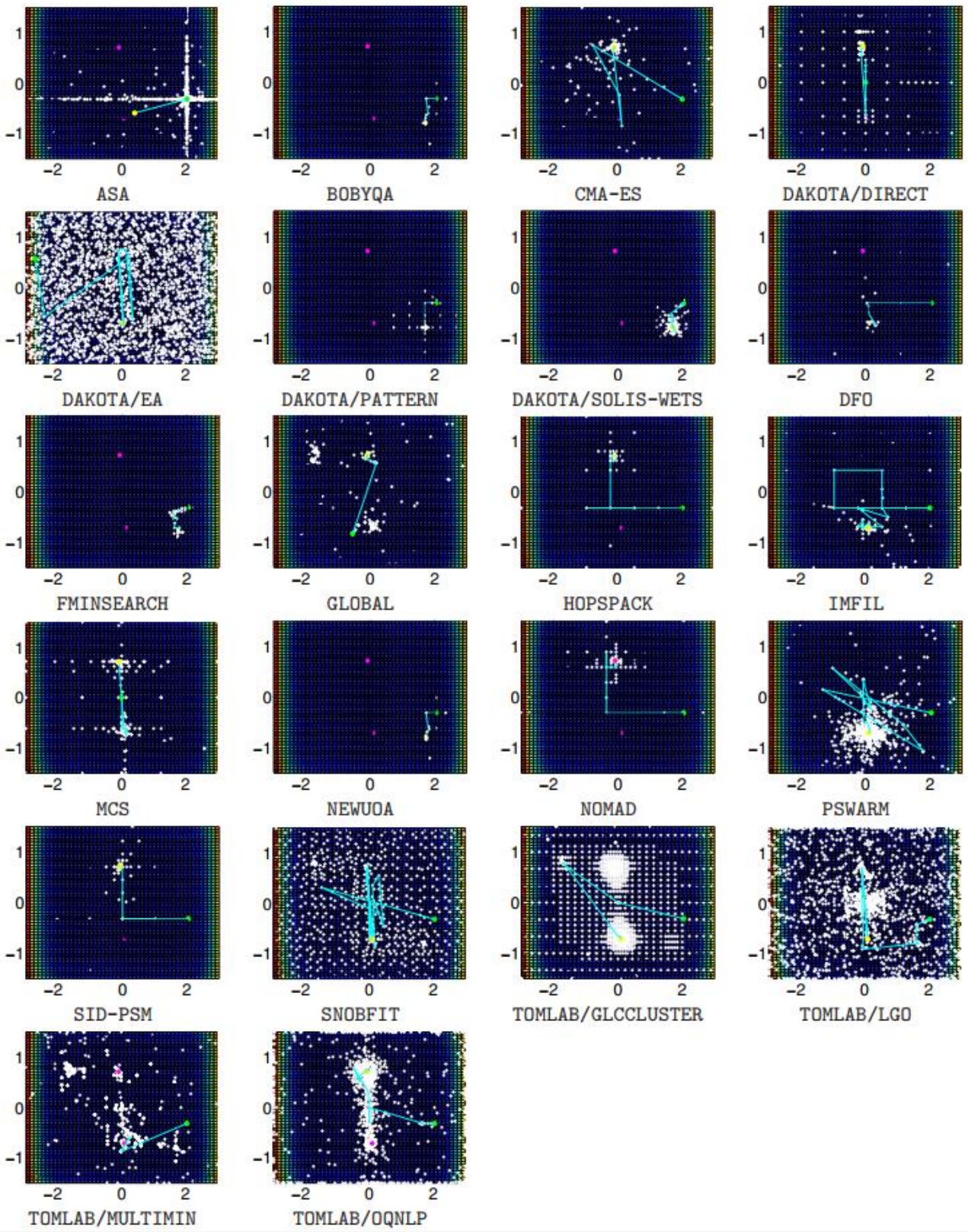


Fig. 14: Solver search progress for test problem camel6 [Rios and Sahinidis, 2013]

On remarque également que la convergence n'a pas toujours lieu au même endroit, en effet il arrive que la recherche reste bloquée dans ce qu'on appelle l' optimum local (voir fig. 15), c'est pourquoi le choix de l'algorithme est primordial au succès de l'optimisation.

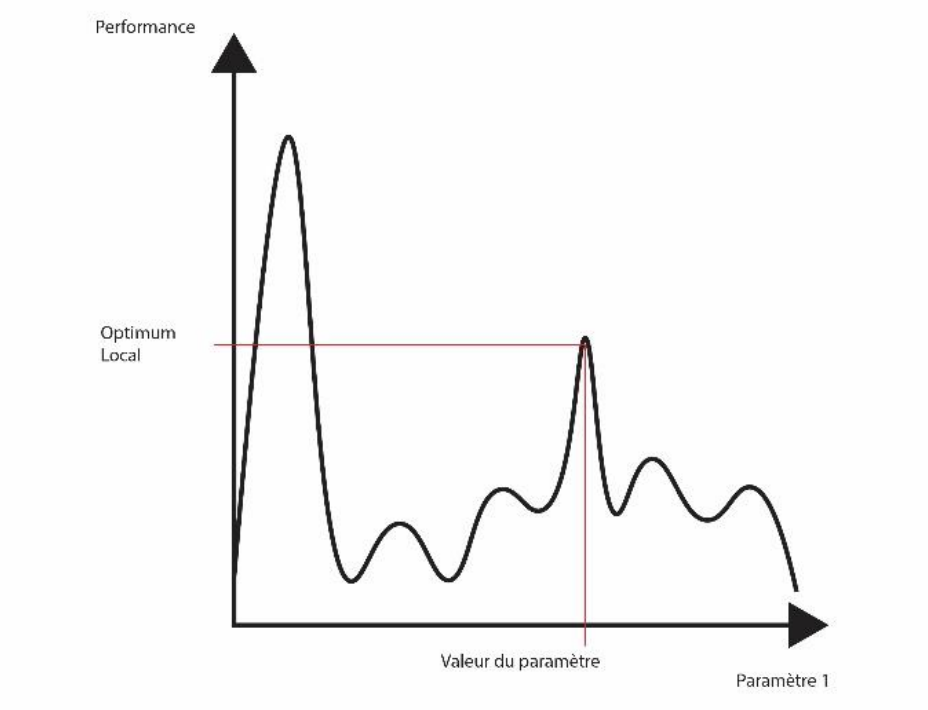


Fig. 15 : Optimisation biaisée par un minimum local par faute de mauvaise exploration et donc de choix d'algorithme

Pour reprendre Wortmann [2017], trois types d'algorithmes émergent actuellement : les métaheuristique, la méthode de recherche directe et l'optimisation basée sur modèle.

Les métaheuristique

Les métaheuristique sont souvent assimilés à l'intelligence artificielle. Ils ont pour but d'étendre les capacités des algorithmes dits heuristiques comme par exemple le « Greedy Search » dont la fonction est de trouver de bonnes solutions de manière grossière à travers un vaste paysage de solutions et ce à moindre coût computationnel. La métaheuristique se prête bien aux problèmes de boite noire étant donné qu'ils n'ont besoin que de très peu d'informations pour fonctionner. Sean Luke [2015] les définit comme un sous ensemble des méthodes stochastiques, c'est-à-dire qui se basent sur une recherche aléatoire (voir fig. 16). Les algorithmes propres à cette méthode s'inspirent souvent de la nature, on y retrouve notamment les Algorithmes

Génétiques, le « Particle Pwarm » qui fait lui-même partie d'un sous ensemble appelé « Swarm Intelligence » ou « le Simulated Annealing ».

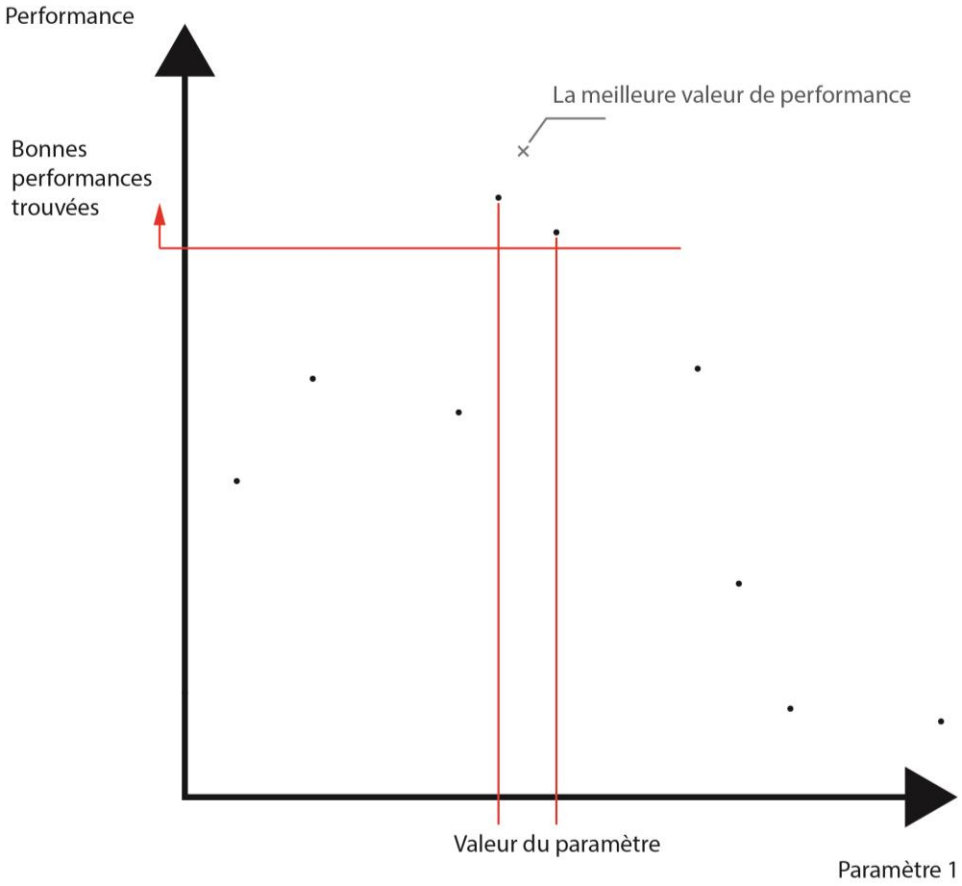


Fig. 16 : Recherche stochastique de l'optimum.

La recherche directe

La recherche directe est une méthode qui comme les métaheuristiques ne nécessite aucune information quant à la fonction à optimiser. Elle ne fait aucune approximation et se contente d'analyser de manière systématique un ensemble de solution pour se refermer au fil des itérations sur les solutions les plus prometteuses. Les algorithmes DIRECT et SBPLEX seront discutés par la suite.

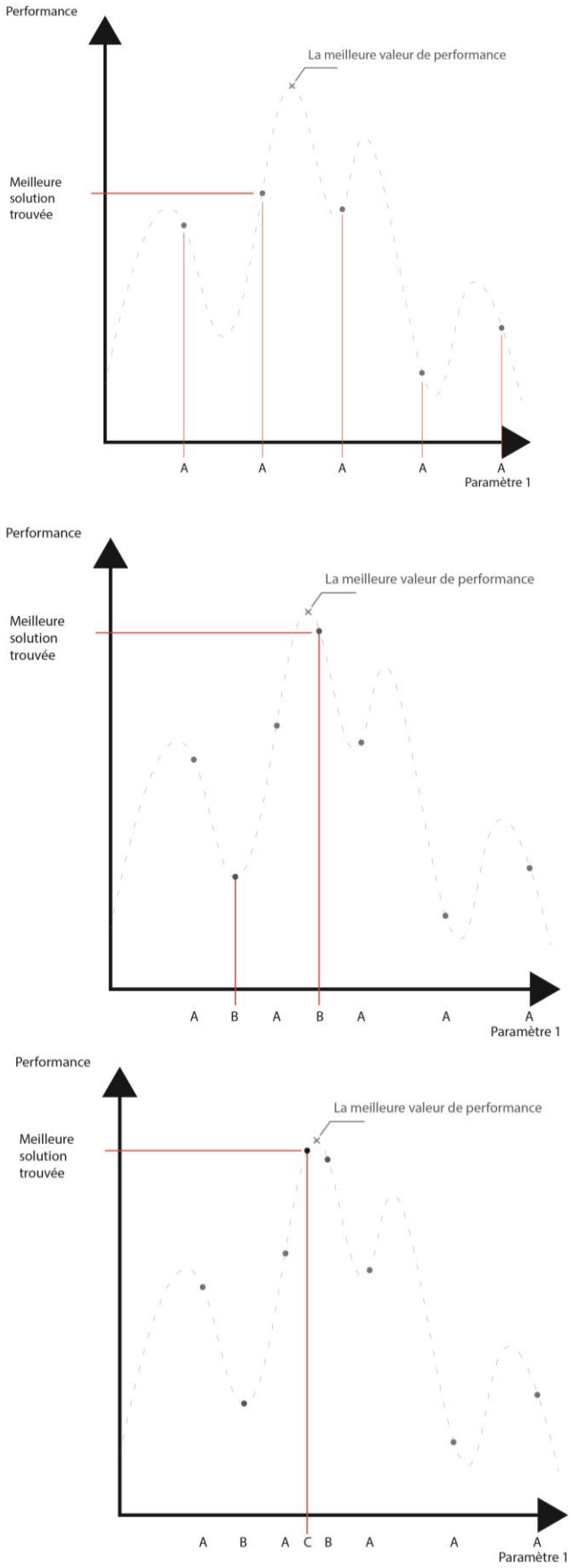
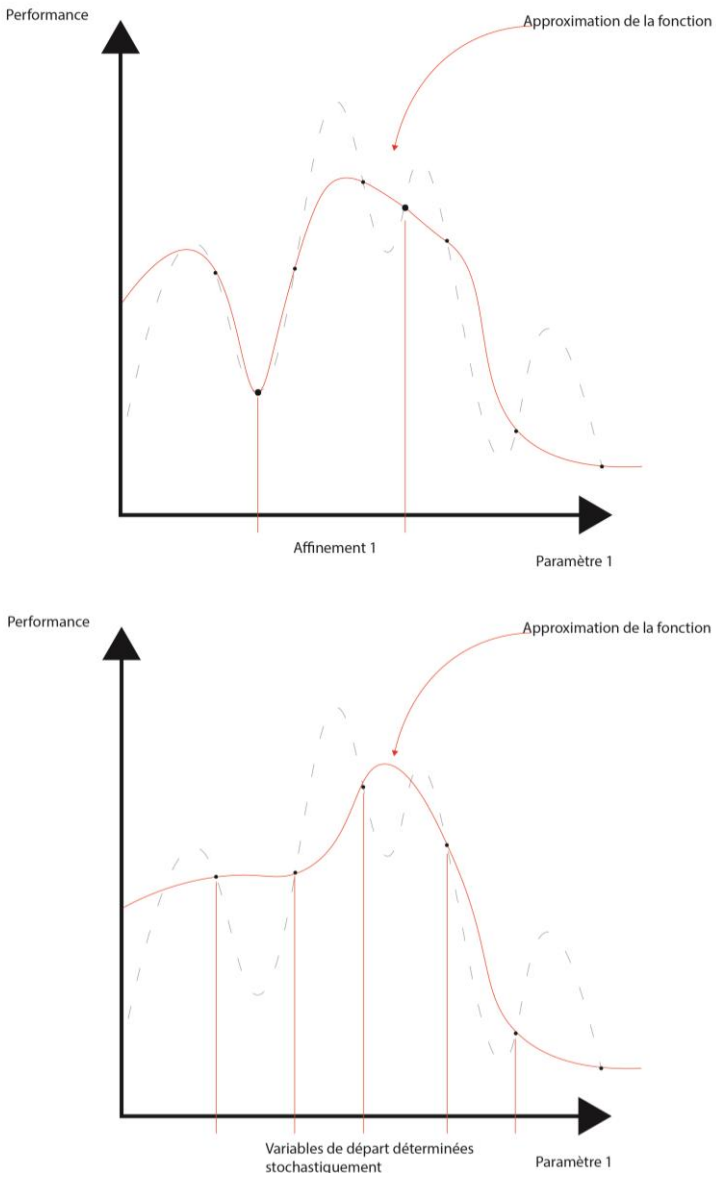


Fig. 17 : Exemple de recherche d'un optimum par recherche directe.

L'optimisation basée sur modèle

L'optimisation basée sur modèle a pour but d'extrapoler la fonction à optimiser ou « fonction objectif » sur base de simulations préalables. Le choix des points de départ se fait soit de manière stochastique soit de manière déterministe en fonction de l'algorithme choisi. Le nombre de simulation détermine alors la précision de l'extrapolation (voir fig. 18). Cette méthode est très efficace de par son économie [Yang, D., 2016] mais ne convient pas à tous les problèmes. COBYLA, BOBYQA, et RBFOpt sont trois exemples d'algorithmes qui seront traités dans la suite du travail.



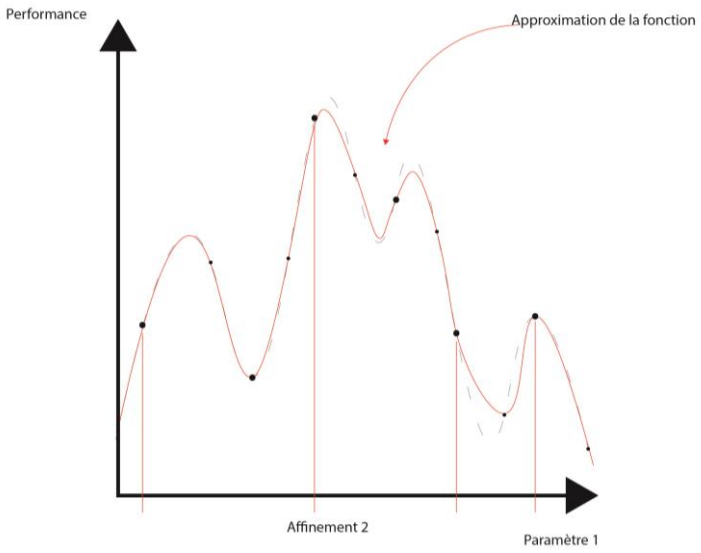


Fig. 18 : Exemple de recherche d'optimum par méthode dite model-based

Il existe donc pour chaque approche une série d'algorithmes. Ceux-ci sont accessibles à travers une interface graphique appelée « solveur ». La figure 19 reprend l'ensemble des solveurs les plus populaires dans Grasshopper ainsi que les algorithmes qu'ils proposent.

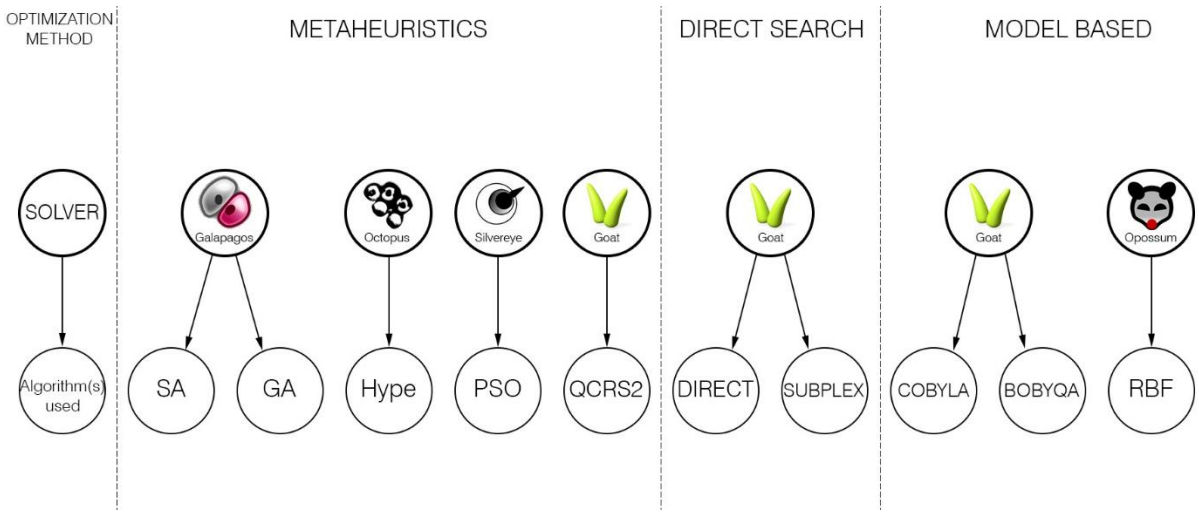


Fig. 19 : Liste des solveurs disponibles les plus populaires sur Grasshopper, les algorithmes qu'ils contiennent et leur classement d'après la catégorie d'algorithme

Dans Grasshopper, le solveur apparaît sous forme d'un nœud qui donne accès aux différents paramètres tels que le nombre de simulations, le choix de l'algorithme, le choix de rechercher un maximum ou un minimum, etc… Ce nœud est bien sûr connecté au résultat de la simulation et agit directement sur les paramètres de départ afin de boucler le cycle de recherche (voir fig. 20). Cette interface est assez similaire à tous les solveurs.

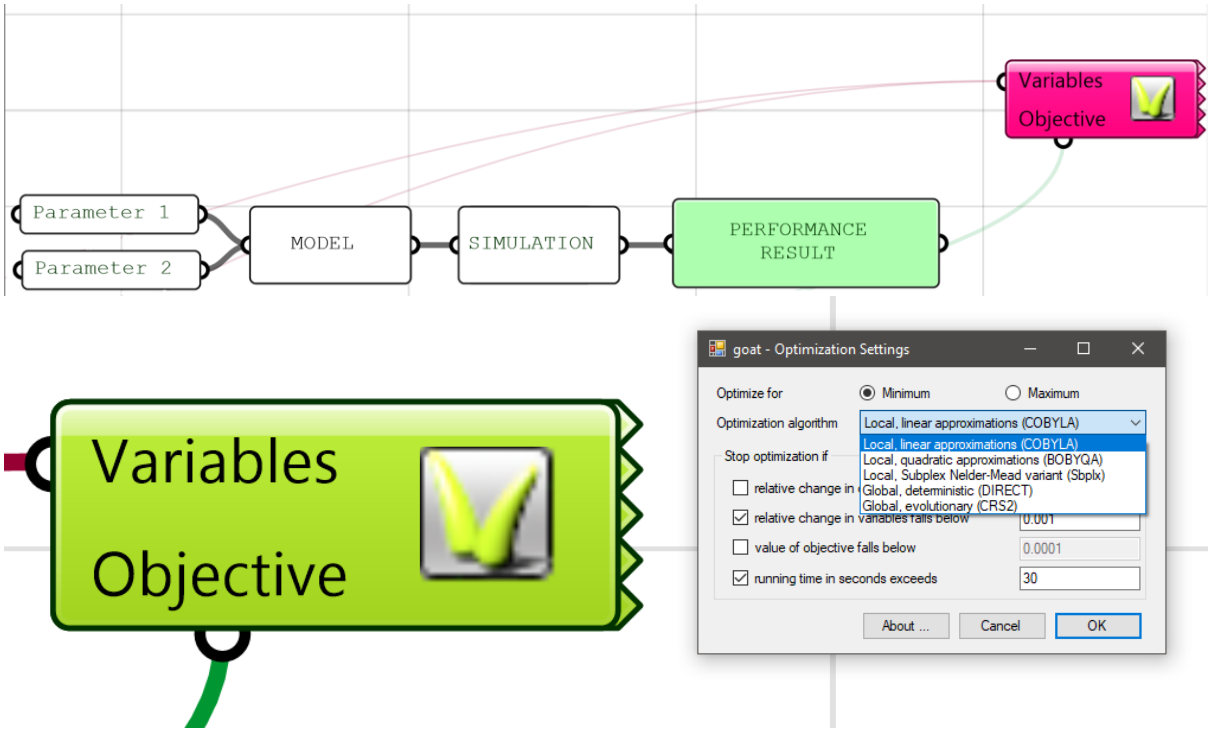


Fig. 20 : Présentation de l'interface du solveur GOAT à l'intérieur du logiciel Grasshopper

Le choix de l'algorithme

En dehors de la méthode utilisée, les algorithmes peuvent être classés en 2 catégories, l'optimisation à objectif simple et l'optimisation à objectifs multiples.

Optimisation à objectif simple

La liste qui suit reprend une série d'algorithmes qui seront analysés par la suite à travers un cas d'étude :

- GA (Genetic Algorithms) : Cet algorithme applique les principes biologiques de mutation, sélection et héritage génétique pour trouver les solutions optimales. Une population de départ, d'un nombre d'individus déterminé, est dispersée aléatoirement à travers le paysage. Les individus avec les meilleurs résultats développent eux même une population de façon locale et ainsi de suite dans l'espoir que leur descendance se rapproche des pics, donc des optimums. A chaque génération, les solutions gagnent alors en résolution. Les développements locaux de certains individus, s'ils sont bien dispersés, peuvent favoriser l'exploration mais l'inverse est possible et l'algorithme peut rester coincer autour d'un maximum local [Rutten, 2013].

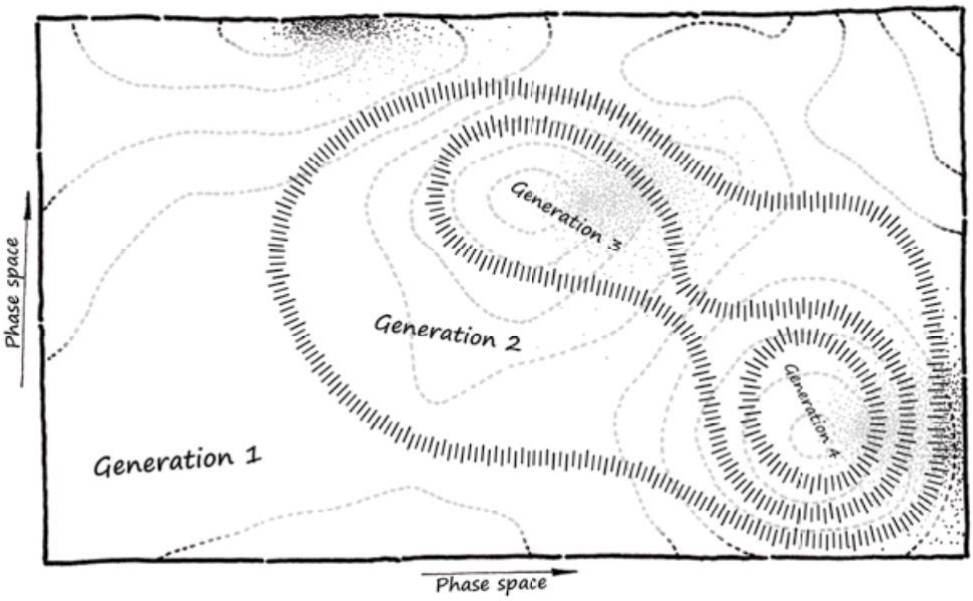


Fig. 22 : Représentation schématique d'une exploration de paysage à l'aide du SA [Rutten, 2013]

- SA (Simulated Annealing) : Cet Algorithme est inspiré d'un processus physique thermodynamique appelé le recuit du métal. Il consiste en la cuisson d'un métal suivi d'un refroidissement dans le but d'agrandir la taille des cristaux de métal et de ce fait rendre le matériau plus résistant et atteindre une nouvelle configuration des atomes à basse énergie. A travers cette métaphore, chaque solution de l'espace représente un état d'énergie du système. L'optimum consiste donc à trouver cet état d'énergie le plus bas. Le choix des configurations à analyser se fait sur base d'un algorithme appelé le Metropolis-Hastings qui choisit de manière aléatoire de nouvelles configurations dans des zones locales du paysage jugé comme ayant du potentiel par une série d'équations décrivant le processus du recuit (voir fig. 22) [Brownlee, 2011]

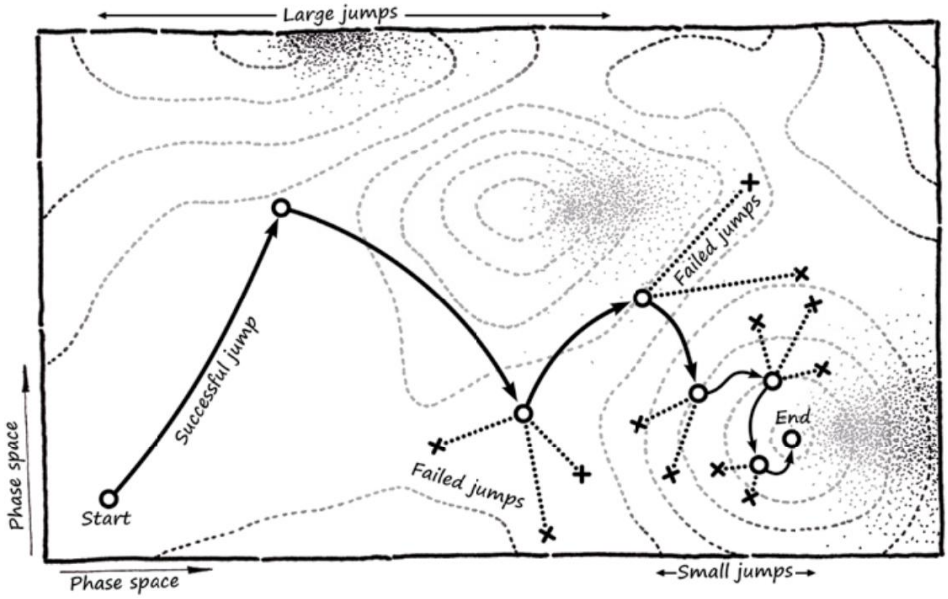


Fig. 22 : Représentation schématique d'une exploration de paysage à l'aide du SA [Rutten, 2013]

- PSO (Particle Swarm Optimization) : Appartient au sous-ensemble « Swarm Intelligence ». Cet algorithme est inspiré du mouvement groupé d'animaux tels que les oiseaux ou les bancs de poissons. L'idée est apparue du fait qu'à travers l'évolution, ces animaux ont affiner leurs trajectoires pour obtenir le meilleur résultat. L'algorithme fonctionne de la façon suivante, une taille de population de base est définie et dispersée dans le paysage comme pour les algorithmes génétiques et chacun des individus se voit attribué une vitesse de déplacement. Cette vitesse correspond à la distance que l'individu peut parcourir après chaque simulation. Les particules vont voyager sur base des résultats obtenus pour les particules déjà simulées dans le but que, sur la durée, toutes les particules convergent vers un même optimum [Brownlee, 2011]. L'algorithme

à l'avantage que, pour converger, chaque particule doit explorer un ou plusieurs morceaux de paysage sur le chemin. Pour reprendre l'analogie de Rutten, c'est comme si une équipe de grimpeurs s'était dispersé à travers une topographie dans le but de se retrouver au sommet le plus haut. Ces grimpeurs sont aveugles mais peuvent communiquer ensemble et préciser leurs altitudes et leurs coordonnées tous les x temps. Cela signifie que si sur le chemin du pic supposé le plus haut, un randonneur obtient une meilleure valeur d'altitude, le reste se redirigera et ainsi de suite permettant un excellent potentiel d'exploration.

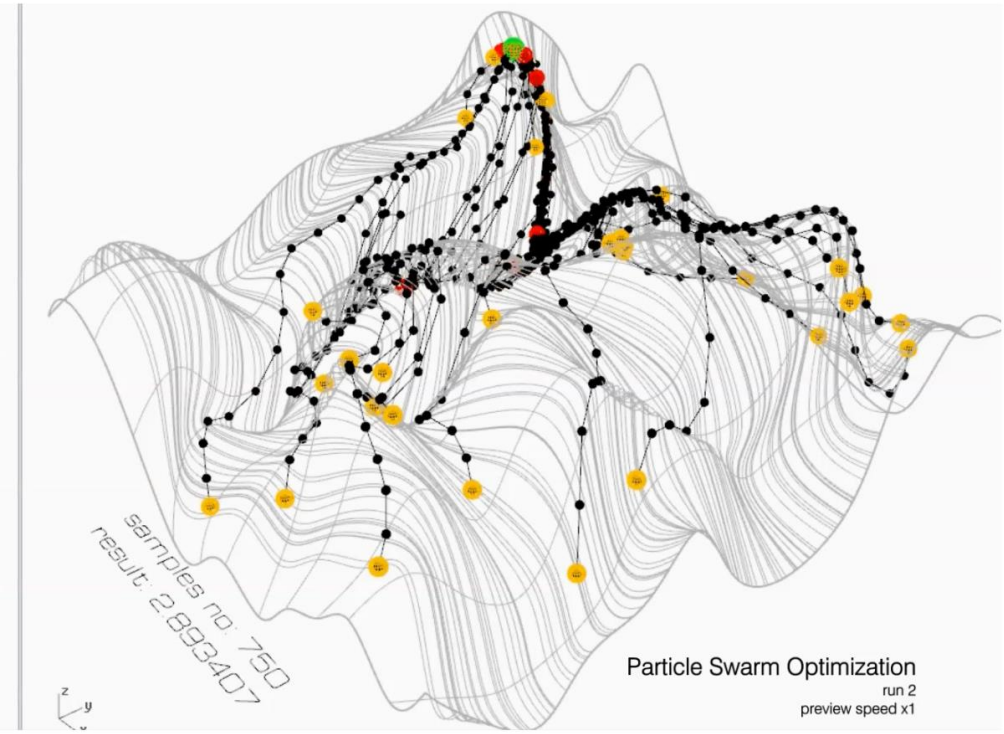


Fig. 23 : Exemple d'exploration au moyen de PSO

- CRS2 (Controlled Random Search) : Cet algorithme est assez similaire aux GA dans la mesure où il démarre également par une population distribuée de manière aléatoire sur le paysage mais observe d'autres règles de convergences vers l'optimum. GOAT ne permet pas de régler la taille de la population de départ à travers l'interface et est par défaut règlée à 10[Kaelo, 2006]. Cette technique n'est pas très populaire en recherche car elle présente peu de consistance dans ses résultats [Eligius et al. 2001].

- DIRECT : Cet algorithme au contraire des métaheuristiques ne se base pas sur une population aléatoire mais sur une analyse déterministe du territoire, cela veut également dire que toutes les optimisations aboutiront au même résultat. Le nom DIRECT est en fait un raccourci linguistique pour « DIviding RECTangles » qui décrit parfaitement la stratégie d'approche de l'algorithme (voir fig. 24) [Finkel, 2003].

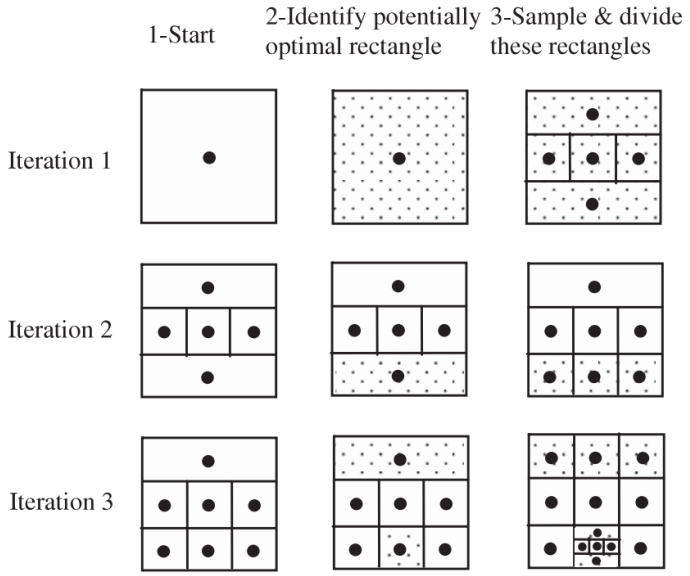


Fig. 24 : Exemple d'exploration du paysage par division par DIRECT[Lang et al., 2014].

- SBPLX : cet algorithme est basé sur un algorithme appelé Subplex basé lui-même sur un algorithme appelé Nelder-Mead publiée par John Nelder et Mead en 1965. Le Nelder-Mead est basé sur la réduction d'un objet géométrique appelé polytope (un polyhèdre en hyperdimension) qui possède n+1 sommet, n étant le nombre de paramètres à prendre en compte. Le polytope se déplace, se déforme et se réduit progressivement jusqu'à ce que ses sommets se rapprochent d'un point où la fonction est localement minimale. Le polytope de départ est réglé de manière arbitraire par SBPLX. Le NelderMead a le désavantage de perdre en efficacité pour les fonctions plus complexes [Nelder, Mead, 1965].

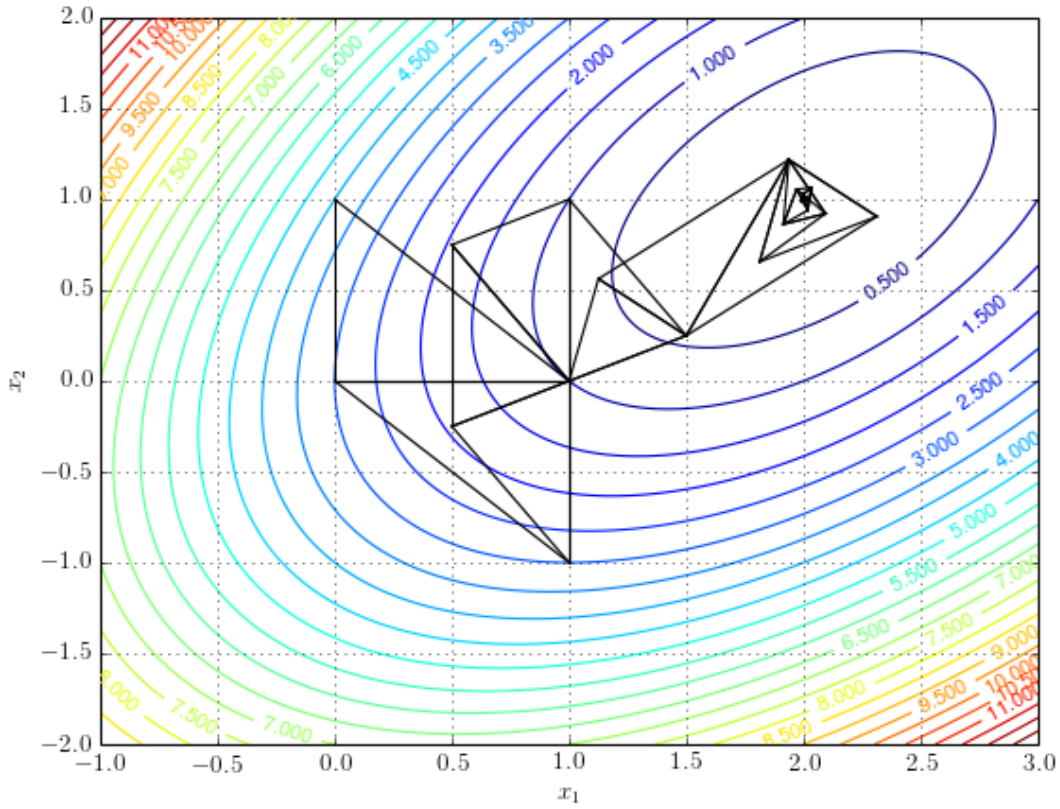


Fig. 25 : Exemple de convergence sur l'optimum par NelderMead [Joglekar, 2016]

- COBYLA (Constrained Optimization By Linear Approximations) and BOBYQA (Bound Optimization By Quadratic Approximation): Ces algorithmes extrapolent le paysage par une série d'approximations linéaires dans le cas de COBYLA et quadratique dans le cas de BOBYQA [Powell, 2009, 98] .Les 2 algorithmes ont été modifiés et aucune info n'est disponible sur le choix de la population de départ.
- RBFOpt (Radial Basis Function Optimization): Comme pour COBYLA et BOBYQA, il s'agit d'un algorithme permettant l'extrapolation des données mais cette fois ci sur base de fonctions radiales. L'avantage ici est que son solveur, Opossum, permet à travers son interface l'interaction directe avec l'algorithme sous réserve de connaissances suffisantes en mathématique. Par défaut, la population initiale est sélectionnée de manière stochastique mais aucune information n'a été trouvé quant à la taille de population de départ.

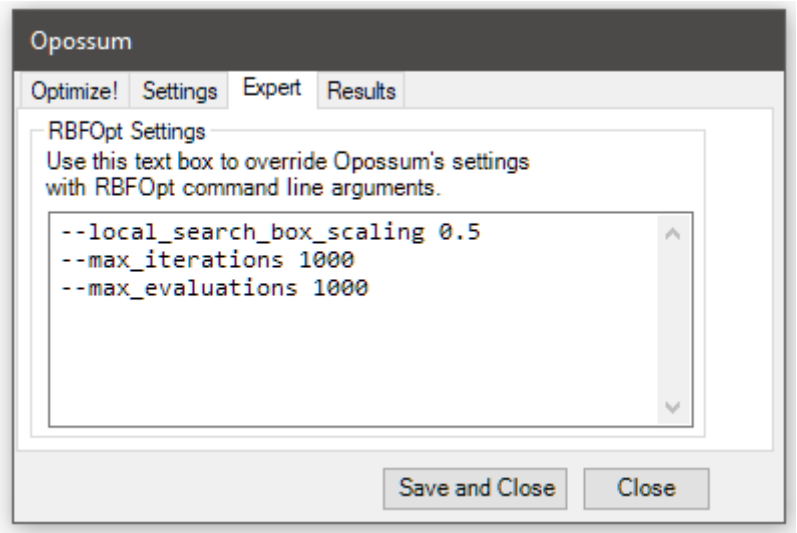


Fig. 26 : Interface de programmation d'Opossum permettant l'intégration des paramètres sous réserve d'expertise informatique.

Les algorithmes destinés à l'optimisation à objectif simple ne peuvent optimiser qu'une seule valeur à la fois hors, comme il a été fait mention plus tôt dans le travail, il existe une multitude d'objectifs à atteindre dans le cas du projet d'architecture. Bien que l'étude de Davis [2012] indique un usage isolé pour chaque élément du projet de manière séparée, l'enquête menée par Cichocka [2017] montre un intérêt de la part des utilisateurs pour l'optimisation à objectifs multiples. Mis à part Octopus et ses algorithmes, SBEA-II et HYPE, tous les solveurs disponibles sur Grasshopper n'acceptent que 1 seul objectif. Il faut alors passer par une méthode appelée la méthode naïve [Luke , 2018].

Elle permet d'optimiser plusieurs objectifs à l'aide d'un solveur qui utilise des algorithmes à un seul objectif. L'astuce est de développer une relation entre les objectifs. Cette technique présente néanmoins plusieurs défauts : il faut être capable de définir la valeur de chaque objectif les uns par rapport aux autres, les relations ne sont sans doute pas linéaires et requièrent probablement un ajustement pour chaque fonction et chaque résultat de simulation s'exprime en unités d'échelles complétement différentes qu'il faut pouvoir compenser [Luke, 2018].

L'explication qui suit se base essentiellement sur le travail de David Rutten [2011]. Supposons un problème simple à 2 objectifs en opposition comme par exemple la quantité de lumière et le coût avec comme seul paramètre la taille de la fenêtre. Le but est de maximiser l'objectif 1 et de minimiser l'objectif 2, il faut donc trouver une relation entre ces 2 résultats.

F(x) = P1 – P2 avec :

- P1 : valeur de performance issue de la simulation 1
- P2 : valeur de performance issue de la simulation 2

Cette fonction est dite fonction pénalisante et doit être modifiée dans le but d'obtenir une cohérence dans le résultat. Par exemple, il est important de favoriser un minimum de lumière, tandis que l'augmentation de lumière, passé un certain niveau de luminosité a de moins en moins d'intérêt. Une fonction racine vient alors tempérer l'évolution du premier objectif (voir fig. 27).

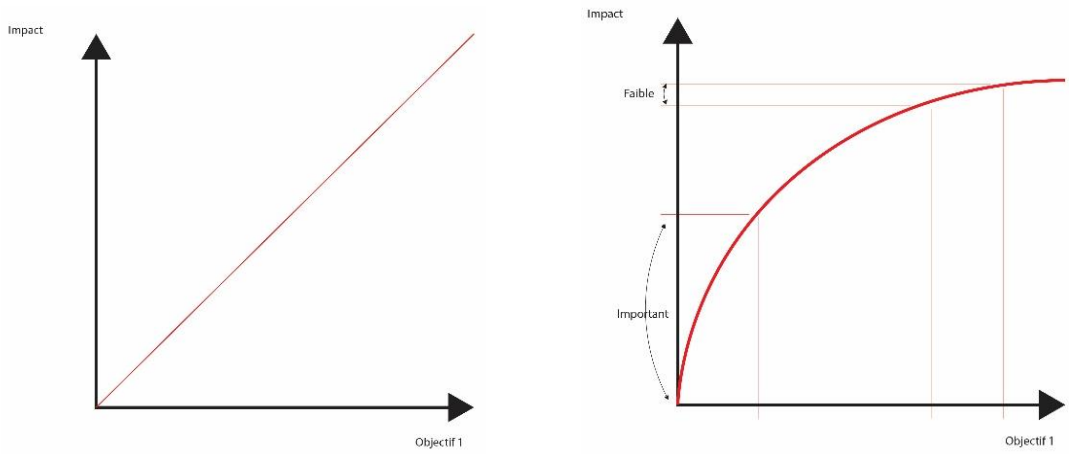


Fig. 27 : Manipulation de fonction en vue de tempérer l'impact sur l'optimisation globale

A l'inverse pour le coût, il existe un investissement de base et le client sera plus ou moins enclin à rajouter à cette somme de base. La relation dans ce cas-là ressemblera plus à une relation linéaire voir exponentielle (voir fig. 28).

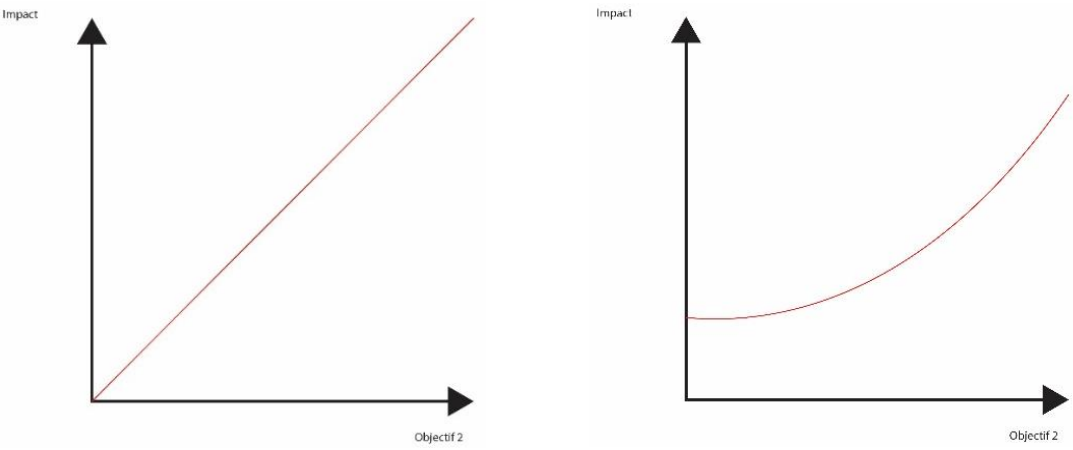


Fig. 28 : Manipulation de fonction en vue de tempérer l'impact sur l'optimisation globale

La fonction pénalisante prend alors la forme suivante :

$F(x) = \sqrt{(P1)} - [(P2)^{2} + i]$

« i » représentant l'investissement minimum de départ.

Il ne reste plus alors qu'à proportionner les valeurs à l'aide des facteurs a et b:

$F(x) = a \times \sqrt{(P1)} - b \times [(P2)^{2} + i]$

Ce résultat est donc une interprétation très subjective de plus, bien que le problème soit très réduit, la relation établie est fortement simplifiée et ne reflète pas la complexité de la réalité. Les désavantages liés à l'usage de la méthode naïve seront élaborés plus loin dans les cas pratiques.

L'autre aspect à considérer dans l'optimisation d'une telle fonction est également le choix de l'algorithme. La fonction pénalisante sur base des objectifs des paramètres et du projet peut présenter une discontinuité ne permettant pas à l'optimisation sur base de modèle par exemple d'être efficace. La discontinuité est due au manque de relations entre les différents objectifs et paramètres, un exemple serait une optimisation dont les objectifs sont la minimisation du poids de la structure et l'apport de lumière naturelle dans un projet où ces 2 aspects ont été traités complètement séparément. La fonction résultante est donc discontinue et non approchable (voir fig. 29)

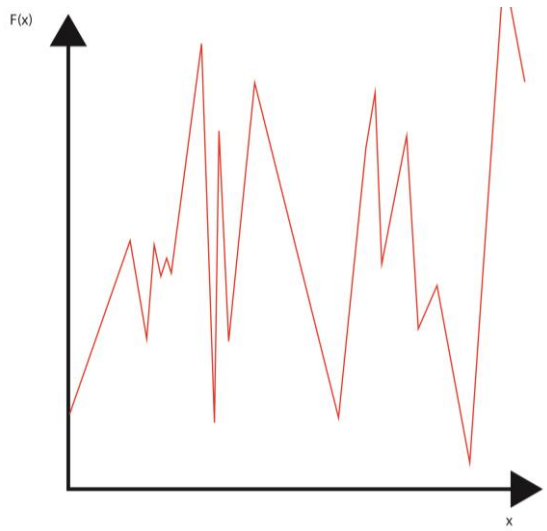


Fig. 29 : Exemple de fonction discontinue

Au-delà des problèmes de discontinuités, les paysages de solutions peuvent présenter des topographies de toutes formes (voir fig. 30) qui déterminerons l'efficacité ou non des différents algorithmes.

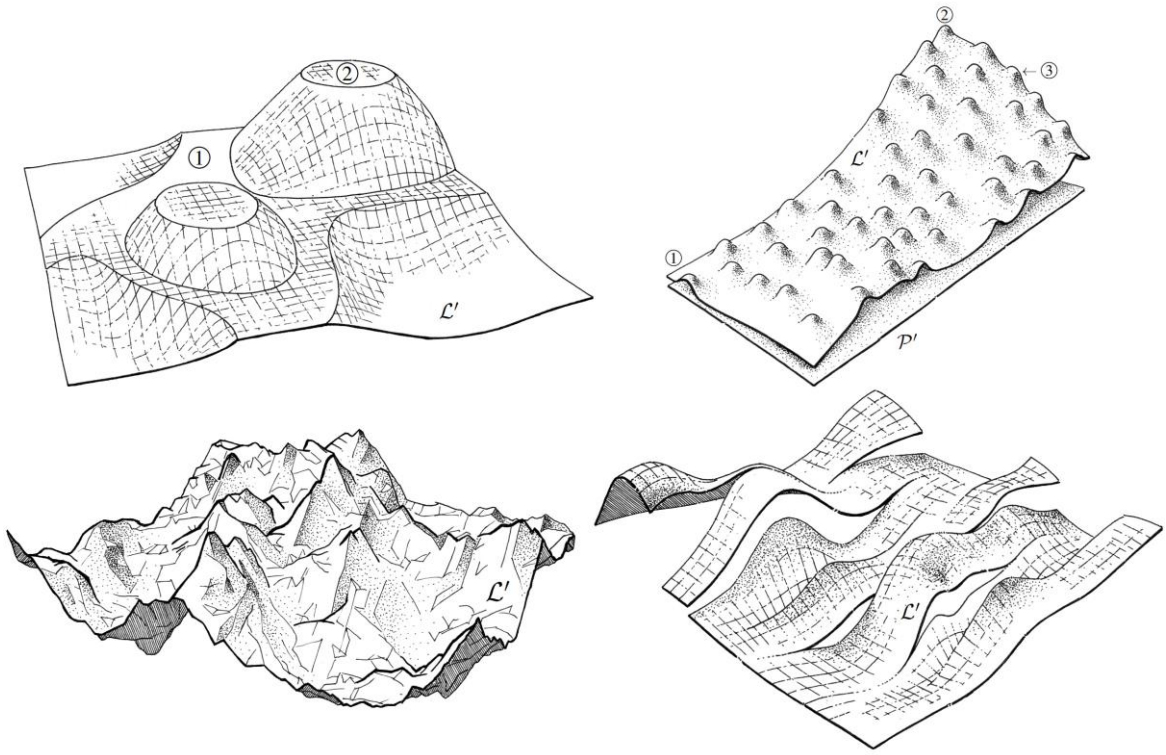


Fig. 30 : exemples de paysages de solutions décrits par D. Rutten [2014]

Cette méthode arbitraire est donc prône à l'erreur mais trouve néanmoins ses adeptes au sein de la communauté [Rutten, 2013]. Une redondance dans les problèmes étudiés permettrait d'affiner ces techniques au fil du temps.

Il existe certains algorithmes qui se basent sur cette démarche simplifiée comme par exemple le « Lexicographic Tournament Selection » [Luke J. 2018] pour lequel il suffit juste de préciser l'ordre d'importance des objectifs à atteindre.

Une autre approche serait une optimisation en deux temps. La réalisation d'une optimisation simple pour chaque objectif et une mise en superposition des données afin de déterminer graphiquement les zones de solutions prometteuse. Cette recherche sera couverte par la suite.

Les outils d'optimisation à objectif unique sont néanmoins plus simples à mettre en place et sont ceux qui observent le plus de développement aujourd'hui en architecture à travers

Grasshopper. Néanmoins, cette étape requiert de l'utilisateur de pouvoir aisément manipuler les fonctions. Une façon d'échapper au problème de la fonction pénalisante se situe au niveau des solveurs à objectifs multiples. Leur avantage est donc considérable. Le développement de tels outils est cependant très complexe et peu de solutions sont disponibles. Comme il a été dit plus haut, le seul solveur à objectifs multiples disponible sur Grasshopper pour le moment est Octopus [Vierlinger, 2013]. Celui-ci repose sur des algorithmes génétiques propres aux méthodes métaheuristiques et fait appel à la méthode Pareto pour sélectionner ses meilleurs candidats.

Optimisation à objectifs multiples

Dans la méthode précédente, la performance est calculée sur base d'un seul résultat, celui de la fonction pénalisante donnée. L'optimisation à objectifs multiples nous offre la possibilité de définir les bonnes solutions sur base des différents objectifs pris séparément.

Le principe du Pareto est le suivant: un individu A domine un individu B si A est au moins aussi bon que B pour tous les objectifs et meilleur que B pour au moins un objectif. L'ensemble des solutions qui ne sont jamais dominées forment ce qu'on appelle le front Pareto. Ceci est exprimé graphiquement à la figure 31. Le concept porte le nom de l'économiste italien Vilfredo Pareto, qui l'a utilisé pour décrire un état de la société dans lequel on ne peut pas améliorer le bien-être d'un individu sans détériorer celui d'un autre.

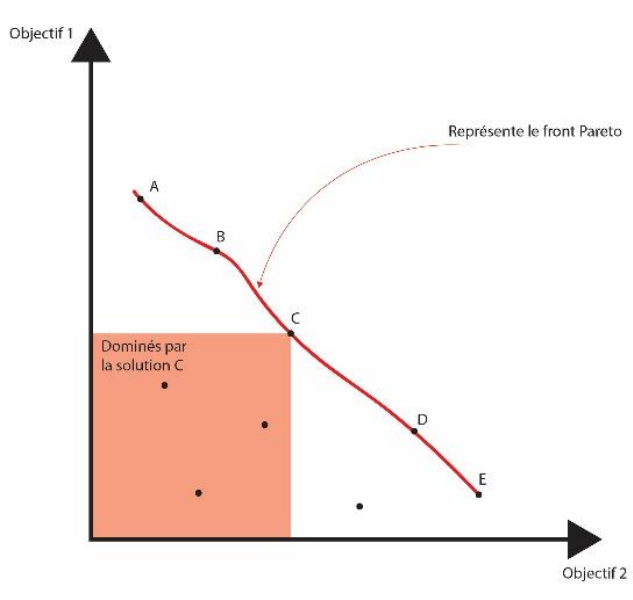


Fig. 31 : Front Pareto est la courbe reliant les solutions dominantes

Les algorithmes propres au multi-objectif sont majoritairement des algorithmes évolutionnaires. Cela s'explique par le fait qu'il est compliqué pour un ensemble de paramètres de toujours obtenir des bons résultats pour chaque objectif. Les relations sont parfois inexistantes d'un objectif à l'autre et tenter de déduire un optimum par extrapolation mathématique n'a alors pas de sens. On utilise alors des méthodes stochastiques qui couvrent de plus grands territoires ou éventuellement des méthodes directes étant donné que les résultats ne se basent que sur les performances calculées. Ceci permet une exploration efficace en paysage discontinu.

SPEA-II, l'algorithme utilisé pour Octopus est basé sur les algorithmes génétiques et se contentera à chaque nouvelle génération de recréer un front Pareto sur base des performances calculées. SPEA-II permet également d'étaler les solutions sur le front afin de favoriser l'exploration des paramètres au maximum. Pour se faire il impose une distance entre les différentes solutions (voir fig. 32). Ce processus de dispersion est intégré à l'algorithme et favorise la logique d'exploration sur l'exploitation. En effet, il est préférable d'obtenir des solutions diversifiées plutôt que de se retrouver avec une série de réponse quasi semblables car proches d'un pic. Le potentiel d'exploration est dès lors très intéressant comparé aux méthodes d'optimisation simple qui courent toujours le risque de rester bloqué sur un maximum local.

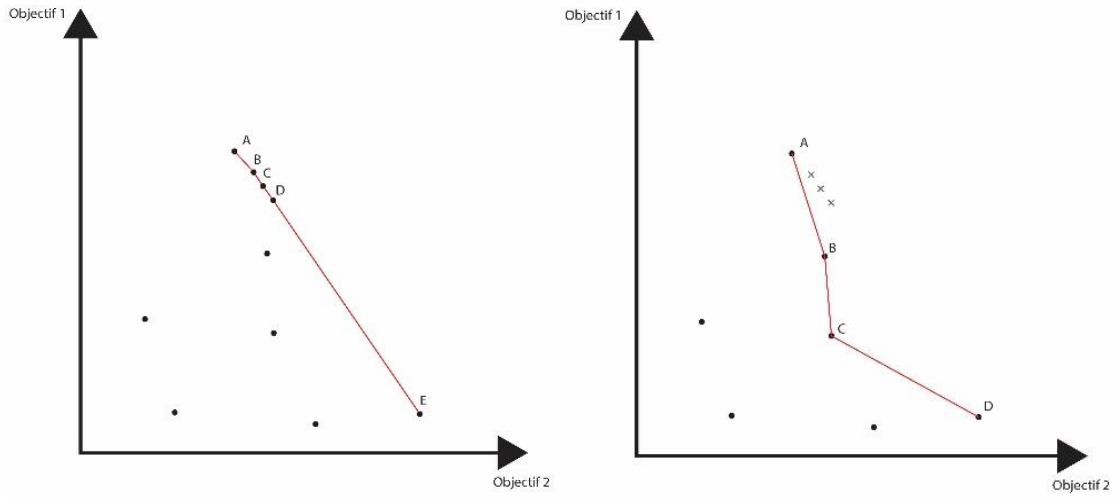


Fig. 32: Distribution des solutions le long du front Pareto

OCTOPUS, au fil du temps, a également intégré l'algorithme HYPE « algorithm for Hypervolume based many-objective optimization ». Hype est un algorithme particulièrement intéressant car, il permet une extrapolation mathématique du front Pareto sur base d'une méthode appelée Monte Carlo [Vierlinger, 2013]. Donc tout comme l'approche basée sur modèle, il est possible d'évaluer l'ensemble des solutions sur base d'un nombre limité de simulations, ce qui représente un gain de temps considérable.

Cas d’étude

Les cas d’études sont destinés à mettre en avant une série d’éléments à prendre en compte lors de la mise en place du processus d’optimisation et serviront de base de discussion par la suite.

Pour la configuration suivante : intel i7-4770k (OC : 4.3GHz) / RAM : 32Gb DDR3 / MB : Z87MX-D3H

Le processeur i7-4770k présente 4 unités de traitements (cœurs) pour 8 cœurs logiques (threads)

La version de Rhinoceros utilisée est la 6.3 et la version de Grasshopper est la 1.0

Temps de simulation

Le but est d’ici d’exposer une série de notions qui affectent de près ou de loin le temps de simulation. En dehors de la définition paramétrique du modèle, il existe des subtilités à prendre en compte lors de la mise en place d’une simulation. L’efficacité de la simulation est cruciale au bon déroulement de l’optimisation.

La simplicité du modèle permet une certaine généralisation des résultats.

Présentation du modèle

La simulation est réalisée à partir des plug-in LadybugTools, une collection de Plug-In dédiées aux simulations climatiques mentionnées plus tôt dans le travail. L’outil utilisé est « SunlightHoursAnalysis ». Il permet de décomposer une surface en sous-surfaces de tailles paramétrables qui seront sujettes à simulation (voir fig. 33). Il renvoie alors le nombre d’heures d’ensoleillement direct par an. Le résultat se présente sous forme d’un maillage coloré accompagné d’une légende elle-même paramétrable en terme de couleur, d’échelle, de position, etc...

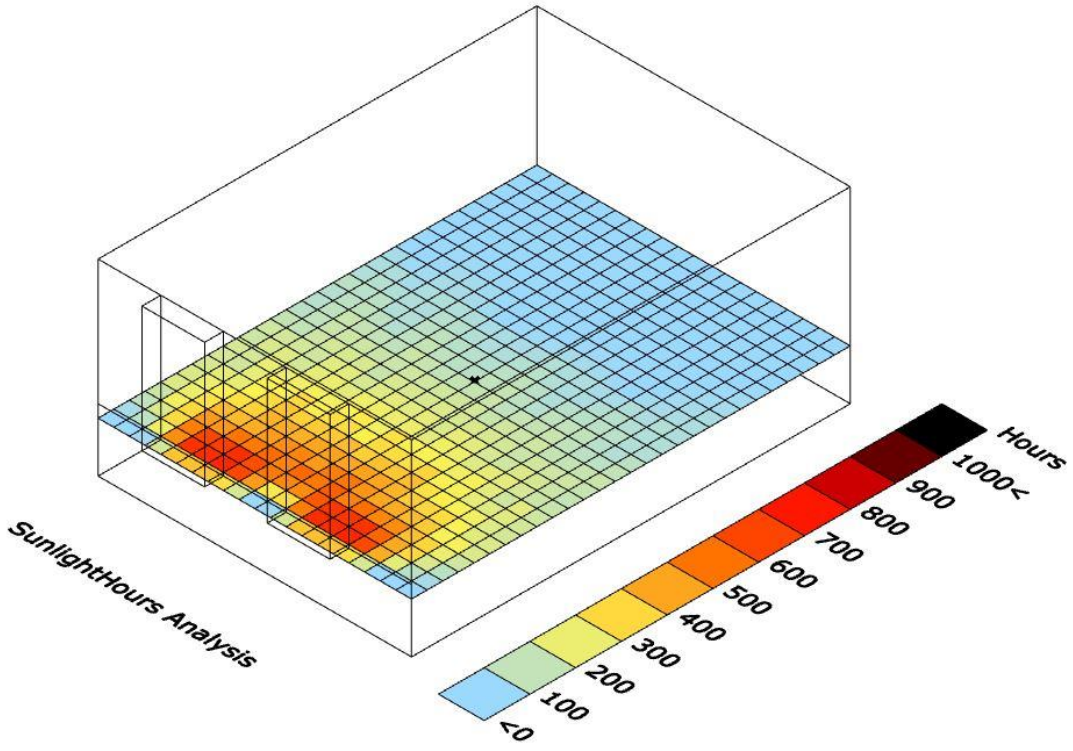
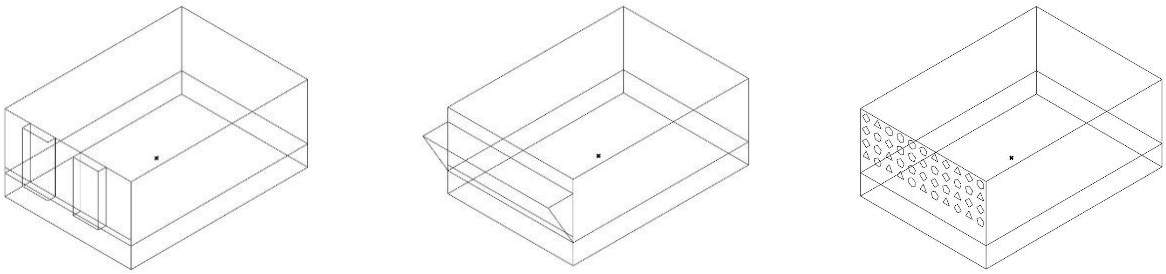


Fig. 33 : Exemple d'un retour visuel, dans une fenêtre de visualisation de Rhinoceros, d'une simulation réalisée à l'aide l'outil SunlightHoursAnalysis de LadyBug.

La définition du projet se fait sur base d'une chaîne mise au point par Chris Mackey et partagée sur la plateforme https://hydrashare.github.io/hydra/index.html. Celle-ci a été modifiée dans le but de servir les intérêts de cette démonstration.

Le modèle d'étude est un espace type boîte à chaussure fermée où l'une des faces verticales représente la variable. Ce paramètre « façade » peut prendre 3 formes prédéfinies différentes qui sont les types A, B et C ayant toutes les trois un niveau de complexité différente (voir fig. 34). La surface d'étude est placée à 80cm du sol, c'est-à-dire au centre de la hauteur d'une personne de 160cm.



Type A *Type B* *Type C*

Fig. 34 : Les 3 types de façade qui seront étudiés

Pour chaque type de façade, la surface d'étude est décomposée en sous surfaces carrées de tailles différentes dans le but d'appréhender le temps de simulation à petite et grande échelle et d'en déduire une relation.

Résultats

La figure 35 présente les temps récoltés en secondes en fonction du type de façade et du nombre de sous-surfaces. Les temps sont relevés à l'aide du profiler, un widget disponible dans Grasshopper indiquant le temps de calcul de chaque fonction.

Contexte 1			Contexte 2			Contexte 3	
nbre	temps 1		nbre	temps 2		nbre	temps 3
35	0.6		35	0.6		35	0.7
140	2.2		140	2.6		140	2.4
560	10.7		560	10.1		560	9.9
3500	50.4		3500	59.2		3500	58.6
14000	228		14000	252		14000	240

Fig. 35: Récolte des temps de simulation en seconde en fonction du nombre de sous-surfaces pour chaque type de façade.

Le type de façade semble avoir très peu d'influence sur le résultat (voir fig. 35). L'élément prédominant est clairement le nombre de sous-surfaces à analyser. Cela veut dire que pour une surface plus importante, le niveau de résolution sera plus bas pour une même durée ou un même nombre de simulations. Il faut également prendre en compte le fait qu'il n'est pas courant de faire une seule simulation à la fois.

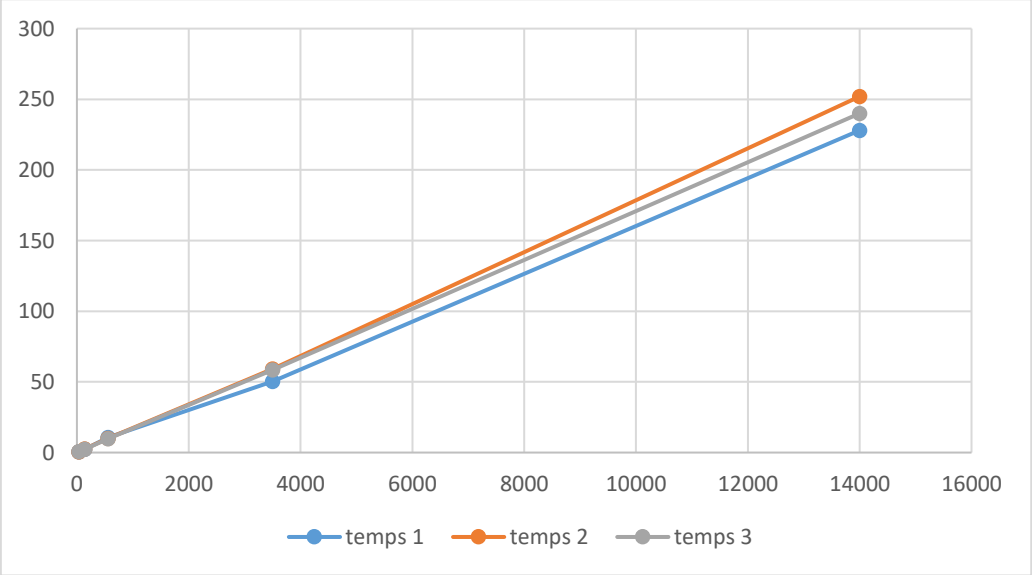


Fig. 36: Comparaison des données récoltées pour chaque type de façade.

Dans ce cas-ci, toutes les ressources du processeur sont dédiées à la simulation. Par exemple, pour le volume de type B, si seul un des 8 cœurs logiques du processeur est utilisé, le temps de simulation pour 3500 sous-surfaces est de 100 secondes au lieu de 58 secondes. L'outil nous permet de choisir entre l'usage multi-cœur et l'utilisation d'un cœur logique à la fois. Par contre si 8 simulations identiques ont lieu en même temps, une pour chaque cœur logique, le temps total de la simulation est de 694 secondes. L'utilisation du processeur fluctue alors entre 17% et aux environs de 40%, l'utilisation n'est donc pas optimale (voir fig. 37). En repassant sur un usage multi-cœur pour chaque simulation on obtient 421 secondes. L'utilisation du processeur est alors de 100% avec quelques fluctuations (voir fig. 38). Rien que ce paramètre représente un gain de près de 40%. Ce genre de choix qui de prime abords ne touche en rien l'architecture peut donc influencer de manière significative le développement du projet.

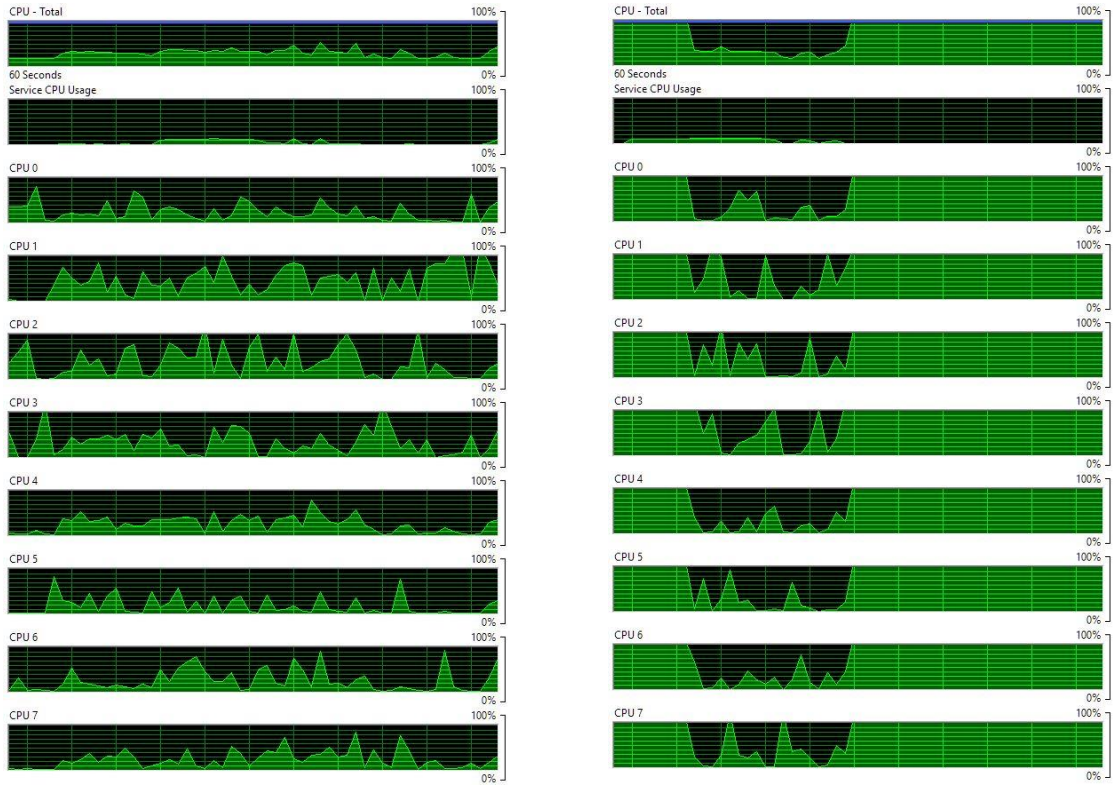


Fig. 37 & 38 : Usage du processeur sans et avec l'option de calcul multi-cœur activé. Le pourcentage d'usage est monitoré à partir du gestionnaire de tâche de Windows.

Choix de l'algorithme

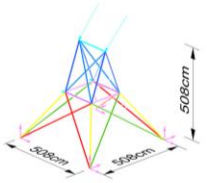
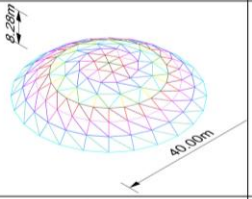
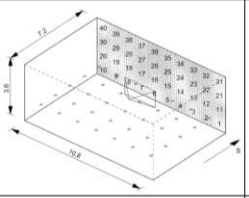
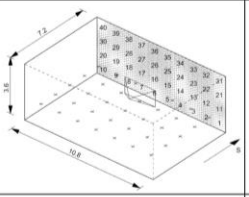
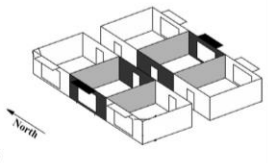
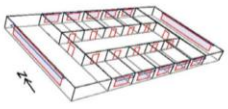
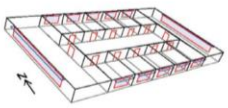
	Définition du problème		Optimisation	Galapagos		Silvereye	Goat	Opossum
			#simula-tions	GA	SA	PSO	DIRECT	RBFOpt
STRUCTURE (Simulation: Karamba)	Problème 1 minimisation de la masse totale Paramètres: • Sections de poutres		200	292kg	/	/	241kg	**237kg**
	Problème 2 minimisation de la masse totale Paramètres: • Sections de poutres		200	65kg	/	/	**26kg**	45kg
LUMIERE (Simulation: DIVA)	Problème 1 maximisation de la lumière naturelle Paramètres: • Niveau d'ouverture de la facade		200	70%	40%	600%	**11%**	40%
	Problème 2 Minimisation de l'éblouissement et maximisation de la lumière naturelle Paramètres: • Niveau d'ouverture de la facade		200	24%	27%	23%	36%	**20%**
ENERGIE (Simulation: EnergyPlus)	Problème 1 Paramètres: • Orientation • Largeur des fenêtres • Transmittance du pare-soleil		200	135.0 kWh/m²	**133.0 kWh/m²**	**133.0 kWh/m²**	**133.0 kWh/m²**	**133.0 kWh/m²**
			500	134.5 kWh/m²	**133.0 kWh/m²**	**133.0 kWh/m²**	**133.0 kWh/m²**	**133.0 kWh/m²**
	Problème 2 Paramètres: • Hauteur des fenêtres • Largeur des fenêtres • Profondeur des pares-soleil • Mise en place des pares-soleil • Mise en route de l'HVAC		200	146.0 kWh/m²	144.0 kWh/m²	143.0 kWh/m²	143.0 kWh/m²	**142.0 kWh/m²**
			500	144.0 kWh/m²	**141.6 kWh/m²**	142.5 kWh/m²	**141.6 kWh/m²**	141.8 kWh/m²
	Problème 3 Paramètres: = problème d'énergie 2 mais avec des chiffres significatifs		200	140.0 kWh/m²	**136.0 kWh/m²**	137.0 kWh/m²	137.0 kWh/m²	136.5 kWh/m²
			500	139.0 kWh/m²	**136.0 kWh/m²**	136.5 kWh/m²	136.2 kWh/m²	136.2 kWh/m²

Fig. 39 : Tableau récaitulatif des travaux menés par T. Wortmann.

Un tableau récapitulatif (voir fig.39) a été mis au point sur base d'une série de travaux menés par Mr Wortmann qui travaille au développement d'Opossum. Ce tableau compare les performances obtenues, par algorithme, pour un nombre de simulations, et donc un temps donné.

La première constatation qui ressort de cette étude est que les algorithmes génétiques ne trouvent jamais la meilleure solution. Différentes études menées par Nguyen et Evins démontrent la dominance des algorithmes génétiques dans la construction (voir fig.40). Ils sont de loin les plus utilisés en optimisation en architecture. Les résultats de leurs études proviennent d'une enquête menée au sein de bureaux d'ingénieurs qui travaillent en général en utilisant l'optimisation multi-objectif. Grâce aux données récoltées dans le tableau (voir fig.39) nous tentons de démontrer dans une certaine mesure que certains problèmes peuvent être résolus de manière plus rapide à l'aide d'autres algorithmes.

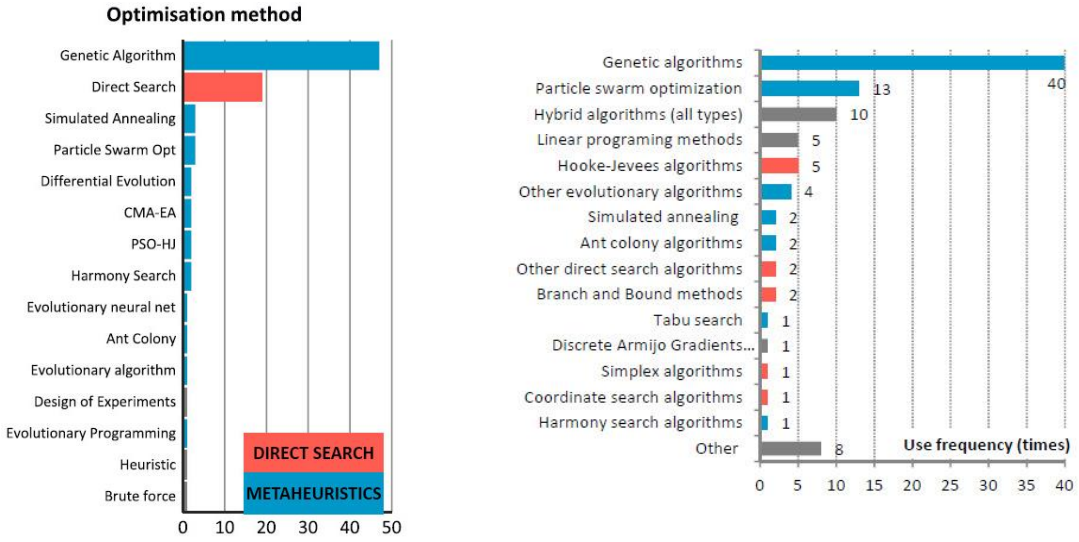


Figure 40 : Fréquence d'utilisation des algorithmes dans l'optimisation du bâtiment, [de gauche à droite Nguyen, 2014, Evins, 2013]

L'ensemble des données récoltées ont permis de dresser une classification selon 2 variables : le temps ou le nombre de simulation, et la quantité de paramètres (voir fig. 41).

#simulations/ #paramètres	Peu de paramètres <32	Beaucoup de paramètres <64*
Peu de temps de temps à disposition <200 simulations	-DIRECT -RBFOPT	-RBFOpt
Beaucoup de temps à disposition	-DIRECT -PSO -SA	-PSO -SA

Fig.41 : Classification des optimisations selon deux paramètres (Wortmann & Cichocka, IASS 2017 Masterclass, 2017)

Il apparaît que les méthodes métaheuristique (PSO, SA) seraient à privilégier pour les plus grands projets. C'est ce qui a probablement contribué à leur notoriété. Aussi comme il a été dit précédemment, les Algorithmes Génétiques ont l'avantage de pouvoir venir à bout de n'importe quel problème, notamment lorsque l'on se retrouve face à des fonctions discontinues, ce qui est visiblement le cas en architecture [Wetter & Wright, 2004]. Avec l'implémentation de nouvelles approches (RBFOpt, DIRECT), l'optimisation devient de manière générale, également intéressante sur des projets de petite échelle car ils pallient au manque de temps au prix d'une exactitude de toute façon subjective.

Dans notre tableau, les problèmes d'énergie, de lumière et de structure étant tous traités de manière indépendante, sont des problèmes idéaux pour les solveurs à objectif simple. Dans une approche numérique du projet globale où plusieurs objectifs sont pris en compte, l'approche par les algorithmes génétiques reste plus adéquate. D'autres méthodes en multi-objectif, notamment basées sur modèle [Model-Based Multi-Objective Optimization: Taxonomy, Multi-Point Proposal, Toolbox and Benchmark] existent mais ne sont malheureusement pas encore implémentées à Grasshopper. Elles ne nous permettent donc pas de pousser plus loin la comparaison.

Fasoulaki [2007] et d'autres auteurs [Wortmann, Wetter] résument très bien le problème de la quantité d'information ainsi que de la complexité liée au projet d'architecture. Les algorithmes génétiques offrent une solution générique qui fonctionne pour tous les problèmes mais la sélection d'une méthode d'optimisation devrait se baser sur les besoins et la nature du problème, non pas sur la popularité de certaines méthodes.

Comparaison des différents algorithmes

Le but est de comparer l'ensemble des algorithmes décrits à la théorie à travers un cas pratique simple présentant 3 objectifs à atteindre. Les 2 premiers objectifs sont en opposition directe et le 3ème objectif est choisi de sorte à créer des discontinuités dans le paysage de solution.

2 optimisations seront calculées, la première confrontera les 2 premiers objectifs et la deuxième prendra les 3 objectifs en compte.

Cela devrait permettre la mise en évidence de l'efficacité des algorithmes dans un contexte de développement de projet où l'architecte a peu de temps disponible ainsi que la faiblesse de la méthode naïve décrite en théorie.

Présentation du modèle

Le modèle d'étude est simplifié de sorte à faciliter la lecture de la définition et à raccourcir les temps de simulation. Il consiste en une extrusion cylindrique à toiture en pente sur le toit d'un immeuble (voir fig. 42). Les différents paramètres sont le rayon de base du cylindre, la position du bâtiment sur la toiture de l'immeuble et la position des 3 points qui contrôlent la toiture. Les positions des points de toitures sont données par une valeur de hauteur ainsi qu'une valeur de rotation. Le rayon de base est contraint arbitrairement à un minimum de 5 mètres et à un maximum correspondant à la distance minimum entre le centre du cercle de base et la bordure du toit de l'immeuble afin que le projet ne se retrouve jamais en porte-à-faux. La position du centre est contrainte à une surface de toiture réduite définie manuellement. La hauteur des points est contrainte de manière arbitraire entre 220cm et 500cm et à un angle de rotation ayant pour centre de rotation le centre du cercle de base compris entre -120° et 120° étant donné que les 3 points sont le résultat d'une division en 3 segments égaux du cercle de base.

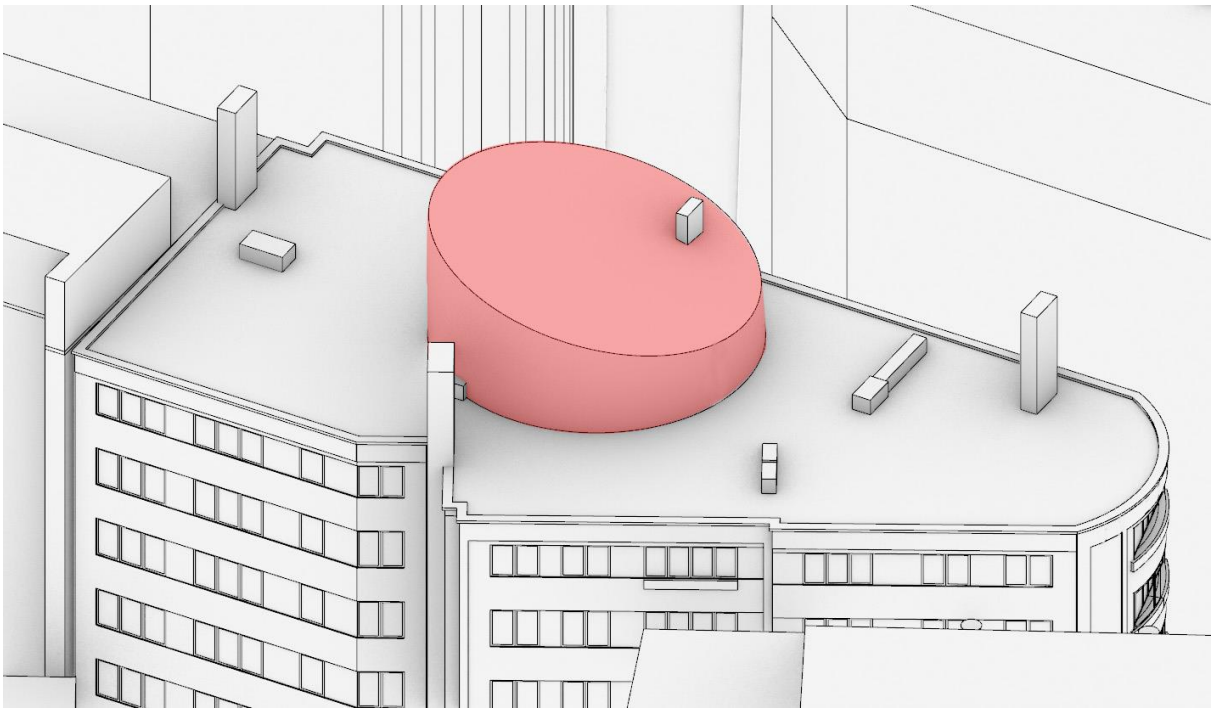


Fig. 42 : Modèle d'étude (en rouge) et son contexte

L'optimisation de ces paramètres se basera sur les résultats de 3 simulations. Le gain d'apport solaire lorsque la température est inférieure à 12°C (voir fig. 43), le surplus d'énergie dû à l'exposition lorsque la température est supérieure à 18°C (voir fig. 44) et l'accès visuel au paysage (voir fig. 45). Pour des raisons de facilité et simplicité, seule la façade est considérée et aucun percement n'est pris en compte. Comme pour la simulation précédente, la surface étudiée est divisée en une série de sous-surfaces pour plus de résolution. La simulation se base sur les données météorologiques de l'observatoire de Uccle à Bruxelles et sont disponibles sur https://www.energyplus.net/.

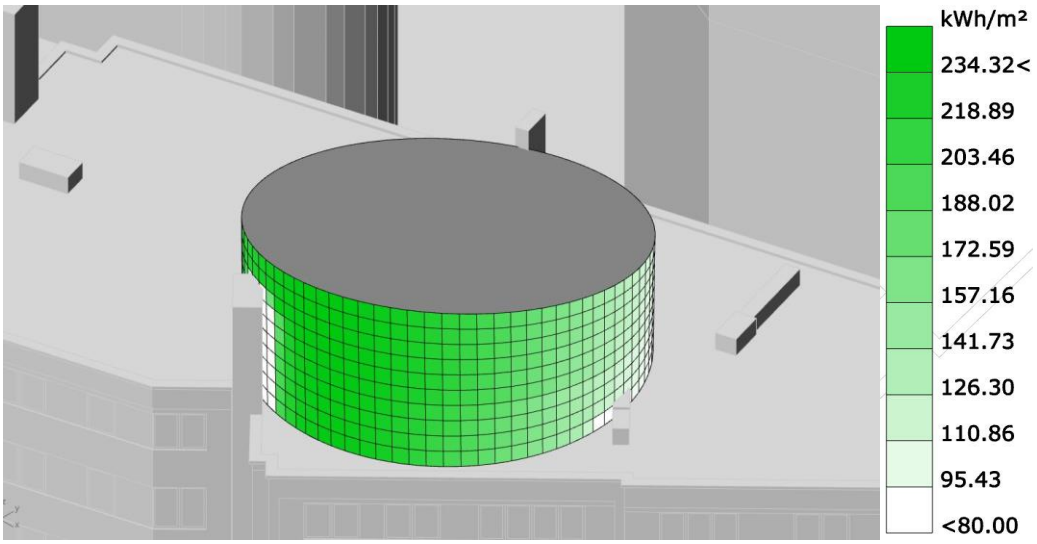


Fig. 43 : Apports énergétiques surfaciques solaires en KWh/m² lorsque la température est inférieure à 12°C

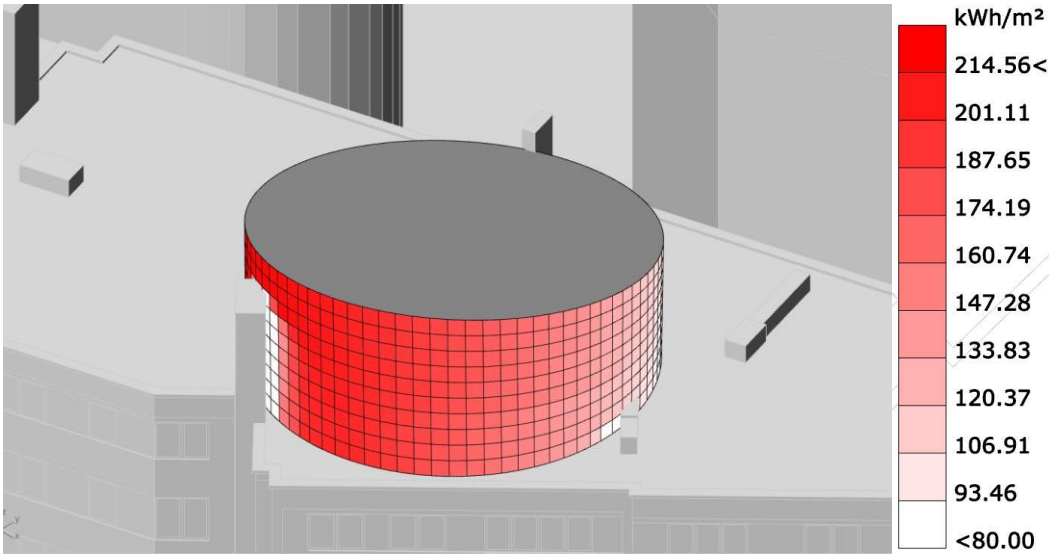


Fig. 44 : Apports énergétiques surfaciques solaires en KWh/m² lorsque la température est supérieure à 18°C

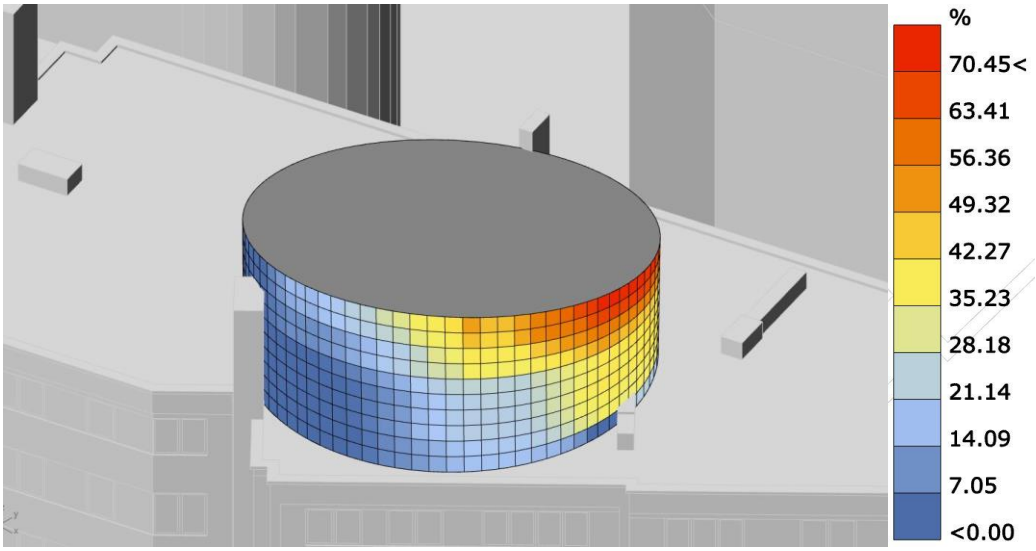


Fig. 45 : Accès visuel au paysage vue (Vue SE)

Dans le but de quantifier l'accès visuel au paysage, un contexte urbain est introduit autour du modèle et le paysage est modélisé sous forme d'un ruban cylindrique autour de la ville. Ce ruban cylindrique est décomposé en 4 parties représentant les NE, NO, SE, SO. Ces 4 morceaux sont ensuite décomposés en une série de points en fonction de leurs intérêts paysager (voir fig. 46). La partie NO par exemple est jugée comme étant la plus intéressante visuellement, elle est donc décomposée en plus de points. Un point duquel il est possible de tracer une ligne avec tous les points du paysage sans qu'il n'y ai de collision avec le contexte présente 100% d'accès visuel.

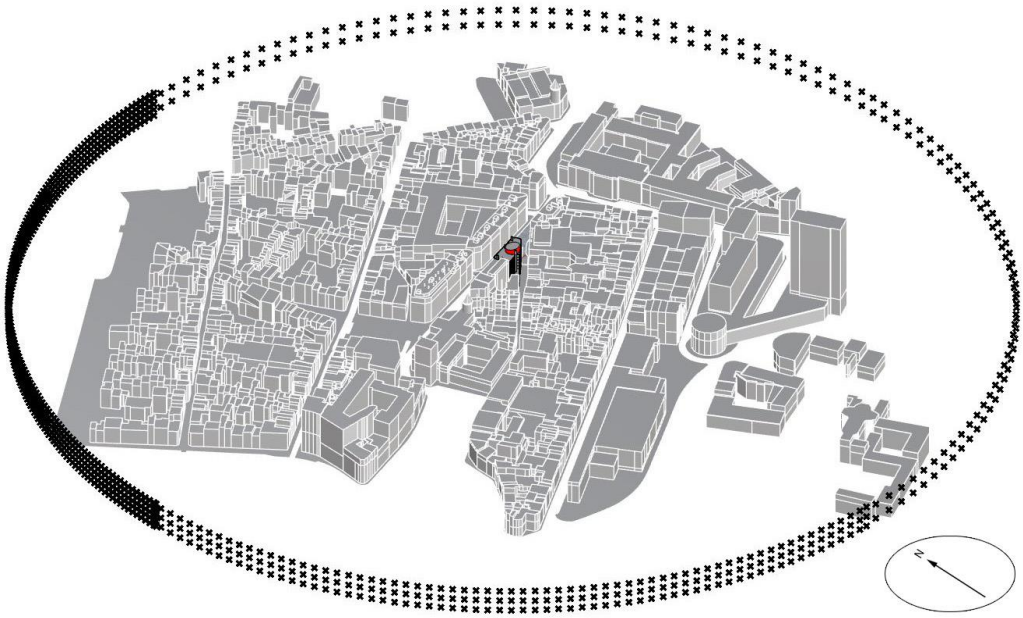


Fig. 46 : contexte urbain et décomposition du paysage

Mise en place de la fonction pénalisante

La fonction pénalisante mettant en relation les 2 objectifs d'exposition au soleil est simple à mettre en place. Chris Mackey [2016] fait la relation dans l'une des définitions qu'il partage par une simple soustraction de valeurs dont il faut maximiser le résultat.

F(v) = P1 - P2 v étant l'ensemble des variables en un point du paysage de solution

Ce résultat est directement visible dans la fenêtre de visualisation de Rhinocéros (voir fig. 47)

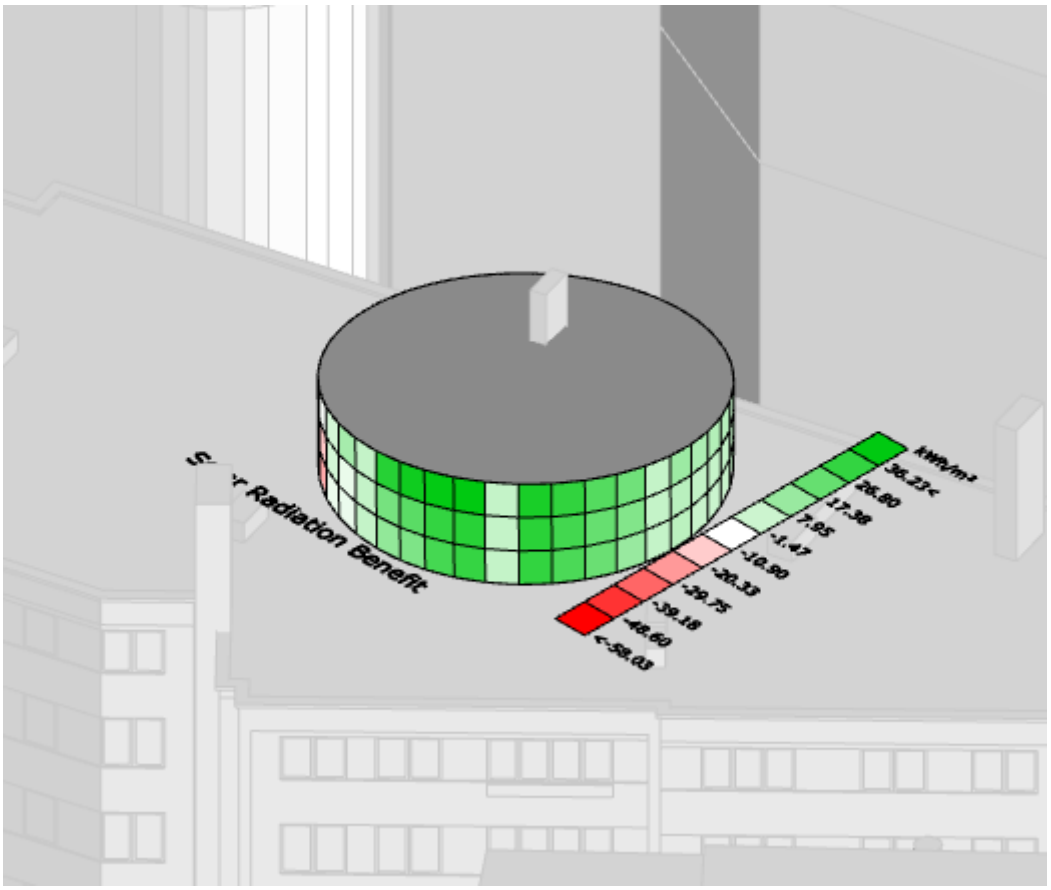


Fig. 47 : Retour visuel du résultat de la fonction pénalisante

Afin de comparer tous les algorithmes face à cette fonction pénalisante, la durée fût limitée à 20 minutes. Tous les solveurs ne possédant pas un plafond de temps paramétrable, ont dû être stoppé de manière manuelle, il faut donc prendre en compte quelque secondes de marge d'erreur.

Les données sont récoltées à l'aide du plugin « Colibri » qui permet l'écriture des paramètres et des performances correspondantes après chaque simulation. Les données sont ensuite traitées à l'aide du tableur Excel.

Pour une même durée, le nombre de simulations calculé varie en fonction du solveur. Le tableau qui suit prend en compte le nombre de simulation minimum calculé qui est de 27 (voir fig. 48).

Solveur	GALAPAGOS		SILVEREYE	GOAT					OPOSSUM
Algorithme	SA	GA	PSO	QCRS2	DIRECT	SUBPLEX	COBYLA	BOBYQA	RBFO
1	1096	851.8334	-999	-524.83	-621.873	-704.361	-53.1275	-568.344	-1419.54
2	886	-257.471	-1005.02	-437.478	-642.977	-723.307	54.95788	-610.354	-293.758
3	718	-1473.68	-1383.1	-389.687	-573.114	-738.107	-4.7001	-1043.28	-925.39
4	1132	-516.004	-787.537	-411.349	-572.585	-731.183	229.8867	-749.899	-740.816
5	1065	-475.259	-913.652	-357.06	-1012.14	-166.894	221.7011	-561.807	-1107.96
6	-884	-674.314	-1055.89	-455.736	-856.536	-563.19	-1854.59	-706.451	-586.291
7	-249	-661	-186.218	-400.455	-743.634	-373.721	-1848.18	-740.866	-1447.85
8	-750	-394.267	-1329.65	-425.894	-976.431	-372.153	-1384.39	-699.154	-902.582
9	-833	-833.711	-1167.65	-348.794	-1114.24	-108.118	-1378.1	-585.741	106.8347
10	-665	-765.719	-645.177	-351.241	-920.461	-129.002	-701.169	-1016.81	-1138.26
11	-327	-541.332	-402.992	-462.794	-1294.28	-203.763	-754.998	-562.072	-396.502
12	-346	-336.598	-853.089	-376.645	-990.59	-348.281	-63.3561	-754.795	-392.29
13	-672	-1209.14	-937.169	-333.51	-1032.59	-439.78	-28.9167	-613.271	-511.877
14	-834	-985.613	-251.339	-353.184	-1042.19	-412.657	-1530.74	-258.757	-105.846
15	-878	154.6752	-896.002	-105.378	-852.788	-193.411	-1577.59	-215.137	-557.317
16	-1079	-387.369	-817.431	-130.484	-567.46	-30.6426	258.0905	-112.899	-563.877
17	-954	-1285.89	52.538	-172.467	-557.045	-185.69	-1827.96	-136.315	108.5906
18	-1324	-189.051	-427.826	-117.5	-716.616	-117.066	-1902.71	-120.47	-400.856
19	-1303	302.3174	-750.776	-583.558	-460.871	-130.058	222.2754	-499.412	-422.626
20	-531	-1521	-422.759	-602.272	-338.285	-43.7643	-1872.13	-493.398	-468.855
21	-570	-1116.75	-1097.77	-930.9	-291.122	-336.202	-1879.16	-493.555	211.6761
22	-598	-346.337	-473.325	-1184.84	-660.242	-282.043	-14.8281	-488.832	-330.361
23	-586	-315.21	-863.146	-1166.73	-629.384	-192.3	218.7957	-494.766	259.6286
24	-550	-1316.76	-224.958	-1258.34	-421.033	-214.731	-1644.19	-493.793	410.2229
25	-436	-450.526	-302.159	-1348.45	-775.044	-119.039	-1896.75	-491.5	208.801
26	-424	408.5604	-97.6515	-1386.93	-430.031	-46.2241	-1920.25	-495.537	262.7113
27	-403	-459.426	143.0111	-660.079	-207.718	-46.4852	233.0528	-333.839	59.89313

Fig. 48 : Résultats de l'optimisation de la fonction pénalisante pour chaque algorithme disponible dans grasshopper. Les données surlignées correspondent à l'optimum trouvé.

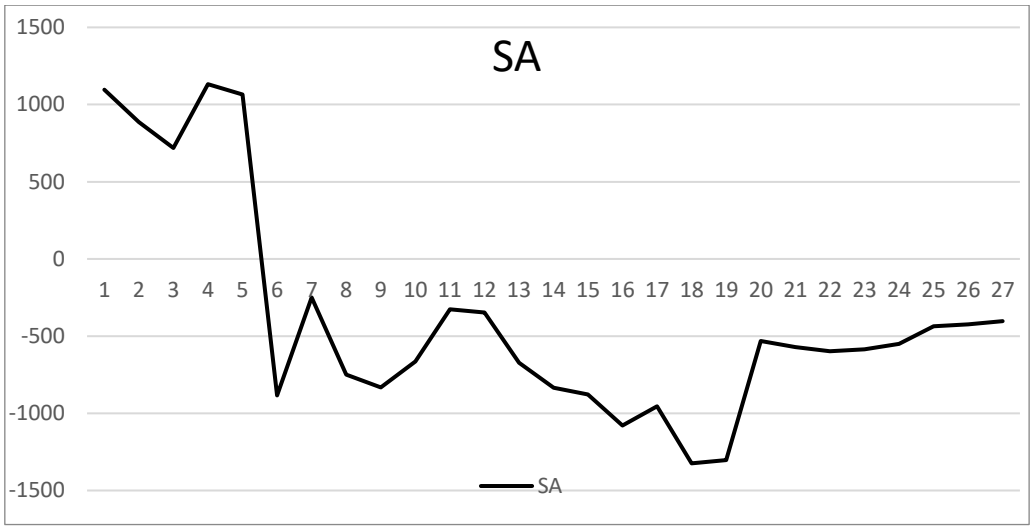


Fig. 49 : Recherche d'un optimum à l'aide de l'algorithme SA proposé par Galapagos

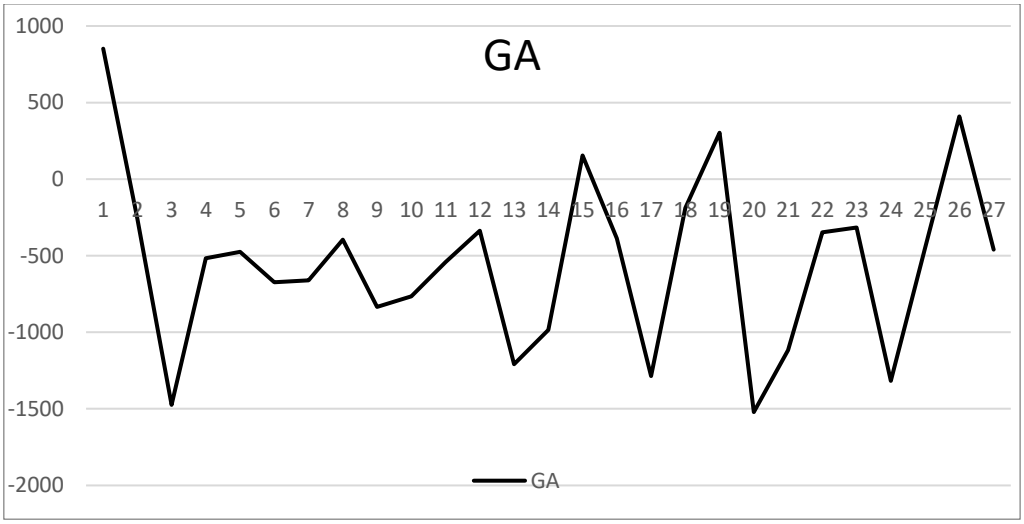


Fig. 50 : Recherche d'un optimum à l'aide de l'algorithme GA proposé par Galapagos

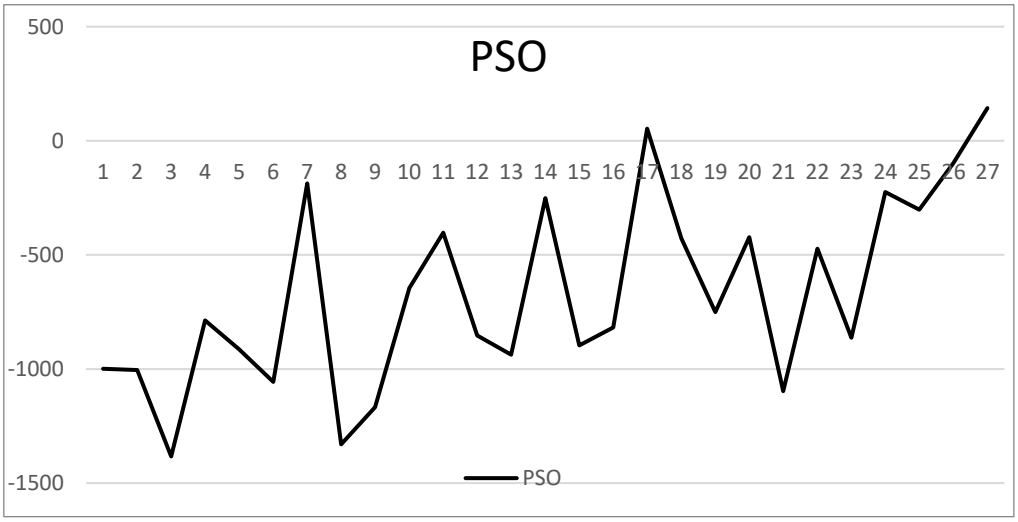


Fig. 51 : Recherche d'un optimum à l'aide de l'algorithme PSO proposé par Silvereye

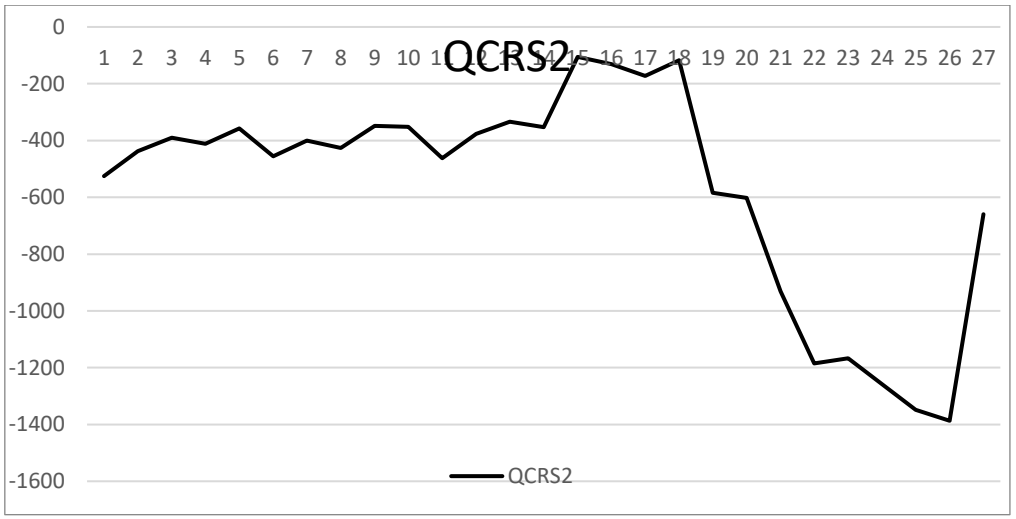


Fig. 52 : Recherche d'un optimum à l'aide de l'algorithme QCRS2 proposé par Goat

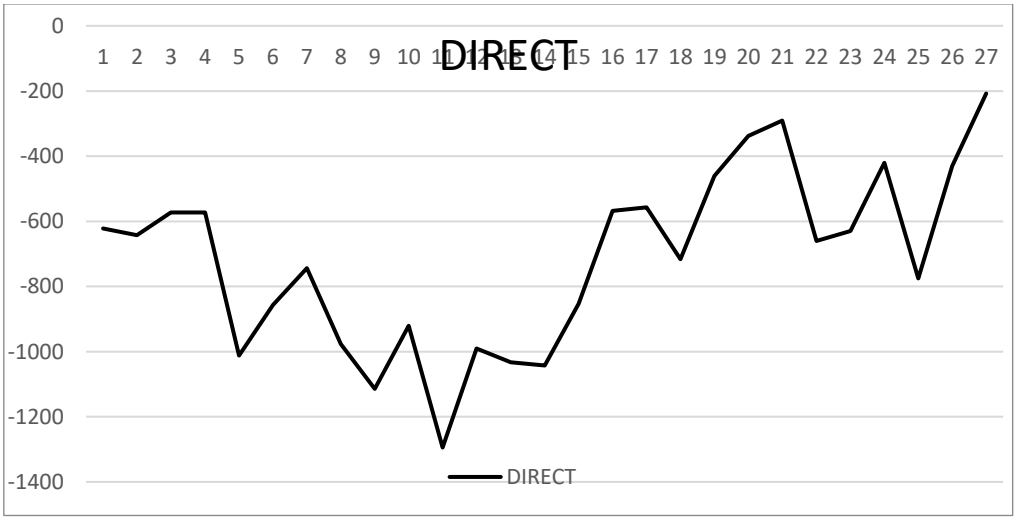


Fig. 53 : Recherche d'un optimum à l'aide de l'algorithme DIRECT proposé par Goat

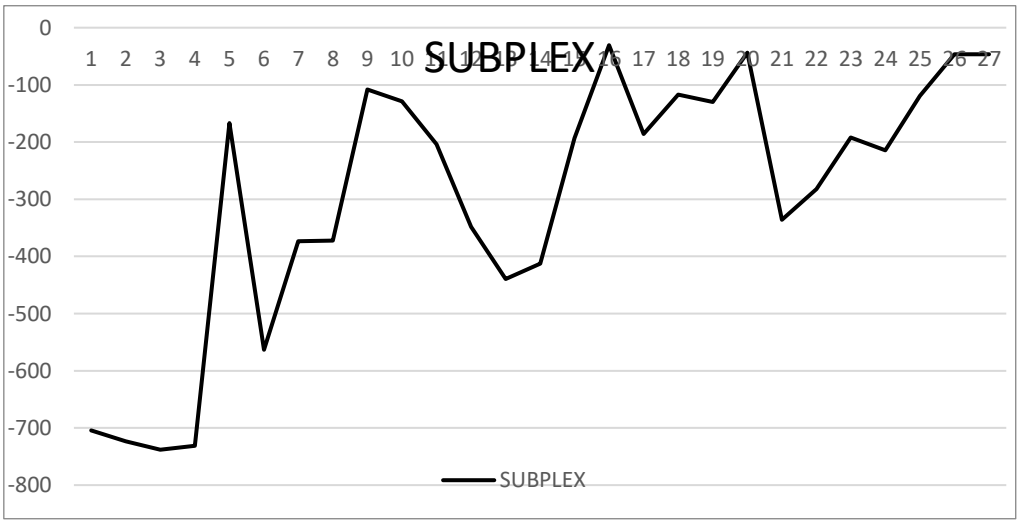


Fig. 54 : Recherche d'un optimum à l'aide de l'algorithme SUBPLEX proposé par Goat

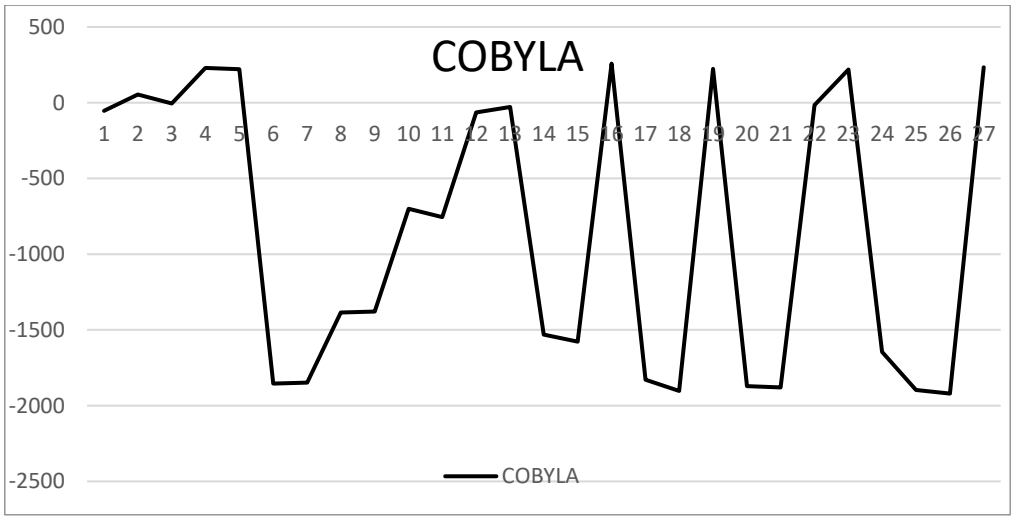


Fig. 55 : Recherche d'un optimum à l'aide de l'algorithme COBYLA proposé par Goat

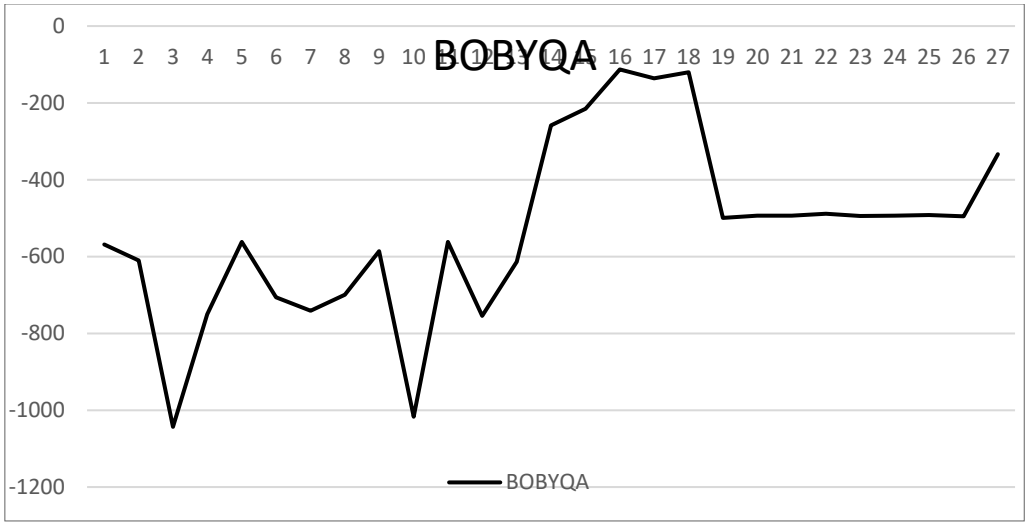


Fig. 56 : Recherche d'un optimum à l'aide de l'algorithme BOBYQA proposé par Goat

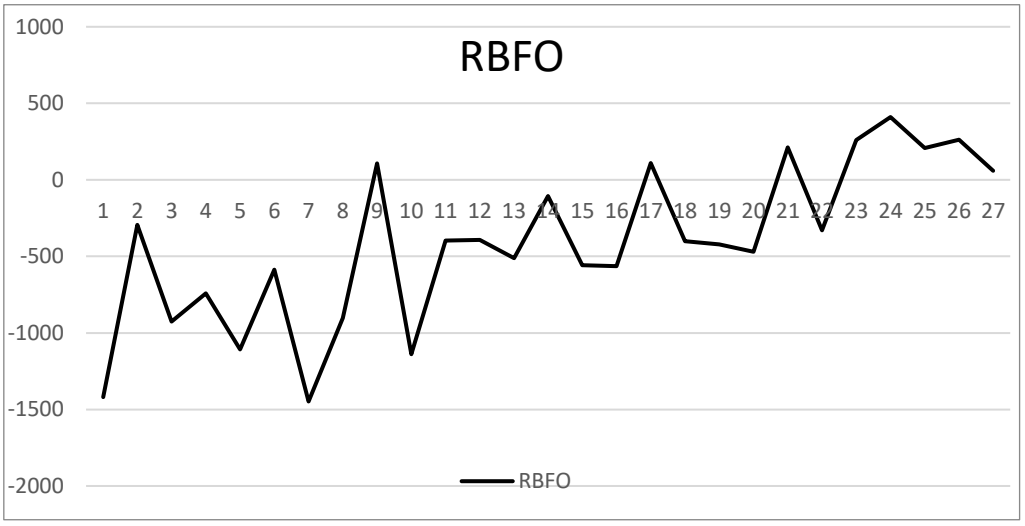


Fig. 57 : Recherche d'un optimum à l'aide de l'algorithme RBFO proposé par Opossum

Galapagos avec son algorithme, le Simulated Annealing, trouve le plus haut maximum de la fonction pénalisante établie (voir fig. 49). Ce résultat n'est pas évocateur de l'efficacité de la recherche, il est le résultat de la distribution de la population de départ sur le paysage de solutions. 27 simulations est très peu pour se rendre compte de l'efficacité d'un algorithme mais dans la mesure ou la simulation prendrait plus de temps, il est pertinent de vouloir savoir vers quel algorithme se diriger. Dans le cas du SA, 27 simulations ne suffisent pas à déterminer de bons candidats.

Il est néanmoins intéressant de noter que si l'optimisation avait continué, Galapagos aurait obtenu une courbe intéressante (voir fig. 58).

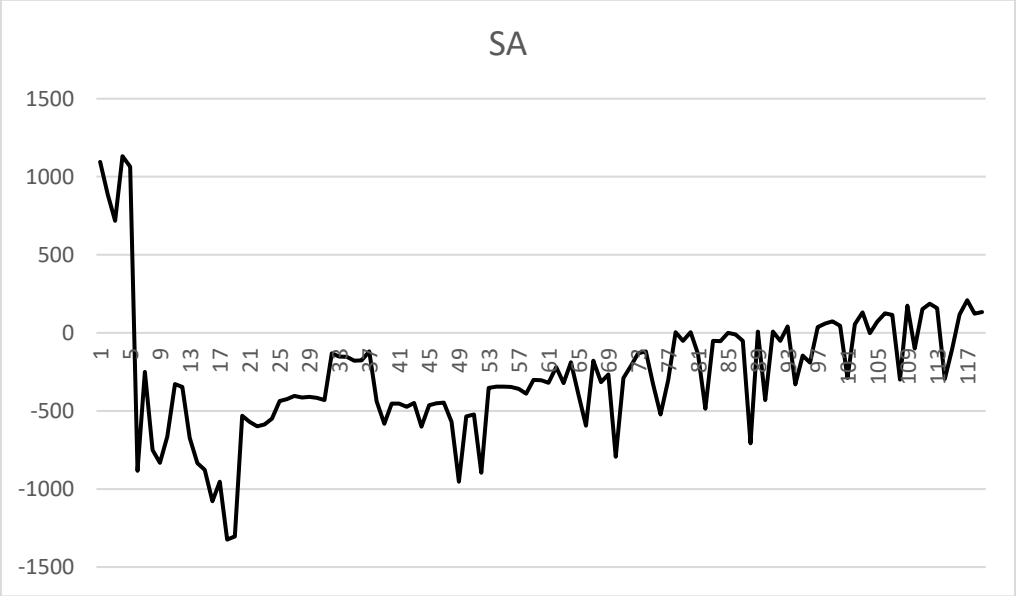


Fig. 58 : Recherche d'un optimum à l'aide de l'algorithme SA proposé par Galapagos sur une durée plus étendue

Les Algorithmes Génétiques présentent le même comportement au départ mais, contrairement au SA, se débrouillent très bien sur 27 simulations. Avec un si petit nombre de simulations, il est nécessaire de régler la taille de population. Dans cet exemple, elle fût réglée à 10 (voir fig. 59), ce qui signifie qu'en s'arrêtant à 27 itérations, l'algorithmes n'était arrivé qu'à la 3ème génération. La génération suivante proposait d'ailleurs de biens meilleurs résultats avec 587 et 598kWh/m². Si la population de départ avait été de 100 comme elle est entré par défaut, la recherche aurait stoppé à 27 individus de première génération, c'est à dire placés de manière aléatoire sur le paysage. Cela signifie qu'une certaine compréhension des mécanismes est nécessaire à l'usage de ces outils.

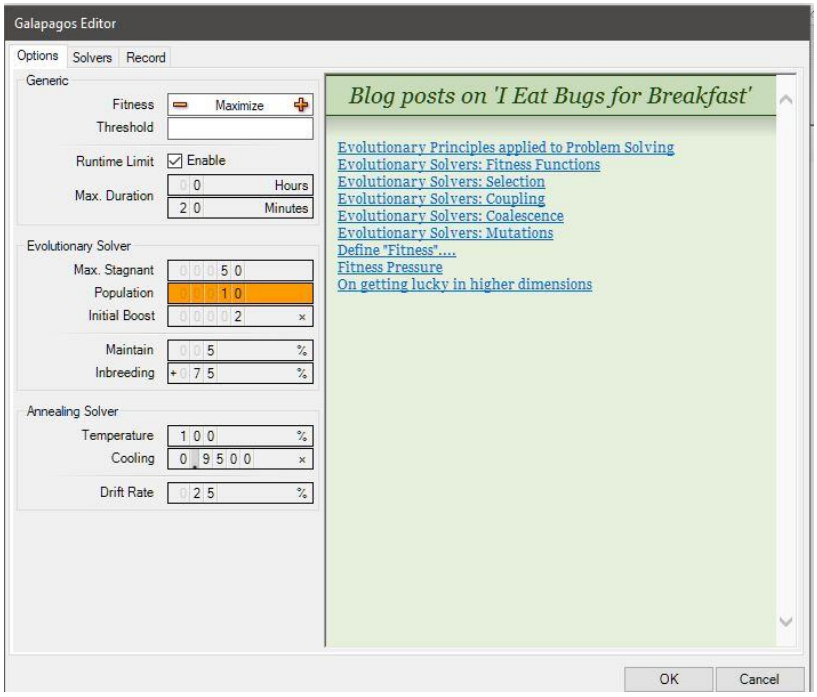


Fig. 59 : Interface du solveur Galapagos. La population de départ (en orange) a dû être modifiée.

La courbe ne montre néanmoins pas de signe distinctif d'amélioration sur les 20 minutes disponibles. A court terme, les résultats des algorithmes génétiques sont donc essentiellement aléatoires. Mais en poussant la recherche à 10 générations, la courbe s'améliore de manière progressive (voir fig. 60).

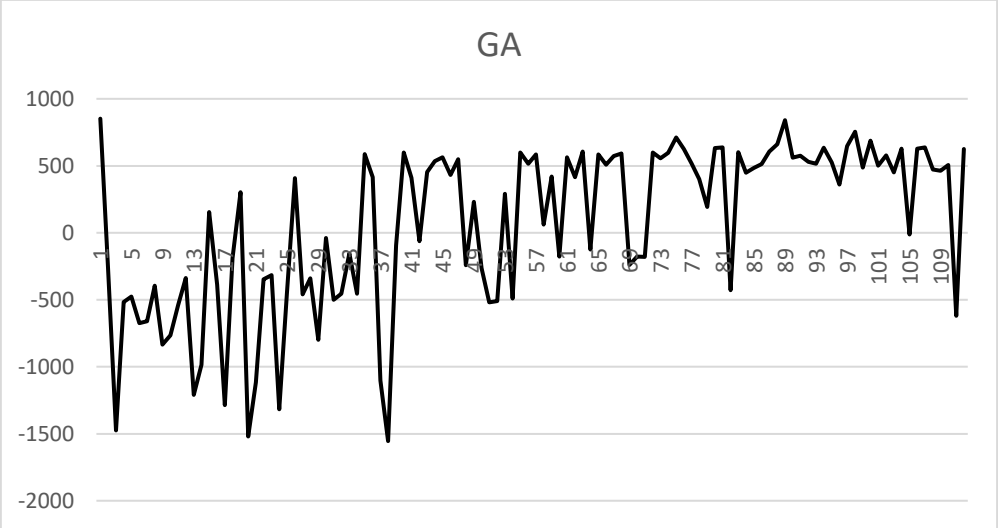


Fig. 60 : Recherche d'un optimum à l'aide de l'algorithme GA proposé par Galapagos sur une durée plus étendue

Le « Particle Swarm Optimization » présente déjà une progression générale continue sur le court terme (voir fig. 51). A nouveau ici, le nombre de particules a dû être réduit à 10 en vue d'obtenir une optimisation plus efficace (voir fig. 61).

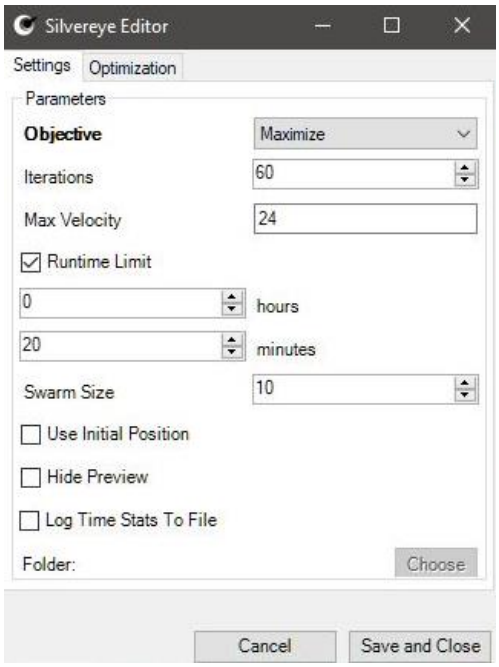


Fig. 61 : Interface du solveur Silvereye. Le « Swarm Size » est modifié à 10.

Les résultats obtenus par le solveur GOAT sont assez mitigés. Seul COBYLA parvient à obtenir un résultat positif mais ne montre aucun signe d'amélioration sur le long terme. BOBYQA et COBYLA ne parviennent donc pas à extrapoler de fonction sur base de ces 27 simulations. A long terme COBYLA ne parvient pas à se stabiliser, ce qui porte à penser que même pour une simulation aussi simple que celle-ci l'approche linéaire n'est pas une bonne solution.

Il faut également noter que les résultats présentés ne sont les résultats que d'une seule optimisation. En dehors de la recherche directe, pour une même optimisation, les résultats ne seront pas les même. Ceci est visible sur la figure 62 ou une seconde optimisation à l'aide de COBYLA fut mise en place.

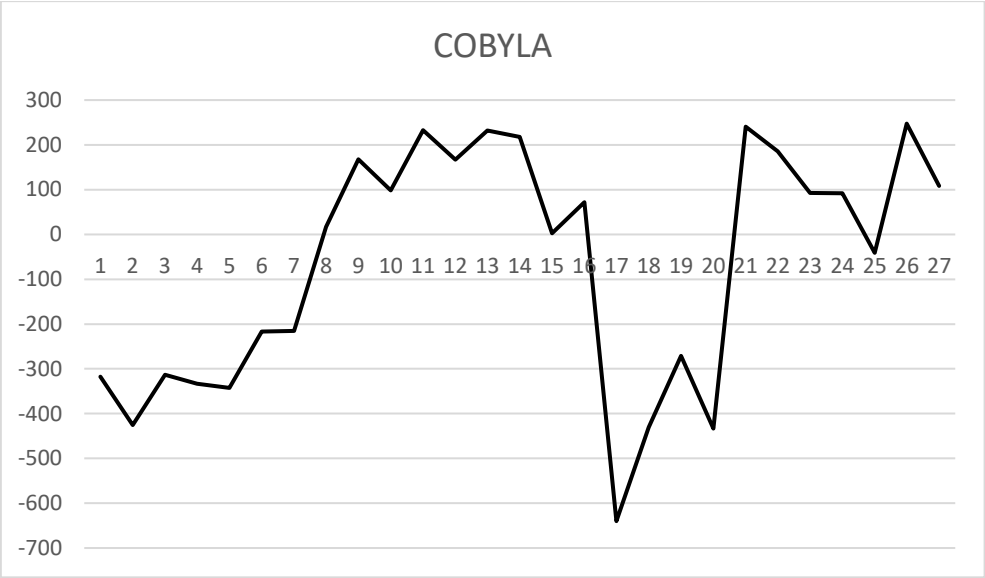


Fig. 62 : Recherche d'un optimum à l'aide de l'algorithme COBYLA proposé par Goat

A long terme cette fois-ci COBYLA présente une légère amélioration globale mais l'analyse des résultats (voir fig. 63) montre qu'il ne fait que s'enfermer dans un maximum local, en effet les résultats montrent très peu de variations sur les 24 dernières itérations, ce qui n'est pas idéal en terme d'exploration. Ces résultats sont directement comparables aux résultats obtenus par Galapagos à long terme qui présente un panel de solution plus intéressant (voir fig. 64).

RBFO semble être le plus prometteur (voir fig. 57). Non seulement il trouve la meilleure solution sur les 27 simulations mais en plus il présente la progression la plus marquée ce qui porte à croire que même au-delà des 27 simulations il restera efficace.

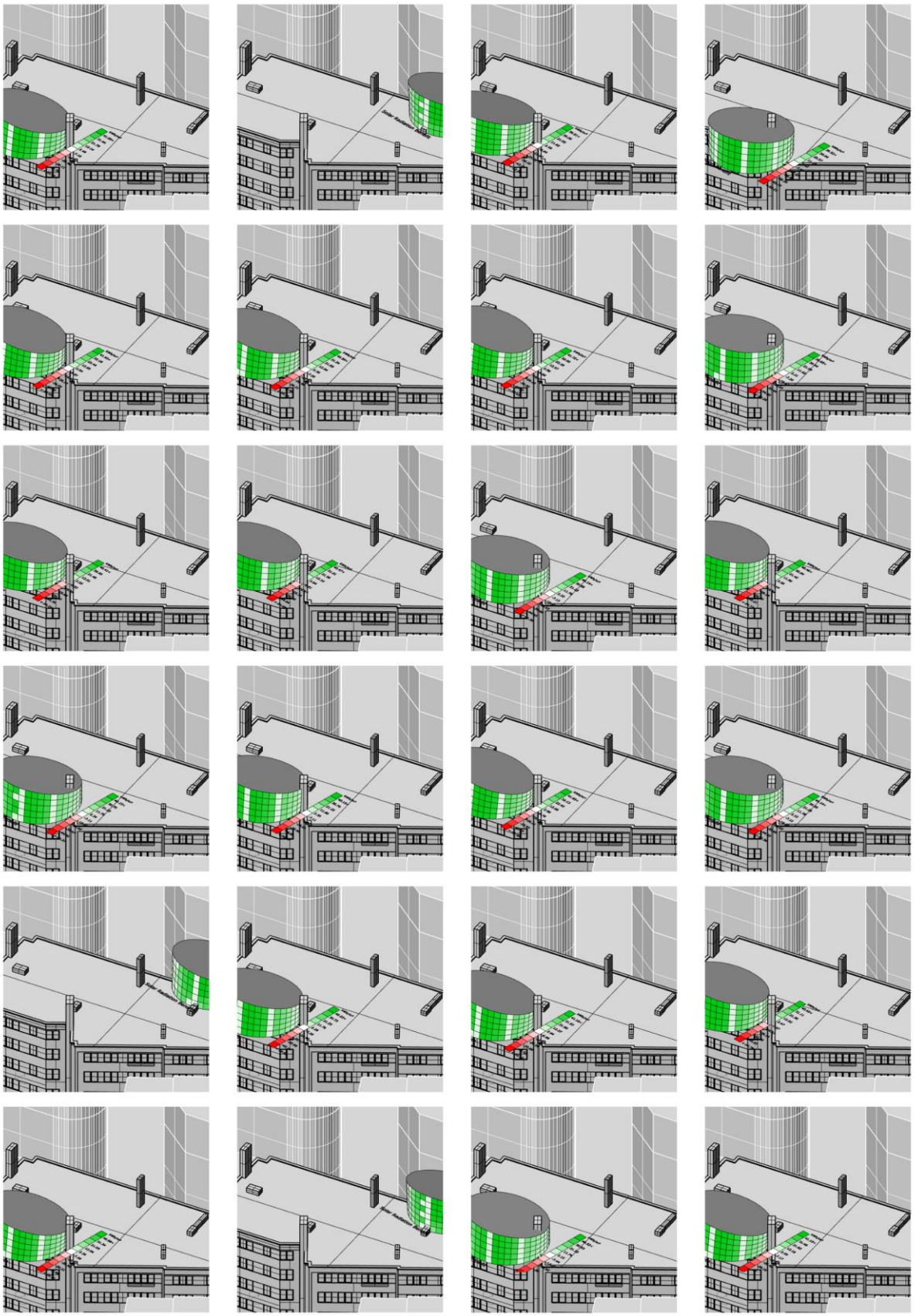


Fig. 63 : Les 24 derniers résultats à long terme obtenus par COBYLA présente une série de solutions quasi identiques, ce qui porte à penser que l'algorithme est coincé dans un maximum local.

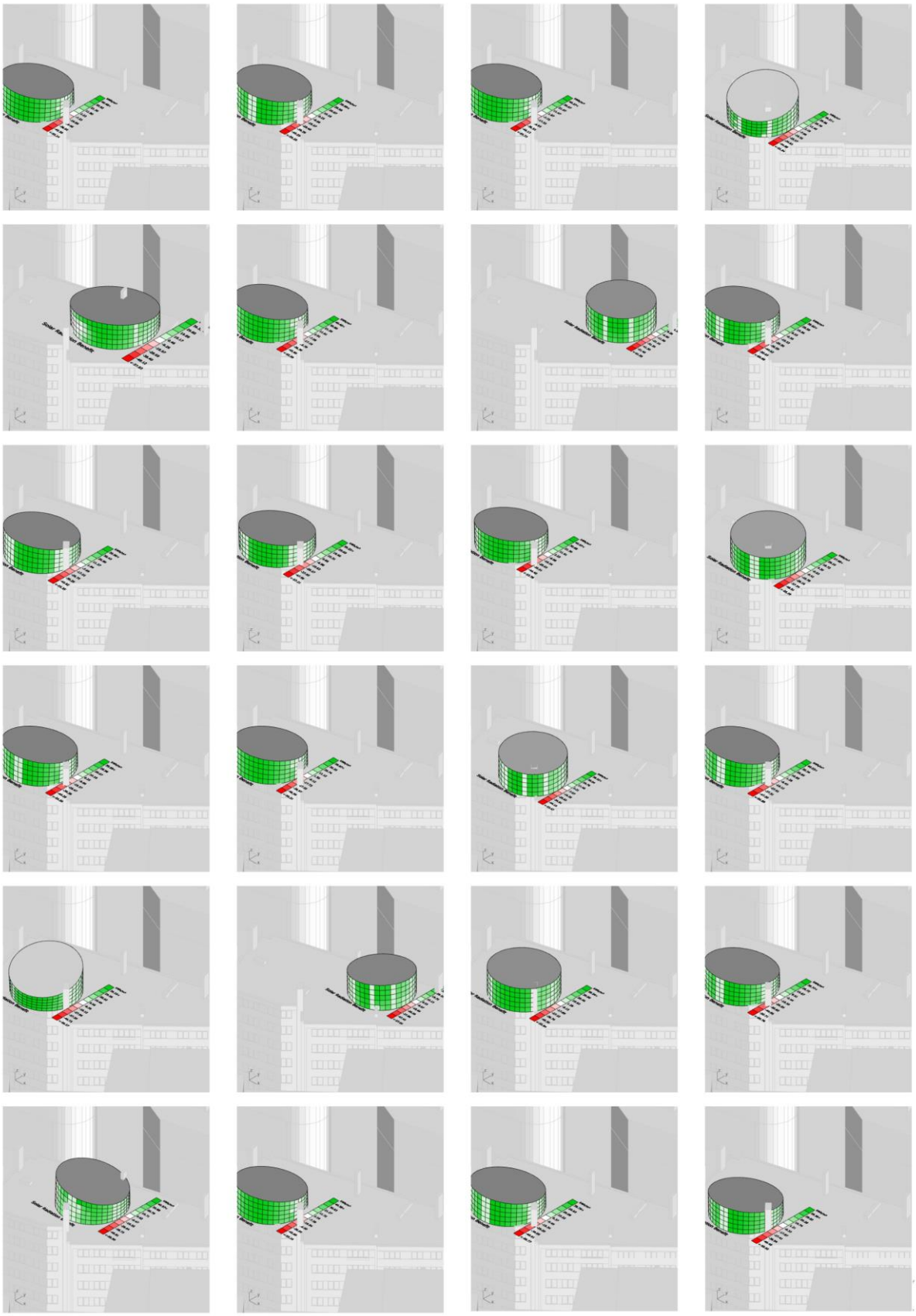


Fig. 64 : Les 24 derniers résultats à long terme obtenus par Galapagos et son GA présente un potentiel d'exploration plus intéressant que COBYLA, en effet il présente un panel diversifié de solutions.

Octopus d'autre part, comme il a été vu, optimise chaque objectif de manière indépendante et propose à travers son interface une série de compromis qu'il expose le long du Front Pareto. Il a l'avantage d'offrir un retour visuel pour chaque solution trouvée ne nécessitant pas de l'architecte de procéder à l'enregistrement de chaque valeur pour chaque simulation et au traitement préalable des données dans le but de retrouver les bonnes solutions. Cliquer sur un point de la courbe charge automatiquement les paramètres encodés pour ce point dans le modèle, de plus il permet l'accès aux solutions qui se rapprochent du Pareto favorisant d'autant plus l'exploration. La position sur le front Pareto nous donne également une information sur l'objectif favorisé (voir fig. 65)

Il est intéressant de remarquer que là où tous les solveurs réalisent environ 30 simulations en 20 minutes, Octopus en réalise presque 90. Etant donné que la forme sans son contexte n'a ici pas beaucoup d'intérêt le rendu graphique directement sur le graph a été désactivé afin d'économiser un peu de puissance de calcul.

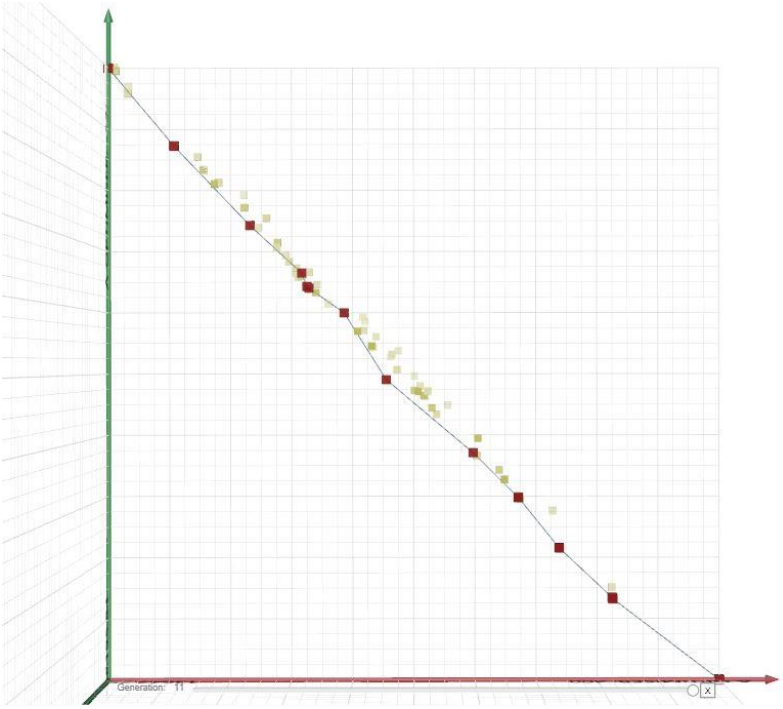


Fig. 65 : Représentation du front Pareto et des meilleures solutions trouvées représentées par les points rouges. Les points jaunes représentent les valeurs proches du Pareto. Les axes rouges et verts représentent les résultats obtenus pour chacun des 2 objectifs.

Le tableau qui suit met en relation les optimums trouvés pour la fonction pénalisante et les valeurs des objectifs pris séparément (voir fig. 66). Etant donné que la fonction pénalisante est mise en place de façon arbitraire, il est important de garder un certain recul par rapport aux résultats et l'analyse des données y joue un rôle important. Par exemple, l'optimum issu de notre fonction pénalisante est de 1132kWh/m² sur 27 simulation hors il faut considérer que cette solution présente également un très grand apport énergétique solaire lorsque la température extérieure est déjà supérieure à 18° ce qui implique peut-être une consommation d'énergie en terme de refroidissement. C'est également pour cela que l'optimisation en architecture est très complexe, énormément d'éléments, liés ou non, sont à prendre en compte.

	SA	GA	PSO	QCRS2	DIRECT	SUBPLEX	COBYLA	BOBYQA	RBFO	HYPE
Valeur Max	1132	851.8334	143.0111	-105.378	-207.718	-30.6426	258.0905	-112.899	410.2229	35.644
Objectif 1 (Hiver)	25788	20393	13029.36	13446	23116	18479	26448	12802	12280	16512
Objectif 2 (Eté)	24656	19541	12886.35	13551	23324	18409	26225	12915	11870	16476

Fig. 66 : Tableau récapitulatif des valeurs obtenues pour la fonction pénalisante et les valeurs correspondantes pour chaque objectif pris séparément.

En revanche que ce passe-t-il lorsque l'on introduit un paramètre en grande partie indépendant du reste comme la qualité de la vue. En effet, les facteurs climatiques qui influencent les 2 premiers objectifs n'entrent pas en jeu dans le calcul d'accès visuel et inversement, l'attribution de la qualité visuelle du paysage n'influe en rien les calculs d'apports énergétique. Le seul élément influant commun est le contexte.

Résultats de la seconde optimisation.

Mise en place de la fonction pénalisante

A nouveau il faut constituer la fonction pénalisante. En terme de vue, un minimum est jugé indispensable, une allure de type racine est donc privilégiée. L'accès visuel possède une valeur maximum de 100, cela veut dire que la valeur maximale atteignable au sein de la fonction est de 10. Arbitrairement, il est décidé que l'accès visuel est légèrement moins important que les gains et surplus d'énergie, et il est imposé un facteur 50. Pour un accès de 50% au paysage, on

aura donc une valeur de 50*√(50) = 353.55. L'équation pénalisante se présente alors sous la forme :

$$F(v) = (P1 - P2) + 50* \sqrt{(P3)}$$

Il est important de noter que le facteur se base sur les résultats précédents. Sans ces derniers, il aurait été impossible de déterminer un facteur convenable.

Solveur	GALAPAGOS		SILVEREYE	GOAT					OPOSSUM
Algorithme	SA	GA	PSO	QCRS2	DIRECT	SUBPLEX	COBYLA	BOBYQA	RBFO
1	-552.886	-69.4798	-675.924	-380.521	-576.246	-891.78	-43.9637	85.45253	-670.122
2	138.4746	-185.966	37.97504	-421.847	-412.127	-903.214	-65.5061	-16.4607	-531.426
3	177.1434	-341.016	-646.55	-333.397	-304.886	-969.936	209.2851	89.52729	-509.391
4	80.60784	-348.757	-381.186	-326.832	-552.167	-1025.82	172.0771	72.48675	-333.394
5	-176.029	-1155.54	-419.792	-160.515	-669.83	455.4385	209.2851	70.76886	-1073.84
6	-260.043	-266.043	-1711.99	-206.989	-520.461	484.5641	-43.4044	101.5456	-314.169
7	-150.264	-656.615	-436.499	-68.6969	-827.913	-53.0503	-19.6449	361.9726	408.7208
8	-176.737	300.4759	-19.7681	112.1575	-575.259	-131.162	-63.7908	579.4186	-288.559
9	15.69418	-1116.93	-1407.2	91.50958	-579.826	523.1709	-48.3414	193.379	431.6757
10	24.76572	-832.126	-385.276	339.8002	-620.886	281.0852	-39.021	333.4626	-802.961
11	90.95883	-1623.86	603.0679	518.1561	-408.379	-65.0922	-45.2584	112.0763	-448.322
12	-140.695	-861.891	-427.345	-137.072	-117.46	80.21279	88.53205	149.0484	464.5205
13	4.236589	-255.719	-111.494	-130.602	-124.033	-53.2623	112.8268	126.5798	58.85751
14	-31.7488	46.01778	-431.789	-194.404	-252.935	-54.1053	-285.376	288.1908	-188.178
15	-690.468	-862.331	-262.117	-267.516	30.67083	49.5843	151.7035	367.5866	-34.7994
16	-516.482	-730.066	218.821	-281.867	-345.334	-12.8379	-289.12	367.5829	28.72253
17	-31.7572	240.1382	179.1961	-403.462	-296.254	-168.092	-50.6934	405.4612	390.0343
18	-163.456	-923.525	179.9598	-517.96	-275.48	-122.244	-47.594	94.53911	-276.421
19	30.66011	112.0911	-71.5105	-583.79	-182.479	488.64	192.3082	-26.0106	-917.464
20	41.93825	-934.613	670.4686	-303.137	-325.894	477.3336	215.2746	24.92132	681.4338
21	39.68131	-730.053	537.1939	-443.462	-431.132	449.1843	202.0524	-13.6313	411.3602
22	-21.7497	-149.075	541.4408	-412.121	-195.717	456.9239	-94.5469	34.87632	-5.97599
23	-1037.53	-154.593	212.8102	-459.314	-454.58	514.0134	-71.4573	26.74379	506.5951
24	-223.918	-838.298	-303.379	-713.863	-104.49	260.0616	-57.0507	412.0399	780.7909
25	191.1557	-343.927	569.9927	-714.004	78.49982	263.882	-51.6995	311.1685	537.2618
26	154.8154	-422.045	632.5702	-855.407	-419.827	511.6878	236.749	285.1908	-87.7245
27	-235.983	-824.31	252.2993	-877.296	-316.569	307.8723	-78.1075	530.6572	579.8737
28	-292.622	-793.845	309.3693	-885.274	-235.513	-826.058	-89.8084	540.9021	796.2963
29	-262.442	-392.864	657.0864	-284.606	-38.6292	-821.414	191.6412	416.7371	381.025
30	-948.063	-507.626	598.9154	-791.365	-8.45453		-87.209	419.227	595.8357

Fig. 67 : Résultat des optimums trouvés pour la fonction pénalisante

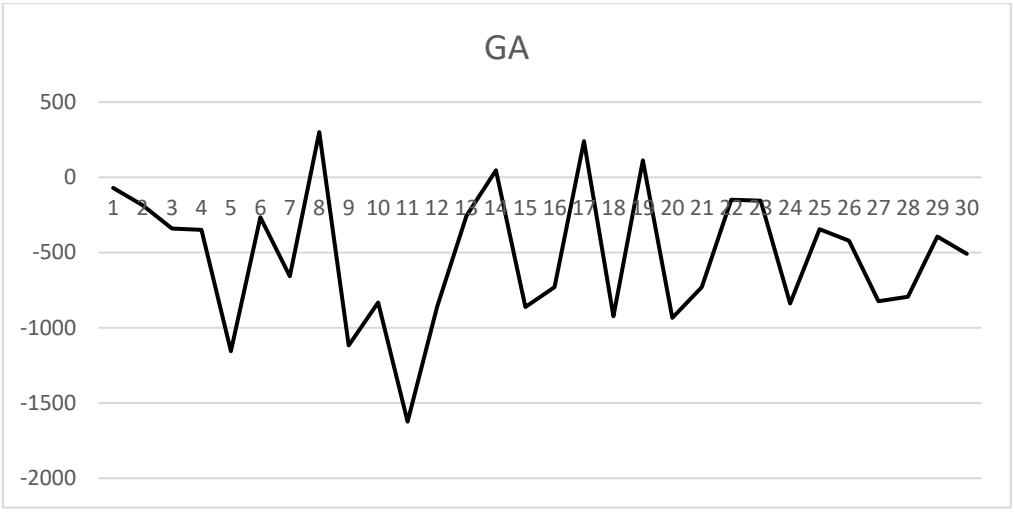


Fig. 68 : Recherche d'un optimum à l'aide de l'algorithme GA proposé par Galapagos

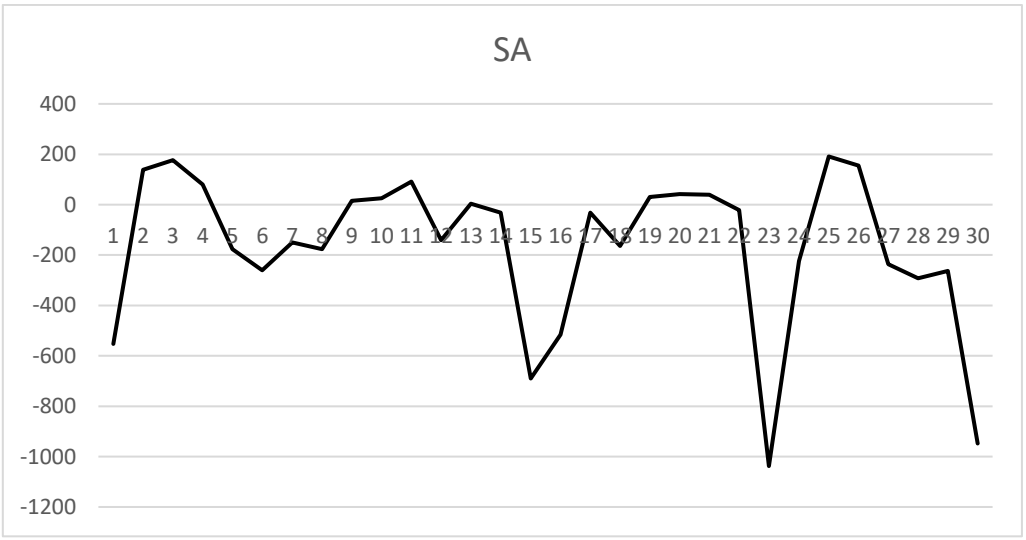


Fig. 69 : Recherche d'un optimum à l'aide de l'algorithme SA proposé par Galapago

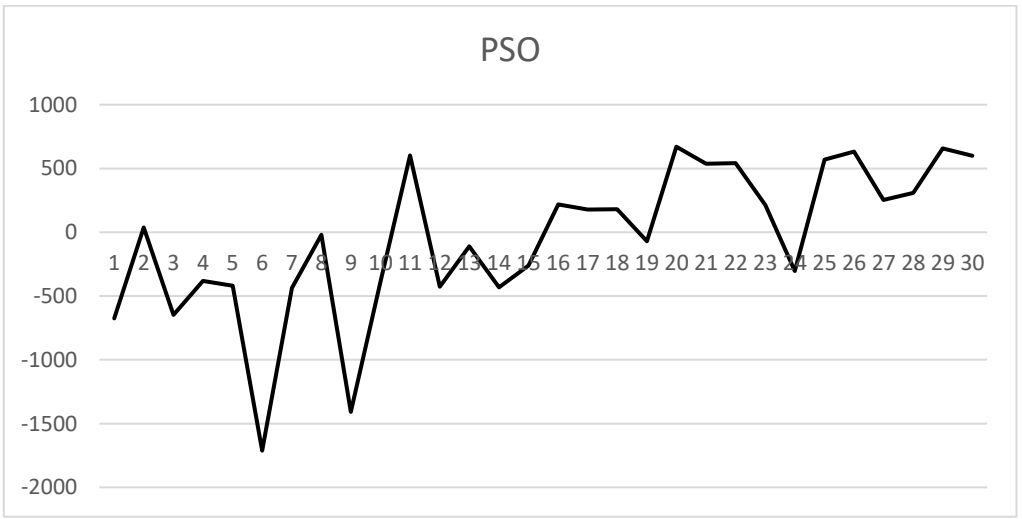


Fig. 70 : Recherche d'un optimum sur 30 simulations à l'aide de l'algorithme PSO proposé par Silvereye

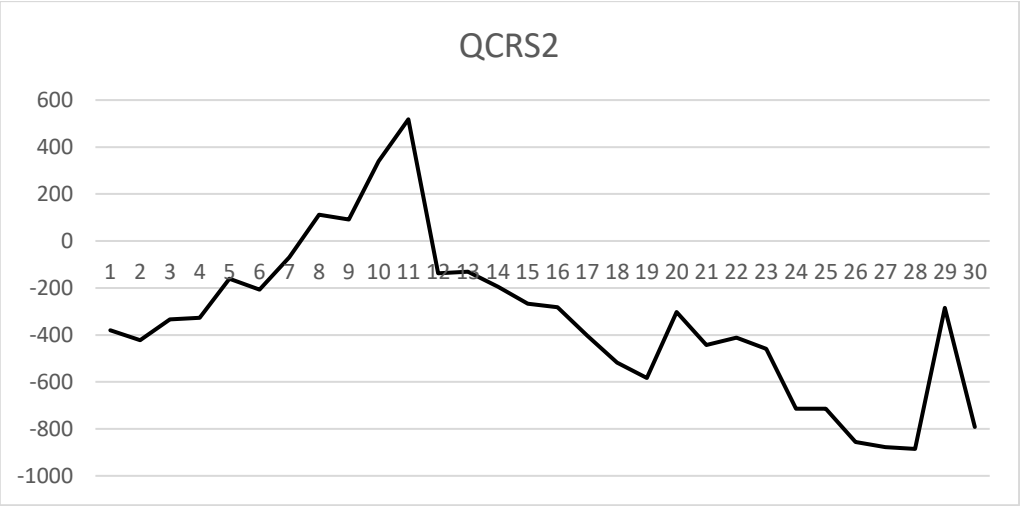


Fig. 71 : Recherche d'un optimum à l'aide de l'algorithme QCRS2 proposé par Goat

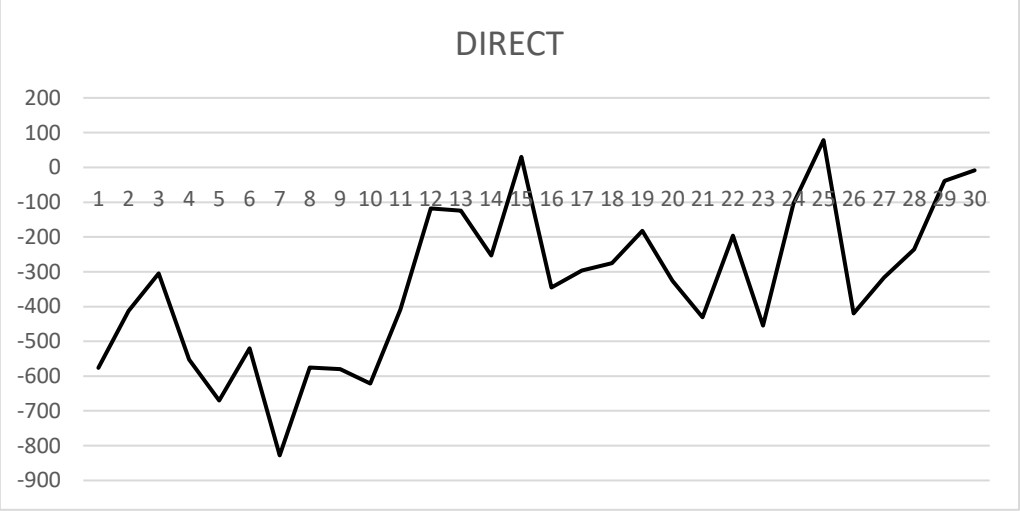


Fig. 72 : Recherche d'un optimum à l'aide de l'algorithme COBYLA proposé par Goat

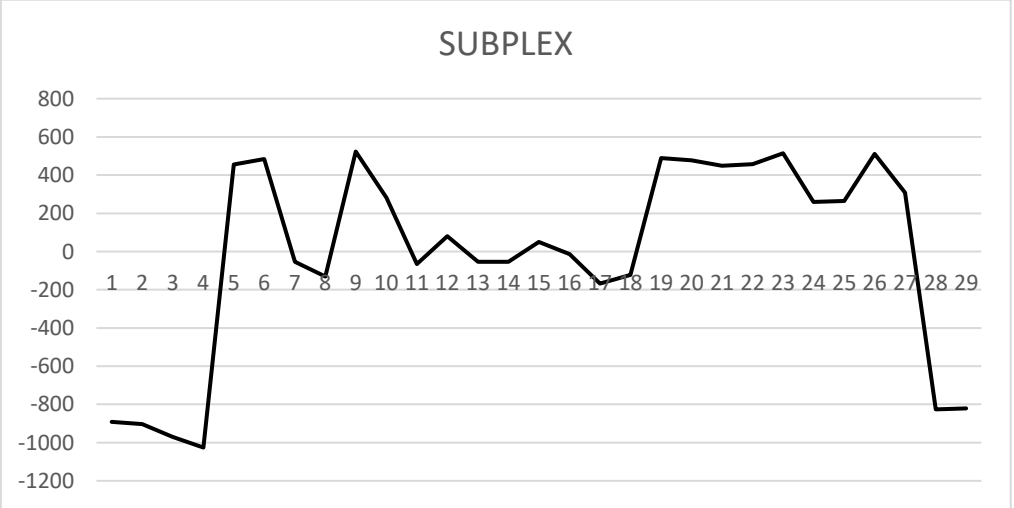


Fig. 73 : Recherche d'un optimum à l'aide de l'algorithme COBYLA proposé par Goat

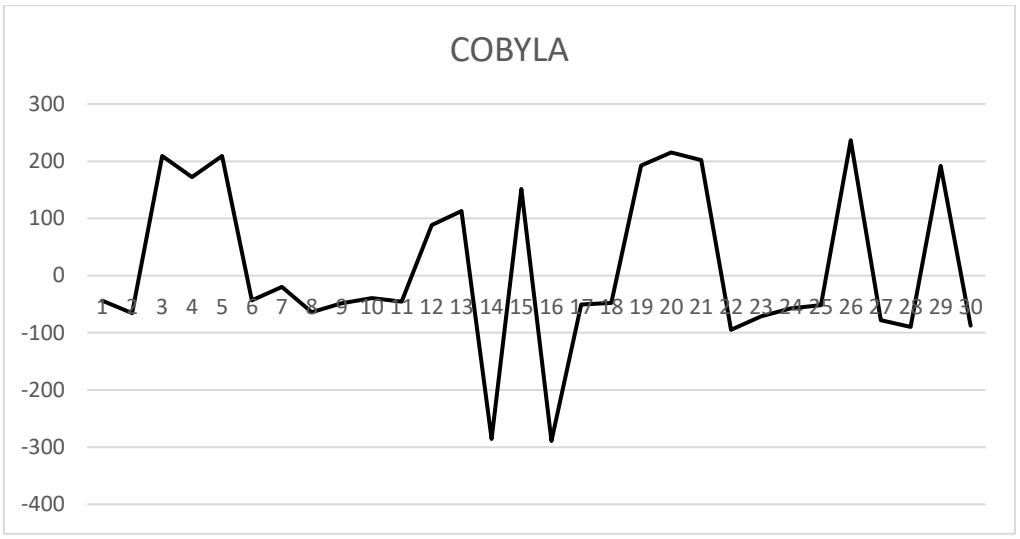


Fig. 74 : Recherche d'un optimum à l'aide de l'algorithme COBYLA proposé par Goat

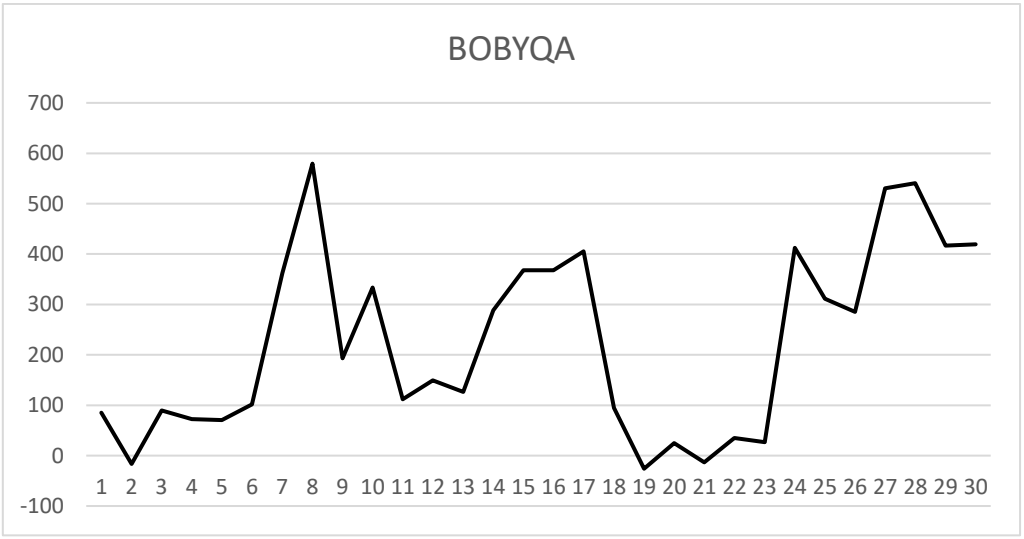


Fig. 75 : Recherche d'un optimum à l'aide de l'algorithme COBYLA proposé par Goat

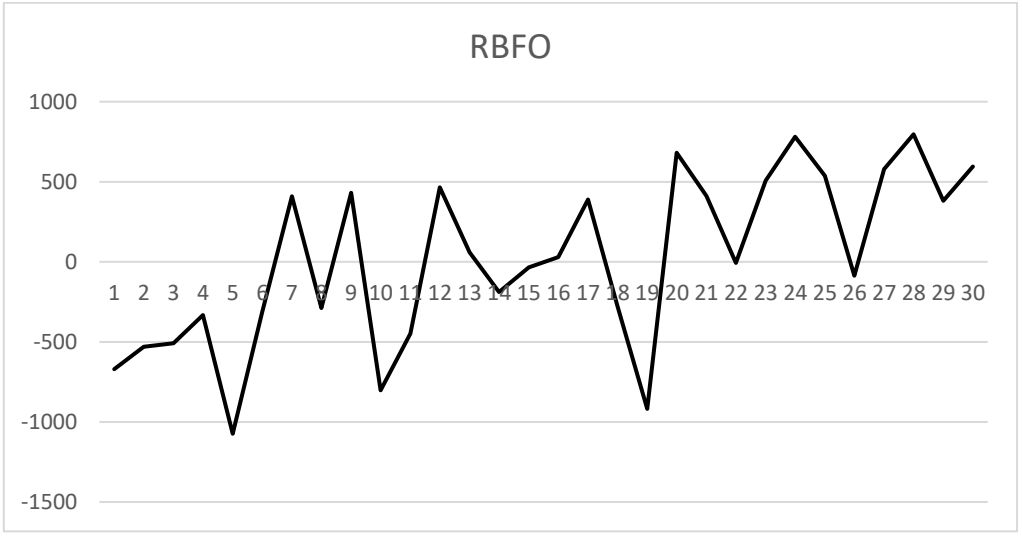


Fig. 76 : Recherche d'un optimum à l'aide de l'algorithme COBYLA proposé par Goat

La courbe des résultats obtenus par l'Algorithme Génétique ou le Simulated Annealing ne montre aucun signe d'amélioration sur la durée (voir fig.68, 69). A nouveau, il faut prendre en compte le fait que seules 3 générations ont eu le temps d'apparaître dans le cas du GA. Le caractère aléatoire des résultats tend à confirmer les propos sur la discontinuité du paysage de solutions par introduction d'un paramètre indépendant.

Silvereye obtient avec son PSO le second meilleur résultat et présente globalement une amélioration.

Si l'on étend le nombre de simulation par contre on se rend compte que très vite la courbe prend un caractère plus aléatoire comme pour les méthodes métaheuristique précédentes (voir fig. 77). Cela s'expliquerait, à l'inverse de précédemment, par un manque de chance sur les paramètres de départ.

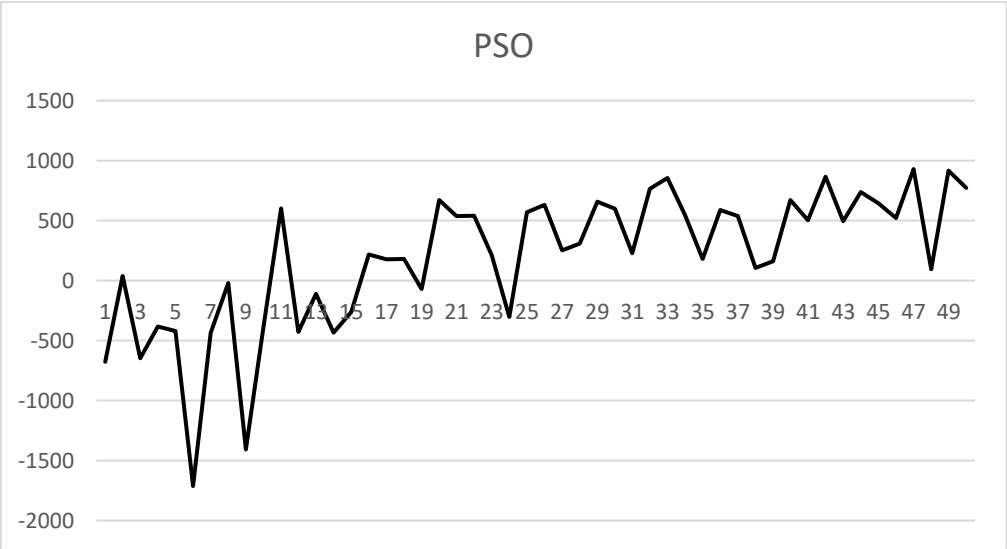


Fig. 77 : Recherche d'un optimum sur 50 simulations à l'aide de l'algorithme PSO proposé par Silvereye

GOAT obtient à nouveau des mauvais résultats. L'algorithme QCRS2 tombe dès le début sur une bonne solution par ses propriétés stochastiques mais s'enfonce directement après. A nouveau, l'optimisation a été prolongée à 50 simulations et comme pour les autres algorithmes la courbe prend une allure aléatoire et ne fait que globalement s'écarter des bonnes solutions (voir fig. 78). DIRECT présente la même allure que PSO et SUBPLEX, COBYLA et BOBYQA présentent tous les 3 des courbes d'allures aléatoires.

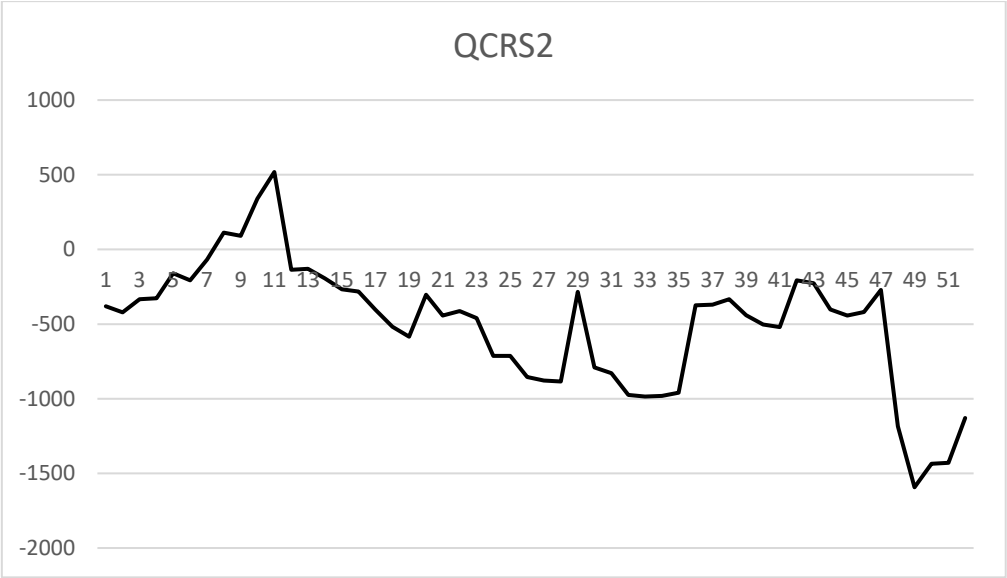


Fig. 78 : Recherche d'un optimum pour 50 simulations à l'aide de l'algorithme QCRS2 proposé par Goat.

Opossum semble présenter une certaine amélioration sur le peu de simulations, mais comparé aux résultats précédents, la courbe présente des écarts plus marqués qui s'expliquent par une discontinuité du paysage elle-même plus marquée.

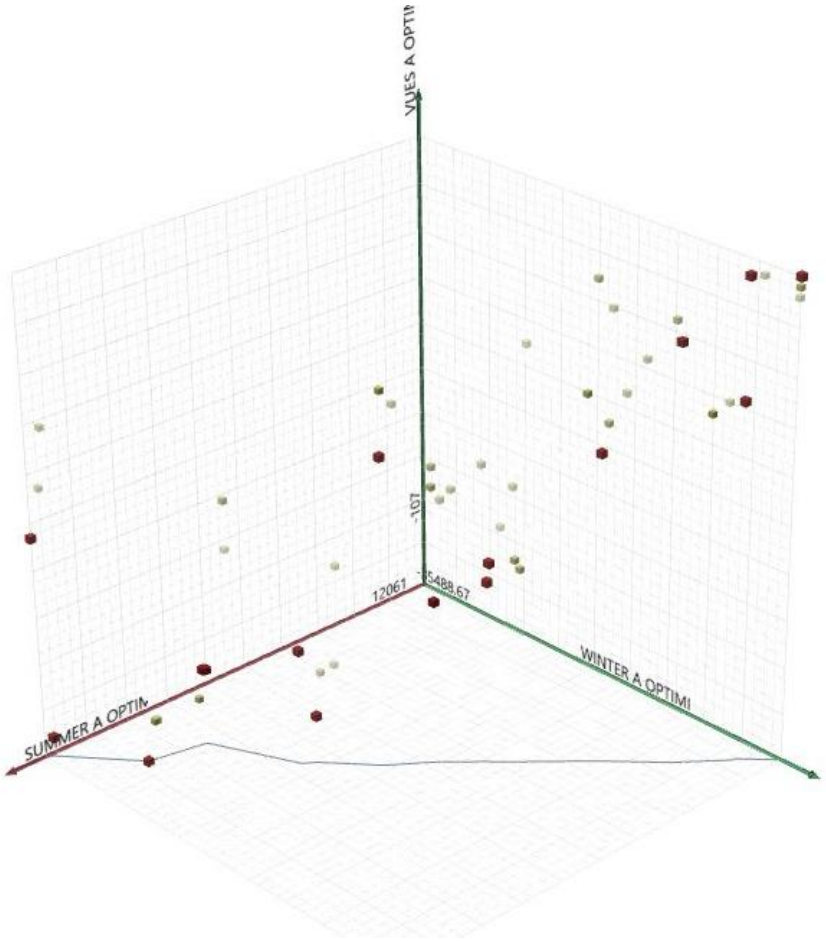


Fig. 79 : Visualisation des résultats obtenus par HYPE dans Octopus

Discussion des résultats

Les résultats obtenus ont tendance à confirmer le propos énoncé sur le manque de relation entre les paramètres. Ce manque de relation amène à plus de discontinuité dans le paysage de solution ce qui le rend plus imprévisible vis-à-vis des algorithmes. Le semblant d'amélioration de certains résultats montre qu'il existe tout de même un certain gradient dans la discontinuité. Cela est probablement dû aux quelques éléments influant communs aux 2 problèmes comme par exemple les cheminées qui, d'une part, empêchent l'exposition au soleil et d'autre part empêchent l'accès visuel.

Le modèle d'étude est ici très simple, il présente peu de paramètres et seul 3 objectifs sont à considérer. En fonction du type de projet, l'architecte pourrait être amené à manipuler des modèles définis par des centaines de paramètres et une quantité importante d'objectifs. Dans ces cas-là une extrapolation mathématique du paysage sur base de modèle à l'aide d'algorithmes comme le RBFOpt est sans doute discutable. L'extrapolation mathématique ne sera efficace que pour les paysages de solutions continues, c'est d'ailleurs sans doute pour cela qu'elle est efficace dans le cas d'optimisations à objectifs simples. L'usage des méthodes directes et métaheuristique est mieux indiqué mais nécessite toujours la mise au point de la fonction pénalisante. A nouveau le cas étudié ici est très simple et en réalité, la complexité du bâtiment ne permet pas ce genre de décisions arbitraires. La seule solution disponible pour le moment, dans l'environnement Grasshopper, est donc l'optimisation multi-objectif par Octopus car elle se débarrasse du problème de mise au point de la fonction pénalisante. Cela sans compter bien entendu les logiciels tiers vers lesquels l'information peut être exportée.

Note sur la représentation des données

La représentation des résultats ainsi que l'interface jouent un rôle extrêmement important. Les techniques de visualisation des résultats sont essentielles à l'extraction d'informations sur les performances. C'est pourquoi en parallèle au développement des nouvelles techniques d'optimisation, une attention particulière est portée sur l'interaction avec le modèle de recherche afin que l'exploration soit la plus efficace possible. La représentation des données lorsqu'il n'est question que de 2 paramètres est présentée sous forme d'une topographie en 3 dimensions mais qu'en est-il lorsque plus de paramètres sont mis en jeux

De manière générale, la solution la plus populaire est l'usage des coordonnées parallèles mais la lecture peut parfois être assez fastidieuse, il est alors également courant de retrouver la représentation en étoile.

Reprenant une forme de visualisation traditionnelle en mathématique comme base de recherche, Wortmann porte un intérêt particulier à transposer ces outils à l'univers de l'architecture en les rendant plus accessibles. Ses résultats ont mené à un mode de représentation instinctif et interactif à travers son solveur pour l'optimisation simple sur base de modèle, Opossum. Il appelle ça la cartographie de l'espace des solutions, c'est un dire une représentation abstraite des solutions possibles [Wortmann, 2017] (voir fig. 80). Cette solution permet d'appréhender le paysage de solution quelque-soit le nombre de dimensions au problème.

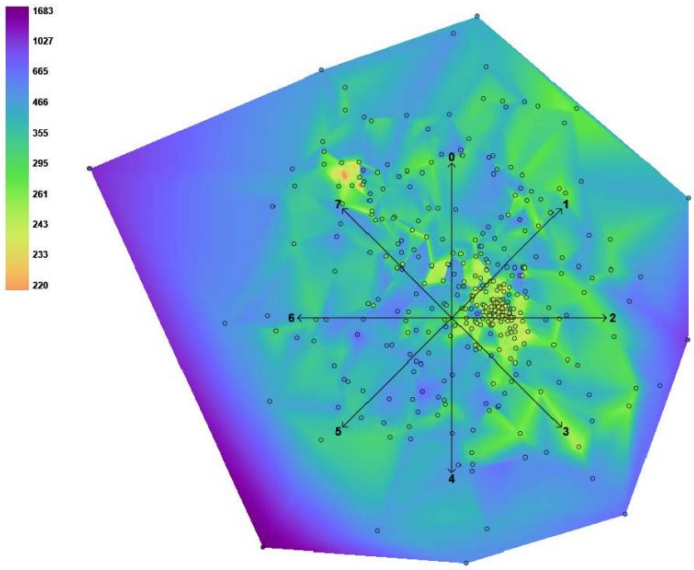


Fig. 80 : Exemple de paysage de solutions produit sur base d'une minimisation à l'aide du solveur Opossum. [Wortmann, 2017]

L’ensemble des solutions est présenté sous forme d’une surface 2D colorée à laquelle a été superposé un graph en étoile pour informer des paramètres. L’indicateur couleur informe l’architecte des bonnes solutions du paysage sur base de l’extrapolation faite par l’algorithme RBFOpt. Les points visibles correspondent aux simulations déjà réalisées mais l’architecte est libre de choisir des solutions dans le paysage qui n’ont pas encore été simulées sur base de l’indicateur coloré. La simulation aura alors lieu le paysage s’affinera et un retour visuel de la forme sera disponible à travers Rhinoceros (voir fig. 81).

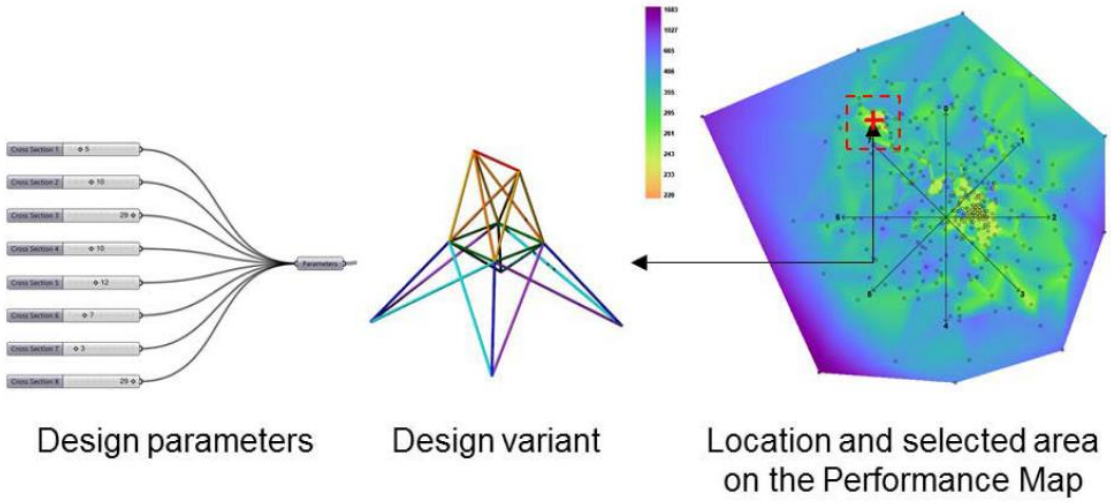


Fig. 81 : Interaction avec le paysage de solution et feedback visuel sur la forme ainsi que sur ses paramètres

Cette approche approximative par la couleur renforce le processus d’exploration. Une remarque néanmoins par rapport à cette approche de l’interface est que l’architecte se retrouve face à une infinité de solution et l’exploration peut sembler interminable. Cette tentative est destinée à de l’optimisation simple étant donné qu’elle est destinée à venir compléter le solveur Opossum.

Il sera discuté plus tard en remarque d’une possibilité d’étendre ce principe à une optimisation à objectif multiple par superposition des topographies.

Octopus est le solveur le plus développé en terme d'interaction pour le moment. Il présente l'ensemble des résultats sur un front Pareto en 3 dimensions. Il est important de noter que le front est extrapolé dans le cas où l'on a affaire à plus de 3 objectifs, ce qui ne rend pas la lecture des résultats efficace. Les résultats peuvent être présentés sous forme de maillages donnant à l'architecte une vision directe des solutions qui lui sont proposées avec une indication sur l'objectif avantagé pour chaque solution (voir fig.82). La position spatiale des solutions, contrairement à la méthode de Wortmann ne nous donne aucune information directe sur les paramètres mais sur les objectifs pris individuellement, une autre source d'information cohérente.

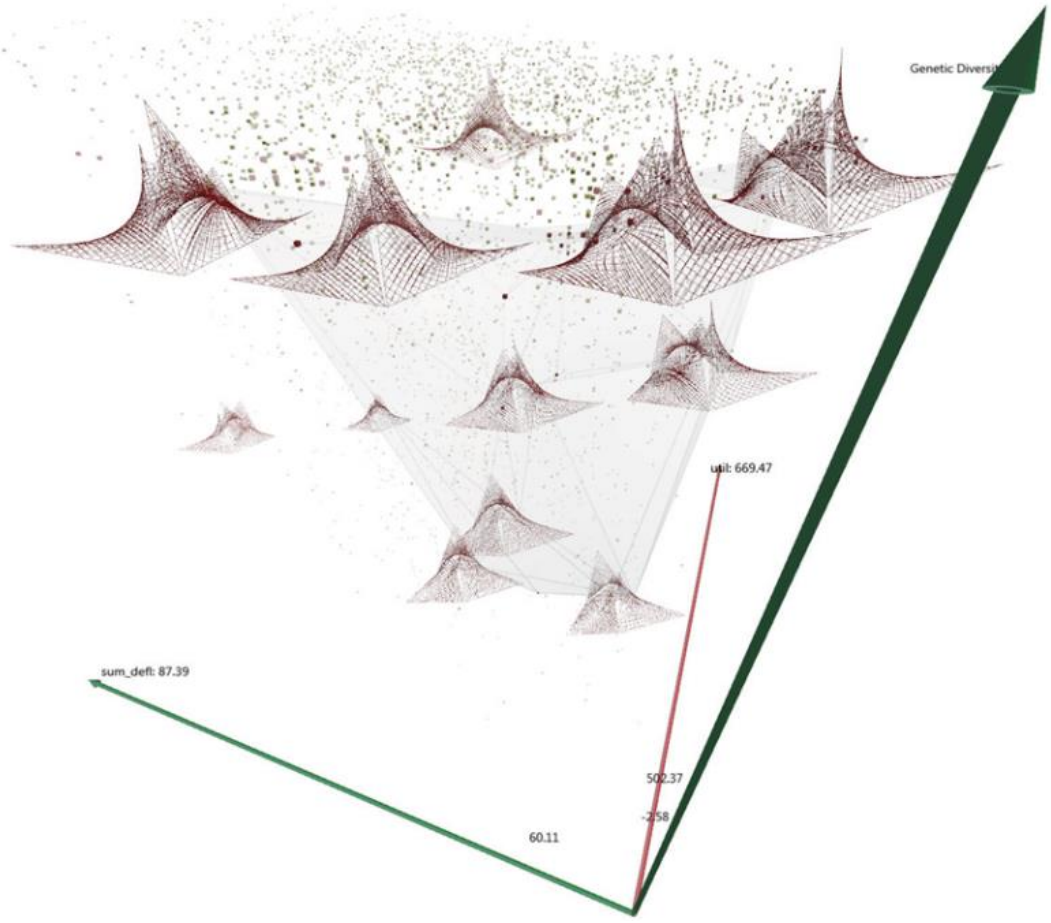


Figure 82: Pareto Front présentant directement l'ensemble des solutions formelles.

Une autre approche par Core Studio et Thornton Tomasetti (2017) présente l'ensemble des solutions à travers une interface web. L'avantage est que ce dernier se nourrit de fichier .csv (Coma-Separated Values), un format informatique ouvert représentant des données tabulaires. Cela signifie que l'utilisateur n'est plus dépendant de l'un ou l'autre Solveur et l'interface que ce dernier impose. L'information numérique est présentée sous forme d'un graphique à coordonnées parallèles où l'architecte décide d'exposer les informations qui lui semble pertinentes, que ce soit concernant les paramètres ou chacun des objectifs atteints. Ce genre d'approche facilite également l'échange avec le reste des acteurs du projet, de manière plus anecdotique, son interface prévoit même un système de notation des solutions.

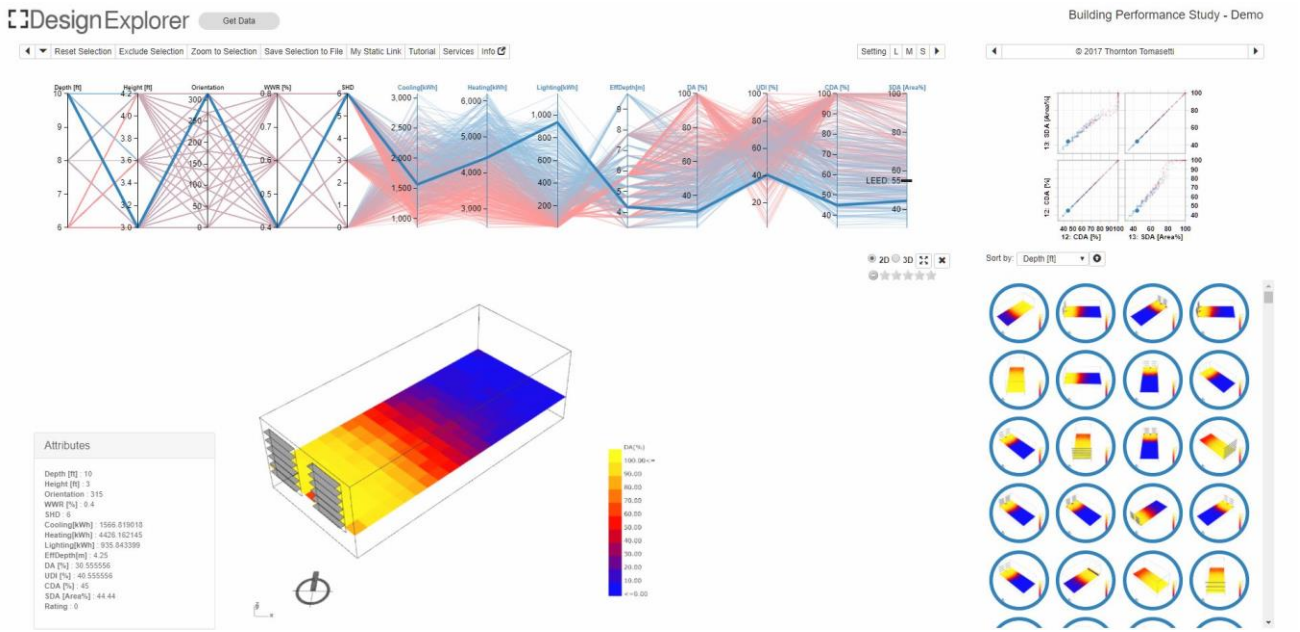


Fig.83 : Visualiseur du projet DesignExplorer [Core Studio & Thornton Tomasetti, 2017]

Pistes pour le futur

Sur base des problèmes rencontrés, une réflexion a eu lieu sur le développement d'un éventuel futur outil d'optimisation.

Pour revenir à l'exemple précédent, il est important de noter que le facteur mis en place lors de la définition de l'équation pénalisante se base sur les résultats obtenus lors de la première optimisation. Sans ces derniers, il aurait été impossible de déterminer un facteur convenable. Dans ces cas-là une pratique courante en mathématique est de diviser l'ensemble des résultats d'une optimisation simple par la meilleure valeur trouvée ayant pour résultat une série de valeur allant de zéro à un pour chaque objectif. A partir de là il est plus aisé d'ajouter des facteurs d'importance à chaque objectif dans la fonction pénalisante.

Afin de continuer dans ce raisonnement, prenons un cas pratique. Le modèle consiste en un bâtiment type boîte à chaussure possédant une seule ouverture sur sa face la plus au sud (voir fig. 84). Le bâtiment se trouvent dans une forêt dont les arbres perdent leurs feuilles en hiver. Le contexte n'est donc pas le même en hiver ou en été.

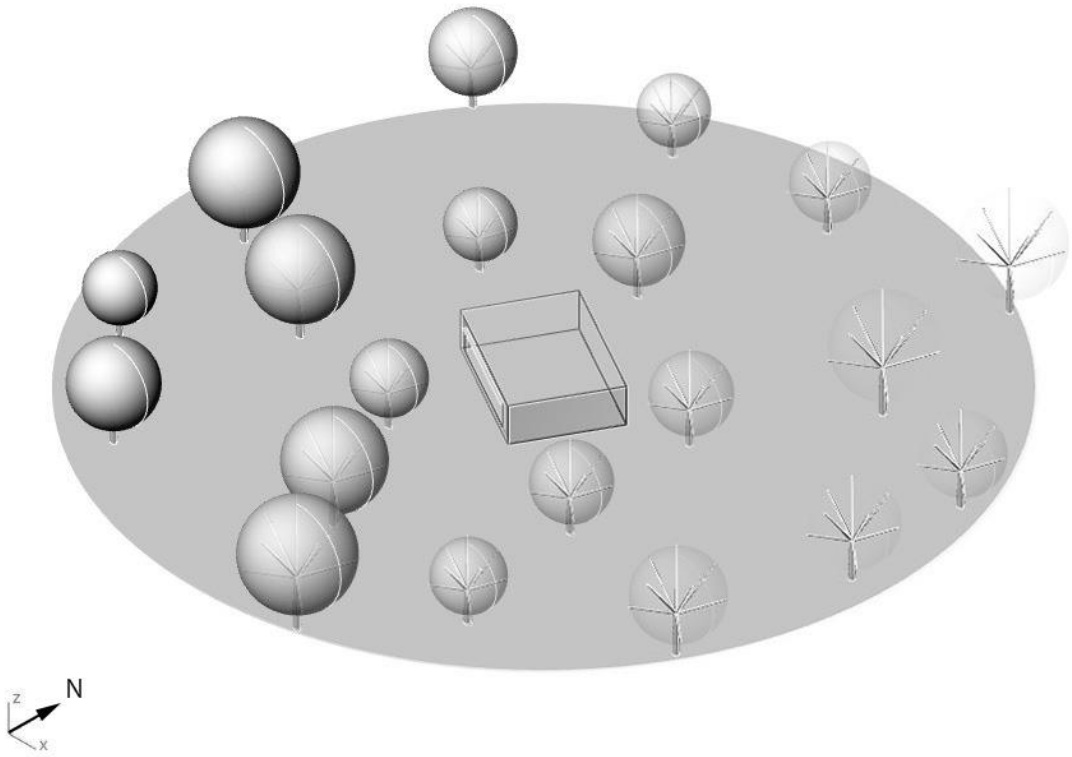


Fig. 84 : Présentation du modèle d'étude.

Les paramètres consistent en la hauteur sous plafond, l'orientation du bâtiment, et la profondeur de la casquette d'ombrage de la baie. Le projet est optimisé sur base des apports solaires en hiver, des apports solaires en été et sur l'apport de lumière directe à l'intérieur du bâtiment.

Sans passer par un solveur, déterminons une série de points sur le paysage capable qui représente ici un hexagone. Ces points représentent une valeur pour chacun des paramètres et pour chaque trio de données sera calculé la performance des objectifs séparément.

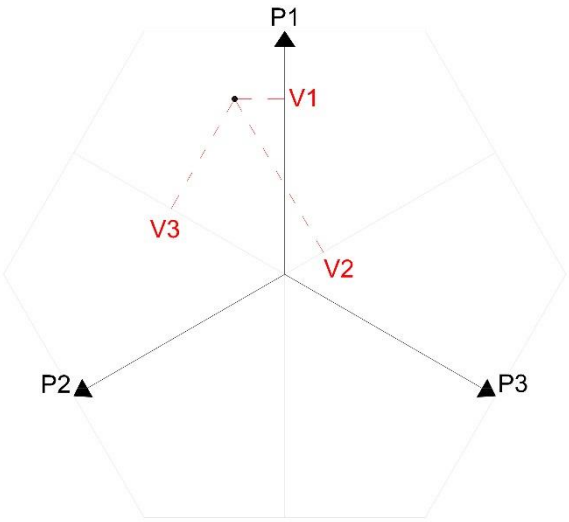


Fig. 85 : Délimitation du paysage de solution par les 3 paramètres à travers un système de coordonnée radial.

La distribution radiale proposée dans les travaux de Wortmann semble être une approche efficace étant donné qu'elle informe directement des valeurs des paramètres.

La performance pour chaque point, après ensemble des calculs, est divisé par la valeur trouvée la plus haute ramenant chaque valeur à un domaine compris entre zéro et un.

On obtient alors les 3 topographies de solutions. Dans ce cas-ci les points ont simplement été reliés entre eux par une série de courbes de Bezier en 2 sens. Ces 2 ensembles de courbes forment ensuite le paysage de solution.

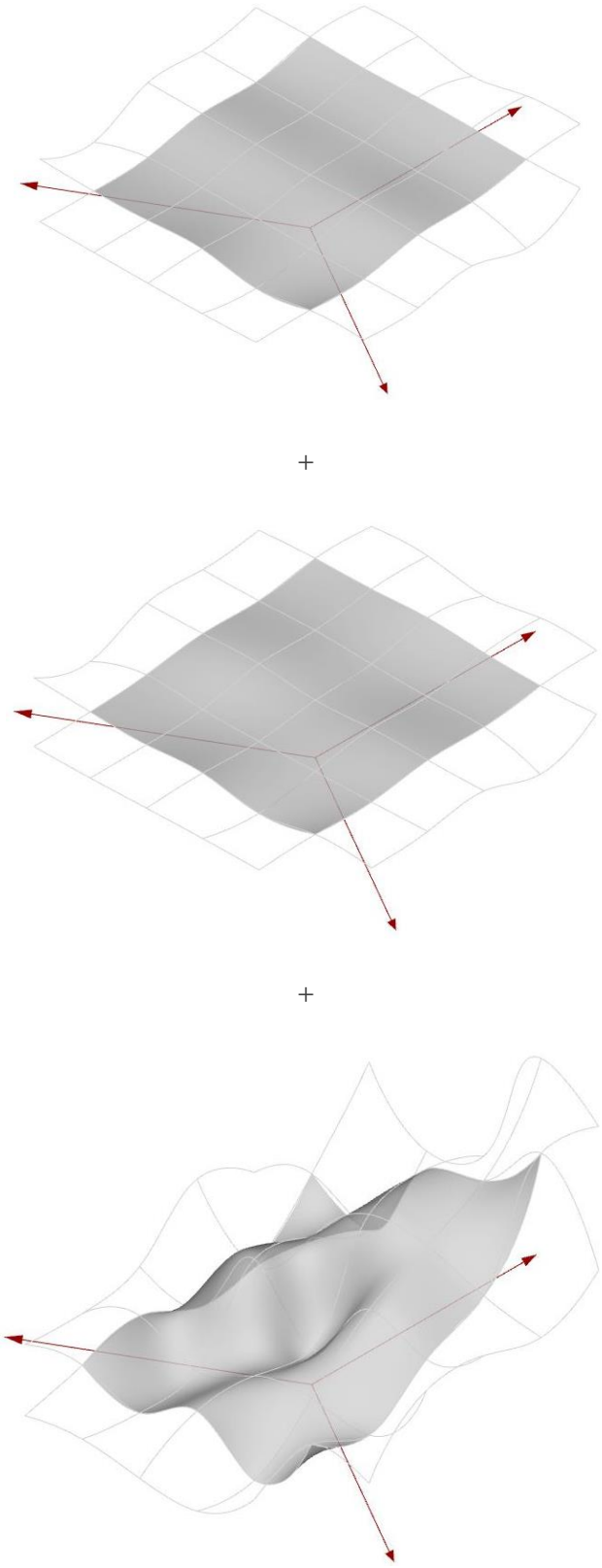


Fig. 86 : De haut en bas ; Paysage de solution pour l'apport solaire en hiver, paysage de solution pour l'apport solaire en été et paysage de solution des apports de lumière directe.

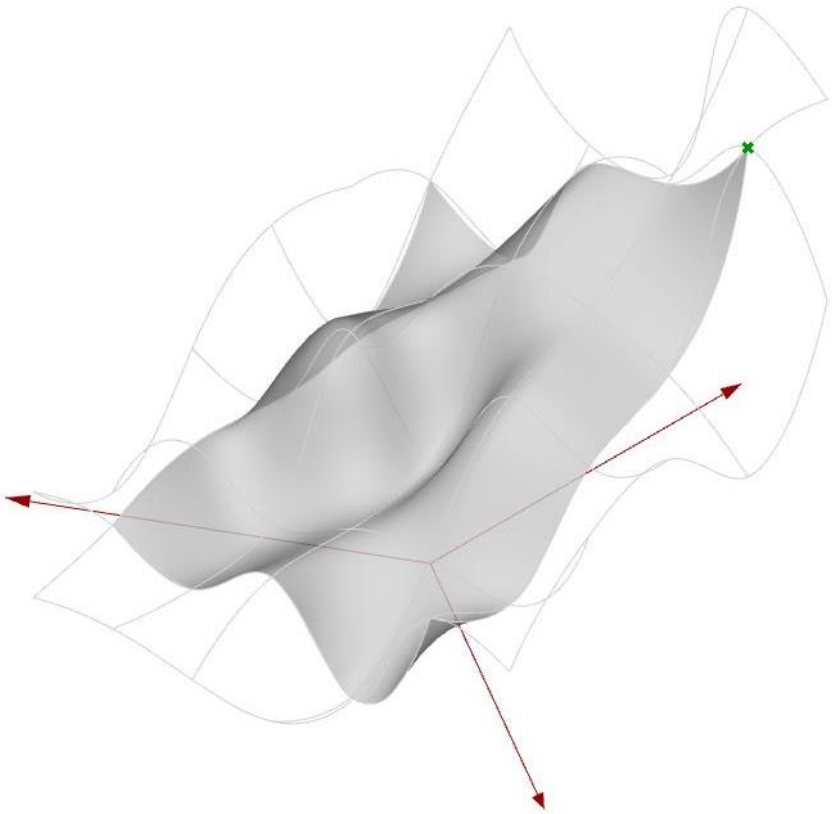


Fig. 87 : Paysage de solutions globales sur base de l'addition des topographies calculées précédemment.

L'addition des 3 paysages donne une topographie correspondant aux résultats d'une optimisation à objectifs multiples, en effet, chaque objectif est calculé séparément. De plus, le paysage global est construit après la simulation, ce qui nous permet de mettre en place une fonction pénalisante et de pouvoir l'ajuster en direct sans devoir refaire de simulation. Un autre avantage important est le fait que chaque simulation est lancée séparément, les résultats sont mis en relation par après, cela veut dire que sur base des mêmes paramètres et des mêmes contraintes, il est possible d'ajouter les couches d'informations au fur et à mesure en fonction des désirs et du temps disponible. Par exemple, dans l'exemple précédent (voir fig.84), l'optimisation dans un premier temps des performances vis-à-vis de l'apport solaire en été et en hiver tend à faire considérer l'intégration d'un autre calcul de performance qu'est la consommation énergétique. Dans un workflow typique, cela voudrait dire de tout recommencer alors qu'ici la couche d'information serait ajoutée indépendamment. Cela veut également dire que le travail peut être partagé à travers plusieurs stations de travail. Sur base d'un même modèle de départ, l'ingénieur en thermique peut mettre en place une optimisation telle que l'orientation du bâtiment, et l'ingénieur structure une minimisation de la matière à utiliser.

Dans l'exemple si dessus l'extrapolation se fait de manière simpliste mais il est envisageable d'y intégrer des algorithmes tel que le RBFOpt qui prouve être assez efficace lorsqu'il s'agit d'optimisation à simple objectif. A nouveau étant donné que chaque topographie est calculée de manière séparée, l'algorithme utilisé serait l'algorithme le plus pertinent à réaliser l'optimisation.

Dans l'éventualité d'une suite à ce travail le développement d'un environnement comme celui décrit ci-dessus est une chose à laquelle je m'intéresserai.

Conclusion

Le travail traite d'optimisation mais cette question fait partie d'un débat plus général sur les processus sous-jacents aux interfaces digitales que l'on retrouve dans la pratique professionnelle aujourd'hui.

Dans peu de temps le bouton « optimiser » apparaîtra dans les logiciels dédiés à l'architecture. C'est l'occasion de remettre en cause l'usage de l'informatique en architecture en général. Le but de ce travail est la prise de connaissance d'une nouvelle approche et de son intérêt. Les enjeux du bâtiment ainsi que la quantité d'informations sont tels aujourd'hui, que le projet tend à s'échapper des mains de l'architecte. L'approche numérique par l'optimisation peut permettre de récupérer un certain contrôle. L'ordinateur, indispensable aujourd'hui, ne représente en majorité encore qu'une étape supplémentaire au projet d'architecture. Il ne tient cependant qu'à l'architecte de l'intégrer complètement dans son travail. L'informatique en architecture est encore mal perçue. Girard (2017) fait mention de technophobie. La notion de lâcher prise, discutée par Bourbonnais (2015), propre à l'entreprise d'exploration à travers l'optimisation, peut paraître comme une perte de contrôle vis-à-vis de la machine, hors il ne tient qu'à l'architecte de définir les limites de son exploration.

En terme d'optimisation, il est intéressant de constater que quelque-soit le projet certains paramètres indispensables à l'architecture, tels que la qualité des espaces, la qualité des lumières, des ambiances, les aspects culturels, sociaux et économiques soient ininterprétables mais également que même au niveau scientifique, l'aspect virtuel des choses remet en cause l'approche générative. La pertinence des processus computationnels n'est plus à prouver et le travail démontre que le recul nécessaire dans cette démarche tend même à renforcer l'implication de l'architecte et la coopération dans le projet d'architecture.

Bibliographie

- Aish, R., Hesselgren, L., Parrish, J., Whitehead, H., Menges, A.(2006). *Instrumental Geometry*, AD, 42-53.
- Alfaiate, P., Leitao, A.(2017) *Luna Moth Supporting Creativity in the Cloud.* ACADIA
  2017, Cambridge.
- Alonso, H. D.(2010) *Exuberance I don't know; excess, i like*. AD, 70-77.
- Attia S. et al.(2013) *Assessing the Gaps and Needs for Integrating Building Performance Optimisation*. Energy And Buildings.
- Barbisan, N., Girard, C. (2013) *Architecture paramétrique: au delà de la morphogénèse visuelle*. ENSA.
- Blanchard, J. (2018) *Algorithmes évolutionnaires.* Université de Namur.
- Bradner, E., Lorio, F., & Davis, M. (2014). *Parameters Tell the Design Story: Ideation and abstraction in Design Optimization* Symposium on Simulation for Architecture and Urban Design. Tampa: Autodesk Research.
- Brownlee, J. (2011) *Clever Algorithms : Nature-Inspired Programming Recipes.* LuLu.
- Burry, M. (2013). *Scripting Cultures*. AD, 00-14.
- Celani, G., and Vaz, C.(2017) *CAD Scripting And Visual Programming Languages For Implementing Computational Design Concepts:A Comparison From A Pedagogical Point Of View*. International Journal of Architectural Computing, 10(1), 121–138.
- Cichocka, J., Browne, W. N., & Ramirez, E. R. (2015). *Evolutionary Optimization Processes as Design Tools: Implementation of a revolutionary swarm approach.* 31th International PLEA Conference ARCHITECTURE IN (R)EVOLUTION. Bologna.
- Cichocka, J., Browne, W. N., & Ramirez, E. R. (2017a). *Optimization In The Architectural Practice: An International Survey*. CAADRIA 2017. Hong Kong.
- Cichocka, J., Migalska, A., Browne, W. N., & Rodriguez, E. (2017b). Silvereye: *The implementation of Particle Swarm Optimization Algorithm in a design Optimization Tool*. CAAD Futures 2017,151-169, Singapore: Springer.
- Cohen, P. S. (2011). *L'alliance du contrôle et de l'imprévisibilité* dans Architecture numérique : culture et stratégies opératoires. *d'architectures*, 30-49.

- Core Studio et Thornton Tomasetti (2017) http://core.thorntontomasetti.com/design explorer/ (consulté le 16/03/2018)
- Couwenbergh, JP. (2014) *L'approche computationnelle un changement de paradigme en conception.* LOCI.
- Davis D. (2012) *Quantitatively Analysing.* International journal of Architecture Computing, 3(12), 307-319.
- Davis, D., Burry, J., & Burry, M. (2011). *Untangling parametric schemata enhancing collaboration through modular programming introduction why parametric modelling can be difficult*. CAAD Futures 2011, ULg.
- Davis, D., Burry, J., and Burry, M.(2012) *Yeti: Designing geometric tools with interactive programming,* Proceedings of the 7th International Workshop on the Design
  and Semantics of Form and Movement, 196-202, Victoria University of Wellington.
- Designalyze Podcast Interview D. Davis 2014 (consulté le 13/04/2018)
- Eligius M. T., Hendrix, P. M., Ortigosa,(2001). *On success rates for controlled random search*. J. Global Optim. 21, 239-263.
- Evins, R.(2013) *A review of computational Optimization methods applied to sustainable building design.* Renewable ans Sustainable Energy Review 22, 230-245.
- Fasoulaki, E. (2007). *Genetic Algorithms in Architecture: a Necessity or a Trend.* Cambridge: Massachusetts Institute of Technology.
- Finkel D. E. (2003*), DIRECT Optimization Algorithm User Guide*. Center for Research in Scientific Computation North Carolina State University Raleigh, NC 27695-8205.
- Frey, C. B., Osborne, M., A. (2013) *The_Future_of_Employment,* Oxford University.
- Girard, C. & Morel, P. (2017). «Nous pensons qu'il faut dynamiter la façon d'enseigner l'architecture en France . Consulté en septembre 2017 sur www.lemonde.fr.
- Grady, M. (2011) *Architecture numérique : culture et stratégies opératoires.* dans Architecture numérique : culture et stratégies opératoires. *d'architectures*, 30-49.
- Haitao L., Liu L., Yang Q. (2007), *A novel method to design flexible URAs*, Journal of Optics A: Pure and applied Optics, 502-505.

- Janssen, P., and Chen, K.,(2011) *Visual Dataflow Modeling: A Comparison of Three Systems,* in Leclercq, P., Heylighen,A., and Martin, G., eds., CAAD Futures, Liège: Les Èditions de l'Université de Liège, 2011, 801–816.
- Kaelo P., and M. M. Ali,(2006) *Some variants of the controlled random search algorithm for global optimization*, J. Optim. Theory Appl. 130 (2), 253-264.
- Leitao . (2017). *Algorithmic-Based Analysis, Design and analysis in a multi back-end Generative Tool.* CAADRIA 2017, 137-147, Hong-Kong
- Leitao, A.(2012). *Programming Languages For Generative Design A Comparative Study.* International journal of Architecture Computing, 1(10), 139-162
- Liddament, T.(1998) *The Computationalist Paradigm in Design Research*. Design Studies, 20(1), University of London.
- Luke J.(2011) *Essentials of Metaheuristics,* http://www.lulu.com (accessed February. 2018)
- Lynn, G. & Foster, M. (2011) *Software Monocultures*. Yale School Of Architecture.
- Nelder J., Mead, R.(1965) *A simplex method for function minimization* , Computer Journal, 7(4), 308-313.
- Nguyen, A.(2014) *A review on simulation-based optimization methods applied to building performance analysis*. Renewable and Sustainable Energy Review 31, 101-112
- Picon, A., & Razavi, A. (2011). *Architecture numérique : culture et stratégies opératoires.* d'architectures, 30-49.
- Powell M.J.D. (2009) *The BOBYQA algorithm for bound constrained optimization without derivatives*. DAMTP
- Powell M. J. D.(1998) *Powell, Direct search algorithms for optimization calculations*. Acta Numerica 7, 287-336 (1998).
- Rios, L.M. and Sahinidis,(2013) *Derivative-free optimization: a review of algorithms and comparison of software implementations*. J Glob Optim, 56(3), 1247-1293.
- Rocker, M. (2006) *When Code Matters*. AD, 16-25.
- Rutten, D. (2013) *Galapagos, On the Logic and Limitations of Generic Solvers.* AD, 132-135.
- Rutten, D. (2014) *Navigating multi-dimensional landscapes in foggy weather as an analogy for generic problem solving.* 16th International conference on geometry and graphics, Innsbruck.

- Sachin Joglekar's Blog (consulté le 03/03/2018).
- Stals, A. (2016). *How Do Small and Medium Architectural Firms Deal with Architectural Complexity? A look into digital practices*.eCAADe.Oulu.
- Terzidis, K. (2006). *Algorithmic Architecture*. Routledge.
- Terzidis, K. (2004). *Algorithmic design: A Paradigm Shift in Architecture?* In Architecture in the Network Society: Proceedings of the 22nd Conference Educaton in Computer Aided Architectural Design in Europe, 201–207. Copenhagen: eCAADe.
- Turrin M., Buelow P., Stouffs, R. (2011), *Design explorations of performance driven geometry in architectural design using parametric modelling and genetic algorithms.* Advanced Engineering Informatics, 25(4), 656-675.
- Vierlinger, R. (2013). *Multi Objective Design Interface*. Wien: TUWien.
- Wetter, M., Wright, J.(2004) *A comparison of deterministic and probabilistic optimization algorithms for non-smooth simulation-based optimization.* Building and environment 39, 989-999
- Whitehead, H. (2003). *Laws of Form.* B. Kolarevic (Ed.), Architecture in the Digital Age: Design and Manufacturing 89–113 Taylor & Francis.
- Wolfram, S.(2002) *A New Kind of Science*, Wolfram Media, Champaign, IL.
- Wortmann, T.(2016). *Black-Box Optimization methods for architectural design*, CAADRIA 2016, 177-186, Hong-Kong.
- Wortmann, T.(2017). *Opossum: Introducing and Evaluating a Model-based Optimization Tool for Grasshopper*. CAADRIA 2017, (pp. 283-293). Hong-Kong.
- Wortmann, T. (2017). *Surveying design spaces with performance maps.* International Journal of Architectural Computing.
- Wortmann, T., & Cichocka, J. (2017c). *Interfacing Architecture*, Engineering. IASS 2017 Masterclass, Hamburg, Germany.
- Wortmann, et al.(2017). *Are Genetic Algorithms Really the Best Choice for Building Energy Optimization?* SimAUD 2017. Toronto.
- Yang, D. et al. (2016*). Application of surrogate models for building envelope design exploration and optimization*.Proceedings of the Symposium on Simulation for Architecture and Urban Design (pp. 11-14), London, 16-18.