UNIVERSITE DE CORSE - PASCAL PAOLI
ECOLE DOCTORALE ENVIRONNEMENT ET SOCIETE
UMR CNRS 6134 (SPE)

Thèse présentée pour l'obtention du grade de

DOCTEUR EN INFORMATIQUE
Soutenue publiquement par

BASTIEN POGGI

Le 12 Décembre 2014

Développement de concepts et outils d'aide à la décision pour l'optimisation via simulation : intégration des métaheuristiques au formalisme DEVS

Directeurs :

M. Jean-François Santucci, Pr, Université de Corse
M. Thierry Antoine-Santoni, Dr, Université de Corse

Rapporteurs :

M. Jean-Jacques Chabrier, Pr Emérite, Université de Bourgogne
M. Stefano Chessa, Pr, Università di Pisa, Italie

Jury

M. Jean-François Santucci, Pr, Université de Corse
M. Thierry Antoine-Santoni, Dr, Université de Corse
M. Jean-Jacques Chabrier, Pr Emérite, Université de Bourgogne
M. Stefano Chessa, Pr, Università di Pisa, Italie
M. Antoine Aïello, Pr, Université de Corse
M. Mamadou Seck, Dr, Old Dominion University, Etats Unis

A mes parents, Marie-Andrée et Jean-Claude pour m'avoir donné le sens du travail et de l'honnêteté.

Remerciements

La réalisation de ce doctorat a sans aucun doute été une des périodes les plus riches de ma vie tant sur le plan professionnel, intellectuel et social. En effet, ce marathon de la connaissance m'a permis de baigner durant trois années dans le monde fascinant et enivrant de la recherche. Aux moments de doutes ont succédé de très nombreux moments de joies intenses gravés définitivement dans ma mémoire. Les lignes suivantes ne seront jamais suffisantes pour exprimer la reconnaissance que je porte aux personnes citées.

Avant tout, je tiens à remercier les professeurs Jean-Jacques Chabrier et Stefano Chessa d'avoir accepté d'être les rapporteurs du présent document ainsi que le docteur Mamadou Seck pour sa participation en tant que membre du jury. La qualité de leurs remarques a été pour moi un réel enrichissement pour mes travaux actuels et futurs.

Je tiens à remercier le professeur Jean-François Santucci ainsi que le docteur Thierry Antoine-Santoni de m'avoir encadré, conseillé, guidé, orienté et soutenu tout au long de ma thèse. Nos échanges plus riches les uns que les autres n'ont jamais cessé d'alimenter ma curiosité. L'expérience et la maturité de mes encadrants ont été de réels moteurs grâce auxquels j'ai surmonté les nombreuses difficultés rencontrées.

Je tiens également à remercier le professeur Paul-Antoine Bisgambiglia directeur de l'UMR 6134 « Sciences pour l'environnement », le professeur Antoine Aiello directeur de l'UMS 3564 « Stella Mare » ainsi que Mr Costa directeur de l'école doctorale 377 « Environnement et Société » de m'avoir accueilli au sein de leurs structures respectives. Le cadre dynamique et convivial de ces pôles de recherche m'a permis d'être immergé dans des problématiques passionnantes.

A mes collègues, membres du projet TIC, du département informatique ma gratitude pour votre aide et vos conseils est sans limites.

A mes collègues, doctorants et jeunes docteurs, nos relations d'amitié ont été de réels piliers pour moi. Tous les souvenirs partagés ensemble sont une source de joie immense.

A mes collègues, personnels administratif de l'Université de Corse, merci pour les fous-rires durant les « pauses café ».

Je souhaite également adresser mes remerciements à Mr Jean-François Rossi, directeur général de la société SITEC pour ses nombreux conseils au sujet de l'intégration de mes travaux dans le tissu économique.

Je profite également de ces quelques lignes pour remercier Mr David Barthe, Mme Apollonie Raffalli et Mr Gérard Monteleone de m'avoir mis le pied à l'étrier et donner le gout de l'informatique.

Enfin je remercie ma famille et mes amis de longue date pour leur patience et leur capacité d'écoute illimitée. Nous allons enfin avoir d'autres sujets de conversation que ma thèse. C'est promis.

Résumé

Nous vivons dans un monde où le besoin d'efficacité s'impose de plus en plus. Ce besoin s'exprime dans différents domaines, allant de l'industrie à la médecine en passant par la surveillance environnementale. Engendrées par cette demande, de nombreuses méthodes d'optimisation « modernes » également appelées « métaheuristiques » sont apparues ces quarante dernières années. Ces méthodes se basent sur des raisonnements probabilistes et aléatoires et permettent la résolution de problèmes pour lesquels les méthodes d'optimisation « classiques » également appelées « méthodes déterministes » ne permettent pas l'obtention de résultats dans des temps raisonnables. Victimes du succès de ces méthodes, leurs concepteurs doivent aujourd'hui plus que jamais répondre à de nombreuses problématiques qui restent en suspens : « Comment évaluer de manière fiable et rapide les solutions proposées ? », « Quelle(s) méthode(s) choisir pour le problème étudié ? », « Comment paramétrer la méthode utilisée ? », « Comment utiliser une même méthode sur différents problème sans avoir à la modifier ? ».

Pour répondre à ces différentes questions, nous avons développé un ensemble de concepts et outils. Ceux-ci ont été réalisés dans le cadre de la modélisation et la simulation de systèmes à évènements discrets avec le formalisme DEVS. Ce choix a été motivé par deux objectifs : permettre l'optimisation temporelle et spatiale de modèles DEVS existants et améliorer les performances du processus d'optimisation (qualité des solutions proposées, temps de calcul). La modélisation et la simulation de l'optimisation permettent de générer directement des propositions de paramètres sur les entrées du modèle à optimiser. Ce modèle, quant à lui, génère des résultats utiles à la progression de l'optimisation. Pour réaliser ce couplage entre optimisation et simulation, nous proposons l'intégration des méthodes d'optimisation sous la forme de modèles simulables et facilement interconnectables. Notre intégration se concentre donc sur la cohérence des échanges entre les modèles dédiés à l'optimisation et les modèles dédiés à la représentation du problème. Elle permet également l'arrêt anticipé de certaines simulations inutiles afin de réduire au maximum la durée de l'optimisation. La représentation des méthodes d'optimisation sous formes de modèles simulables apporte également un élément de réponse dans le choix et le paramétrage des algorithmes. Grace à l'usage de la simulation, différents algorithmes et paramètres peuvent être utilisés pour un même processus d'optimisation. Ces changements sont également influencés par les résultats observés et permettent une adaptation automatique de l'optimisation aux spécificités connues et/ou cachées du problème étudié ainsi qu'à ses différentes étapes de résolution.

L'architecture de modèle que nous proposons a été validée sur trois problèmes distincts : l'optimisation de paramètres pour des fonctions mathématiques, l'optimisation spatialisée d'un déploiement de réseau de capteurs sans fil, l'optimisation temporisée de traitements médicaux. La généricité de nos concepts et la modularité de nos modèles ont permis de mettre en avant la facilité d'utilisation de notre outil. Au niveau des performances, l'interruption de certaines simulations ainsi que dynamisme de l'optimisation ont permis l'obtention de solutions de qualité supérieure dans des temps inférieurs.

Abstract

In the world in which we live the efficient needs are increasing in various fields like industry medecine and environnemtale monitoring. To meet this needs, many optimization methods nammed « metaheuristics » have been created over the last forty years. They are based on probabilistic and random reasoning and allow user to solve problems for which conventional methods can not be used in acceptable computing times. Victim of their methods succes, the developers of the methods have to answer to several questions : « How can the fitness of solutions be assessed ? », « How to use the same method for several projects without change the code? », « What method will we choose for a specific problem ? », « How to parametrize algorithms ? ».

To deal with this problem, we have developed a set of concepts and tools. They have been developed in the context of modeling and simulation of discrete event systems with DEVS formalism. The aims pursued are : allow temporized and spacialized optimization of existing DEVS models, improve the optimization process efficiency (quality of solutions, computing time). Modeling and simulation are used to propose parameters toward the input of problem to optimize. This one generate results used to improve the next proposed solutions. In order to combine optimization and simulation, we propose to represent the optimization method as models which can be easily interconnected and simulated. We focus on consistency of exchanges between optimization models and problem models. Our approach allows early stopping of useless simulations and reduce the computing time as a result. Modeling optimization methods in DEVS formalism also allows to autimatically choose the optimization algorithm and its parameters. Various algorithms and parameters can be used for the same problem during optimization process at different steps. This changes are influenced by collected results of problem simulation. They lead on a self adaptation to the visible or/and hidden features of the studied problem.

Our models architecture has been tested on three different problems : parametric optimization of mathematical functions, spacialized optimization of a sensor network deployment, temporized optimization of a medical treatment. Genericity of our concepts and scalability of our models underline the usabily of proposed tool. Concerning performance, simulation breaks and dynamic optimization have obtained higher quality solutions in a short time.

Sommaire

Chapitre 1
Introduction générale

« Rien n'est trop difficile pour la jeunesse »

Socrate

L'humanité doit aujourd'hui plus que jamais faire face à de nombreux défis dont la complexité semble dépassée nos capacités de raisonnement. Nous pouvons citer quelques exemples dans différents domaines. Au niveau environnemental, les ressources doivent être exploitées de manière plus efficace et plus durable. En effet les modes de productions actuels sont générateurs d'importants gaspillages et de pollutions ayant pour conséquences des coûts économiques, écologiques et sanitaires non négligeables. La raréfaction des matières premières et des sources d'énergie nous contraignent à mettre en œuvre rapidement de profonds changements. Au niveau des transports, la structure des réseaux doit également être réorganisée afin de satisfaire aux impératifs de qualité de service attendus par les utilisateurs tout en respectant certaines contraintes telles que le coût et la sécurité des passagers. Au niveau médical, les récentes découvertes nécessitent des outils d'analyse performants et fiables permettant aux praticiens de disposer d'informations structurées et cohérentes. Nous sommes donc à l'aube de changements majeurs marquant l'ère du « développement durable ».

Tous ces changements reposent sur un unique point de départ : « la décision ». Une décision repose sur différents éléments. Le premier est une bonne connaissance, visibilité du problème pour lequel une ou plusieurs décisions se doivent d'être prises. Le second est l'élaboration d'un raisonnement cohérent reposant sur des démonstrations et des preuves. Enfin le troisième est la méthode d'élaboration et de structuration de ce raisonnement. Les décisions que les dirigeants actuels sont amenés à prendre dans des délais restreints impliquent de nombreuses entités interdépendantes difficilement analysables. Pourtant « l'efficience » des choix opérés doit être maximale et doit dans un même temps répondre à de très nombreux critères interdépendants. De tout temps, la solution que l'humanité a trouvée pour compenser ses faiblesses est l'élaboration d'outils venant étendre ses capacités. Dans le cadre de décisions complexes, un des outils le plus utilisé se nomme « les systèmes d'aide à la décision ». Parmi ceux-ci, certains ont l'objectif ambitieux de venir étendre l'intelligence humaine par la création d'une nouvelle substance impalpable : «l'intelligence artificielle ». Bien que de nombreux systèmes d'aide à la décision soient d'ores et déjà présents sur le marché, ceux-ci se limitent généralement à un seul domaine d'application et sont difficilement accessibles. Leurs coûts peuvent être prohibitifs et leur maitrise nécessite de nombreuses heures de formations.

A partir du constat que nous venons de dresser, nous souhaitons proposer un ensemble de concepts regroupés à l'intérieur d'un outil ouvert et extensible d'optimisation par simulation intégrant des techniques d'intelligence artificielle novatrices et performantes permettant de proposer des solutions optimales aux problèmes posés. La représentation du problème sera réalisée par des modèles et des simulations informatiques représentant le système à optimiser. En effet les différents outils inhérents à la modélisation et la simulation sont de nos jours un support d'étude incontournable pour l'ensemble des domaines scientifiques et industriels. Ce succès s'explique par le fait que l'innovation, l'analyse et l'apprentissage se fondent sur la notion d'expérimentation (Shannon, 1998; Ingalls, 2008). L'usage de la modélisation et de la simulation permet la synthèse et l'acquisition de connaissances par la compréhension, l'analyse et la prédiction de phénomènes et processus hétérogènes. Cette virtualisation du monde réel offre des avantages incontestables. Elle permet par exemple des gains de temps considérables pour certaines expériences nécessitant des durées importantes dans le monde réel (e. g. évaluation de l'érosion du littoral). Elle permet aussi de réduire significativement le coût d'expérimentation (e. g. construction d'un prototype d'avion). Elle offre également la possibilité d'effectuer des expérimentations difficilement réalisables dans le monde réel (e. g. simulation d'un incident nucléaire, évacuation d'une ville). La résolution du problème sera quant à elle effectuée grâce à des techniques avancées d'optimisation fréquemment utilisées dans les différents systèmes d'aide à la décision. L'optimisation consiste à trouver la réponse « optimale » pour différents types de problèmes complexes : NP-complets, NP-durs. Elle permet de répondre au besoin d'efficience que

nous avons abordé précédemment. Tout comme la modélisation et la simulation, l'optimisation trouve ses racines dans les sciences mathématiques et informatiques. Contrairement à la modélisation et la simulation sa création est très ancienne et le nombre de méthodes d'optimisation est considérable. Certaines sont propres à une catégorie spécifique de problèmes alors que d'autres sont profondément génériques. Cette multiplicité des méthodes proposées ne présente pas que des avantages. En effet ce nombre ajoute un niveau de difficulté. La résolution du problème n'est plus seulement d'optimiser la solution d'un problème mais aussi de choisir la méthode adéquate en fonction des caractéristiques du problème et des priorités exprimées par le décideur (e. g. temps, fiabilité, rapidité de calcul).

Notre approche qui consiste à coupler la simulation et l'optimisation au sein d'une architecture de modèles DEVS est génératrice d'une synergie nouvelle. En effet, nous pensons que la création de métaheuristiques simulées permet de répondre à différentes problématiques actuellement non résolues de généricité et de performances. Premièrement, notre intégration permet l'optimisation paramétrique de modèles DEVS réalisés précédemment, en dehors de tout contexte d'optimisation. Pour cela, nous proposons des modèles d'optimisation génériques, indépendants et interconnectable avec de nombreux modèles respectant les spécifications du formalisme DEVS et ayant comme entrées des variables décisionnelles et comme sorties les conséquences de ces mêmes décisions. Les concepts que nous développons permettent également d'automatiser le choix et le paramétrage des algorithmes d'optimisation en profitant du dynamisme offert par la simulation. Ils automatisent de manière intelligente le choix d'une méthode d'optimisation correctement paramétrée et adaptée à la nature du problème. Ce dynamisme se matérialise par différents évènements déclenchés grâce à une analyse de l'évolution de la qualité des solutions. Cette analyse itérative permet de limiter la durée de certaines simulations dont les résultats générés ne sont pas inutiles au processus d'optimisation. Ces différents aspects de notre travail de recherche permettent donc : de mutualiser les modèles d'optimisation pour différents problèmes modélisés, de proposer des solutions de meilleures qualités, dans des temps moins importants.

Afin de présenter au mieux le travail de recherche réalisé, ce manuscrit est organisé en six chapitres. Le chapitre 2 présente une vision d'ensemble des différents concepts de l'optimisation. Un focus est réalisé vers les méthodes d'optimisation probabilistes inspirées par l'observation de phénomènes naturels (Yang, 2008; Chiong, 2009) que nous avons choisi d'utiliser pour notre outil. Le chapitre 3 présente les concepts de la modélisation et la simulation ainsi que les principaux formalismes de systèmes à évènements discrets actuellement présents dans la littérature. Nous nous intéressons plus spécifiquement au formalisme de modélisation et simulation de systèmes à évènements discrets DEVS (Discret Event SYstem Specification) (Zeigler et al., 2000). DEVS repose sur une théorie formelle de haut niveau qui lui offre la possibilité d'englober de nombreux autres formalismes de modélisation et de simulation. Cet aspect lui permet de ne pas se restreindre à des applications de natures spécifiques. Dans ce chapitre, les recherches concernant l'optimisation via des simulations DEVS sont présentées et analysées. Le chapitre 4 présente notre architecture de modèles. Dans ce chapitre nous présenterons les apports de celle-ci par rapport aux travaux présents dans la littérature. Les deux concepts novateurs de notre architecture sont « l'évaluation externalisée » et « l'optimisation intelligente ». « L'évaluation externalisée » permet de réaliser des optimisations paramétriques sur l'ensemble des systèmes modélisés dans le formalisme DEVS sans avoir à les modifier. «L'optimisation intelligente » permet d'automatiser le choix et le paramétrage de l'optimisation en vue de réduire le temps d'exécution de l'optimisation et d'améliorer la qualité des solutions proposées. Ce chapitre présente également la conception UML et la formalisation DEVS des différents modèles qui composent notre architecture « orientée modèle ».Le chapitre 5 illustre les différents résultats obtenus par trois applications distinctes. La première concerne l'optimisation

paramétrique de fonctions mathématiques non-linéaires. La seconde consiste en l'optimisation d'un déploiement d'un réseau de capteurs sans-fils dans l'espace. Et enfin, la troisième a comme objectif l'optimisation de traitements médicaux dans le temps. Enfin le chapitre 6 présente le bilan critique de nos travaux et énumère les différentes perspectives qui s'offrent à nous.

Chapitre 2
Optimisation de problèmes complexes

« Un problème sans solution est un problème mal posé »

Albert Einstein

Nos travaux de recherche concernent l'intégration de méthodes d'optimisation avec des modèles simulés. Dans cette optique, ce chapitre introduit les concepts fondamentaux de l'optimisation. Pour cela, dans un premier temps, nous donnerons la définition générale d'un problème d'optimisation et énumèrerons les différentes caractéristiques (nature et complexité) des problèmes pouvant être rencontrés. Nous mettrons également en avant la principale difficulté de l'optimisation à savoir, la distinction entre « minimum locaux » et « minimum globaux ». Nous présenterons ensuite les avantages et les inconvénients des « méthodes d'optimisation déterministes » et des « méthodes d'optimisation probabilistes ».

Dans un second temps, nous concentrerons notre analyse sur les « méthodes probabilistes» également appelées « méthodes inspirées par la nature », « méthodes modernes » (Lee and El-Sharkawi, 2008; Rothlauf, 2011) ou encore « métaheuristiques ». Nous présenterons les concepts communs de ces méthodes ainsi que les différentes extensions disponibles (parallélisation, multicritères, hyperheuristiques). Nous insisterons également sur les problématiques de choix et de paramétrage ces algorithmes qui restent aujourd'hui leur principale difficulté.

Enfin, à partir d'une classification basée sur le nombre de solutions utilisées et sur le domaine d'inspiration des méthodes nous présenterons les principaux algorithmes. Le fonctionnement de chacun d'eux sera détaillé par des exemples pédagogiques que nous commenterons.

2.1 Concepts généraux

2.1.1. Le concept de problème

Un problème d'optimisation peut être caractérisé de la manière suivante (Bailleux, 1996) :

$$\Omega = (D,C,f,E \prec)$$

- D : est un ensemble dont les éléments sont appelés les instances I du problème
- C : est un ensemble dont les éléments sont appelés les configurations du problème
- E : est un ensemble représentant l'espace d'évaluation
- f : est une fonction partielle de D x C dans E représentant la « fonction objectif »
- $\prec$: est une relation d'ordre total sur E

L'ensemble des configurations C_I d'une instance I de Ω peut être définie comme suit :

$I \in D$, $x \in C_I$, tel que f(I,x) existe

L'ensemble des solutions optimales C^{opt}_I d'une instance I de Ω peut être défini comme suit :

$I \in D$, $x \in C_I$ une solution optimale, telle que pour tout $y \in C_I$ $f(I,x) \prec f(I,y)$

La valeur optimale E^{opt} d'une évaluation d'une instance I de Ω peut être définie comme suit :

Soit $x \in C_I$ une solution optimale, $f^{opt} = f(I,x)$

2.1.2. La classification des problèmes

De nombreuses catégories de problèmes sont présentes dans la littérature. Leur classification se base généralement sur l'utilisation de deux concepts : « la machine de Turing déterministe » et la « machine de Turing non déterministe ». Un exemple de « Machine de Turing déterministe » est présenté sur le Tableau 1. Sa description est la suivante :

$$MTD = (Q, \Gamma, \sqcup, \Sigma, i, \delta, A)$$

- Q : un ensemble fini d'états
- Γ : un ensemble fini de symboles (alphabet)
- $\sqcup \in \Gamma$: symbole blanc
- Σ : un alphabet des symboles d'entrée tel que $\Sigma : \Gamma \setminus \{\sqcup\}$
- $i \in Q$: état initial
- δ : est la fonction de transition $Q \times \Sigma \longrightarrow Q \times \Gamma \times \{\text{gauche, droite}\}$
- A : un ensemble d'état finaux (états acceptants)

Etat actuel	Lecture	Ecriture	Déplacement	Etat future
Q_1	Σ_1	Γ_1	droite	Q_2
Q_1	Σ_2	Γ_2	Gauche	Q_1
Q_2	Σ_1	Γ_2	Gauche	Q_2
Q_2	Σ_2	Γ_2	Droite	Q_2
Q_3	Σ_1	Γ_1	Droite	Q_2
Q_3	Σ_2	Γ_1	Droite	Q_1

Tableau 1 Exemple de machine de Turing déterministe

Un exemple de « Machine de Turing non-déterministe » est présenté sur le Tableau 2. Contrairement à une « Machine de Turing déterministe » plusieurs transitions d'états sont possibles pour un même symbole lu dans un état courant. Cet aspect est décrit dans la définition suivante :

$$MTND = (Q, \Gamma, \sqcup, \Sigma, i, \delta, A)$$

- Q : un ensemble fini d'états
- Γ : un ensemble fini de symboles (alphabet)
- $\sqcup \in \Gamma$: symbole blanc
- Σ : un alphabet des symboles d'entrée tel que $\Sigma : \Gamma \setminus \{\sqcup\}$
- $i \in Q$: état initial
- $\delta \subseteq (Q \setminus A \times \Sigma) \times (Q \times \Sigma \times \{droite, gauche\})$
- $A \subseteq Q$: un ensemble d'états d'acceptation

Etat actuel	Lecture	Ecriture	Déplacement	Etat future
Q_1	Σ_1	Γ_1	droite	Q_2
Q_1	Σ_2	Γ_2	Gauche	Q_1
Q1	Σ_2	Γ_1	Droite	Q_2
Q_2	Σ_1	Γ_2	Gauche	Q_2
Q_2	Σ_2	Γ_2	Droite	Q_2
Q_3	Σ_1	Γ_1	Droite	Q_2
Q_3	Σ_2	Γ_1	Droite	Q_1
Q_3	Σ_2	Γ_2	Droite	Q_1

Tableau 2 Exemple de machine de Turing non-déterministe

A partir des deux machines que nous venons de décrire, différentes classes de problèmes peuvent être définies (Garey and Johnson, 1979) comme cela est visible sur la Figure 1.

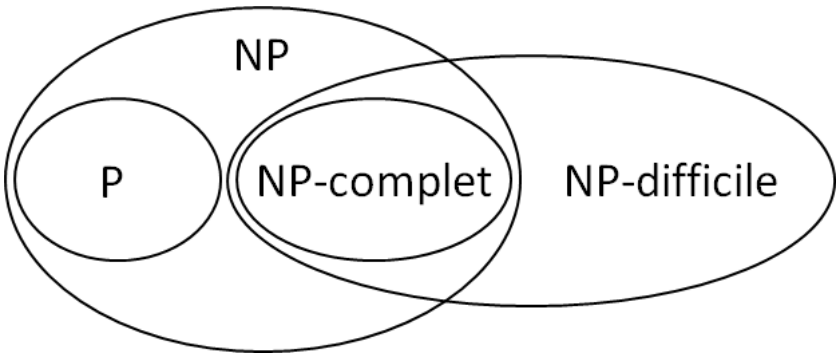


Figure 1 Principales classes de problèmes d'optimisation

La classe P regroupe les problèmes pouvant être résolus par une machine de Turing déterministe dans des temps polynomiaux. En d'autres termes, cette classe rassemble les problèmes pour lesquels il existe au moins un algorithme déterministe de résolution.

La classe NP quant à elle regroupe les problèmes qui peuvent être résolus par une machine de Turing non-déterministe. En supposant que la machine de Turing non déterministe effectue les bons choix lorsque plusieurs transitions sont possibles. Le temps pour « trouver une solution » est proportionnel à la taille de l'instance du problème. Le temps pour « vérifier une solution » de cette classe est quant à lui polynomial. Un problème A est complet pour la classe NP si celui-ci appartient à la classe NP et que tout problème B appartenant à la classe NP peut être réduit à A. Les problèmes NP-complets sont forcément NP-difficiles. Cette classification reste aujourd'hui à démontrer

formellement. Actuellement, elle se base sur la conjecture que P ≠ NP et implique qu'il n'existe aucun algorithme déterministe pouvant résoudre un problème NP-complet dans un temps polynomial. Des exemples de problèmes, leurs complexités ainsi que leurs classes sont présentés dans le Tableau 3.

Exemple	Fonction	Classe
Conversion de devise	O(1)	P
Recherche dichotomique	O(log n)	P
Recherche itérative	O(n)	P
Tri par tas	O(n * log n)	P
Somme d'un tableau de deux dimensions	$O(n^2)$	P
Problème du sac à dos	$O(n^k)$	NP-complet
Problème du voyageur de commerce (cycle hamiltonien)	O(n !)	NP-complet
Coloration de graphes	$O(n^k)$	NP-complet

Tableau 3 Complexité des problèmes

Les nombreux problèmes d'optimisation peuvent également être de natures hétérogènes (Yang and Koziel, 2011). Les différentes natures d'un problème sont listées dans le Tableau 4.

Caractéristique	Caractéristique opposée
Mono-objectif	Multi-objectif
Linéaire	Non-linéaire
Convexe	Non-convexe
Statique	Dynamique
Continu	Discret
Unidimensionnelle	Multidimensionnelle
Avec contraintes	Sans contraintes
Types homogènes	Types hétérogènes
Déterministe	Stochastique

Tableau 4 Natures des problèmes d'optimisation

2.1.3. Le concept d'optimisation

L'optimisation est omniprésente dans le monde dans lequel nous vivons. Le besoin d'optimisation est le fruit de la confrontation de deux entités : la prédiction de résultats et un objectif. La définition du problème et des finalités recherchées sont donc indispensables à la réussite de tout processus d'optimisation.

Dans sa définition formelle, l'optimisation est un processus de recherche visant à trouver le vecteur de variables permettant de minimiser (ou maximiser) la sortie d'une « fonction d'évaluation » aussi appelée « fonction objectif » ou « fonction de coût » en vue de trouver « l'optimum » (Yang and Koziel, 2011). Elle peut être caractérisée de la manière suivante :

$$\textit{Minimiser } f_i(s),\ (i=1, 2,3,.., l),\ s \in S$$

$$\textit{En fonction des contraintes}$$

$$h_i(s),\ (i=1, 2,3,.., m)$$

$$g_i(s),\ (i=1, 2,3,.., n)$$

- $s \in S$ désigne le vecteur des différentes variables du problème
- S désigne l'ensemble des solutions réalisables
- f_i désigne les différentes fonctions de coûts dont le nombre est exprimé par l. Pour chaque $s \in S$, f_i génère une évaluation $e \in E$ où e désigne le « paysage de fitness ».
- h_i désigne les différentes contraintes d'égalité du problème dont le nombre est exprimé par m
- g_i désigne les différentes contraintes d'inégalité du problème dont le nombre est exprimé par n
- l désigne le nombre de fonctions de coût
- m désigne le nombre de fonctions représentant les contraintes d'égalité
- n désigne le nombre de fonction représentant les contraintes d'inégalité

L'optimum est la combinaison de variables représentant la meilleure solution ou l'une des meilleures solutions parmi l'ensemble des combinaisons de valeurs possibles dans « l'espace de recherche ».

Si l'objectif du problème est une minimisation, l'optimum sera la combinaison de variables ayant comme sortie le minimum globale de la fonction d'évaluation. En considérant que S désigne l'ensemble des solutions possibles d'un espace de décision et que $s \in S$, une solution s^* sera considérée comme un minimum globale si :

$$f(s^*) \leq f(s)\ \forall\ s \in S$$

Si l'objectif du problème est une maximisation l'optimum sera la combinaison de variables permettant d'obtenir le maximum global de la fonction d'évaluation. En considérant que S désigne l'ensemble des solutions possibles d'un espace de décision et que $s \in S$, une solution s^* sera considérée comme un maximum globale si :

$$f(s^*) \geq f(s)\ \forall\ s \in S$$

La Figure 2 illustre ces concepts à travers une projection en deux dimensions de la fonction multidimensionnelle « Alpine ». Le « paysage de fitness » visible peut être défini de la manière suivante :

Paysage de fitness = (S,f,V)

- S désigne l'ensemble des solutions
- f : S → R la fonction à optimiser
- V désigne l'ensemble des relations de voisinage

Sur cette même figure, nous pouvons observer deux groupes de valeurs : les minimums locaux et les maximums locaux. Ceux-ci représentent des solutions sur une portion limitée de l'espace de recherche et sont la principale difficulté de nombreux processus d'optimisation complexes.

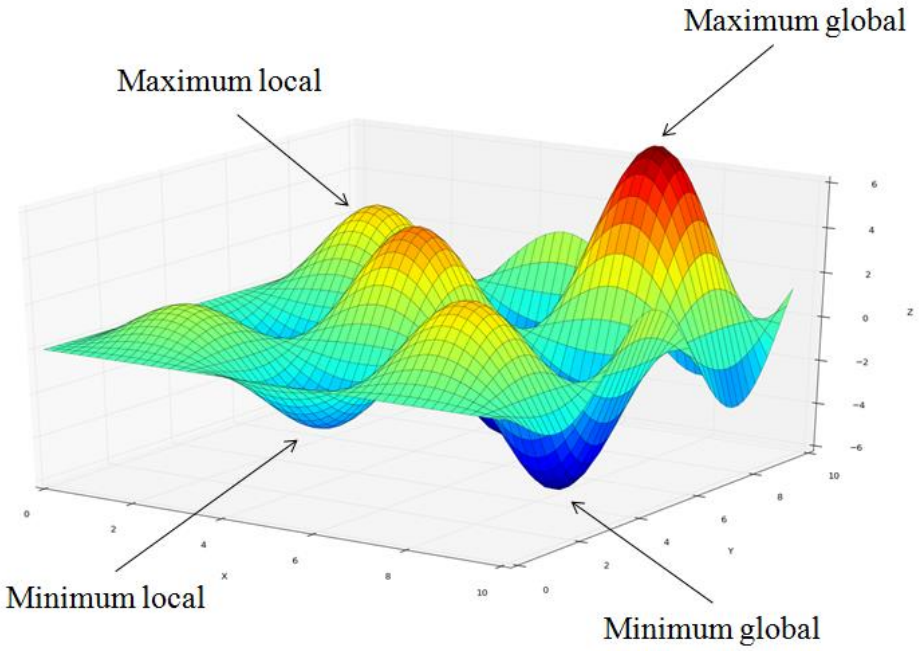


Figure 2 Paysage de fitness de la fonction « Alpine »

2.1.4. Classification des méthodes d'optimisation

Une multitude de méthodes d'optimisation existe dans la littérature (Rao, 2009) et leur nombre croît régulièrement. Cette diversité s'explique par le nombre considérable des domaines utilisant des processus d'optimisations. Cependant, comme nous pouvons le voir sur la Figure 3, deux catégories de méthodes peuvent être isolées : les méthodes exactes et les méthodes approximatives (Weise, 2008).

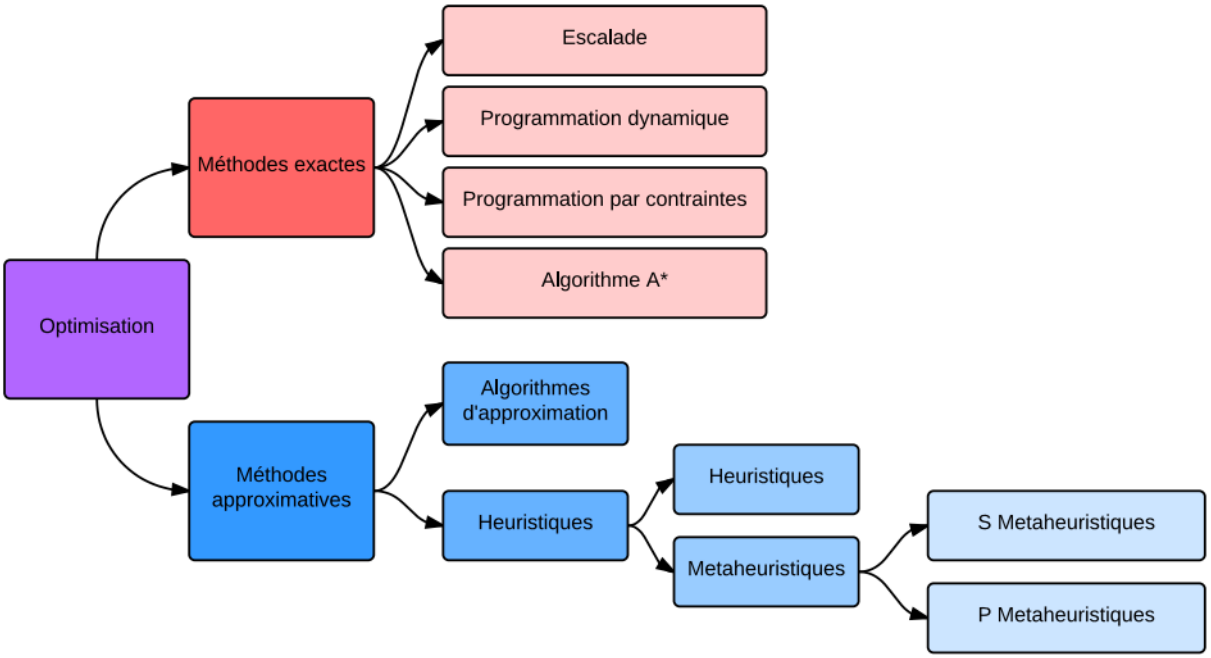


Figure 3 Classification des méthodes

Les méthodes exactes se basent sur des raisonnements mathématiques démontrés. Elles peuvent être utilisées avec succès pour des problèmes de nature linéaire et continue pouvant être modélisé aisément. Parmi ces méthodes, nous pouvons citer les méthodes de programmation linéaire et les méthodes de descente de gradient qui sont les principales. Un exemple de méthode de programmation linéaire : l'algorithme du « simplex »(Dantzig et al., 1955) est présenté sur la Figure 4. Dans cet exemple, on peut voir que l'espace de recherche est progressivement réduit à partir des contraintes énoncées. Une fois la phase de restriction terminée, la valeur sélectionnée est celle qui en x renvoie la plus grande valeur de y car nous sommes dans le cadre d'un objectif de maximisation.

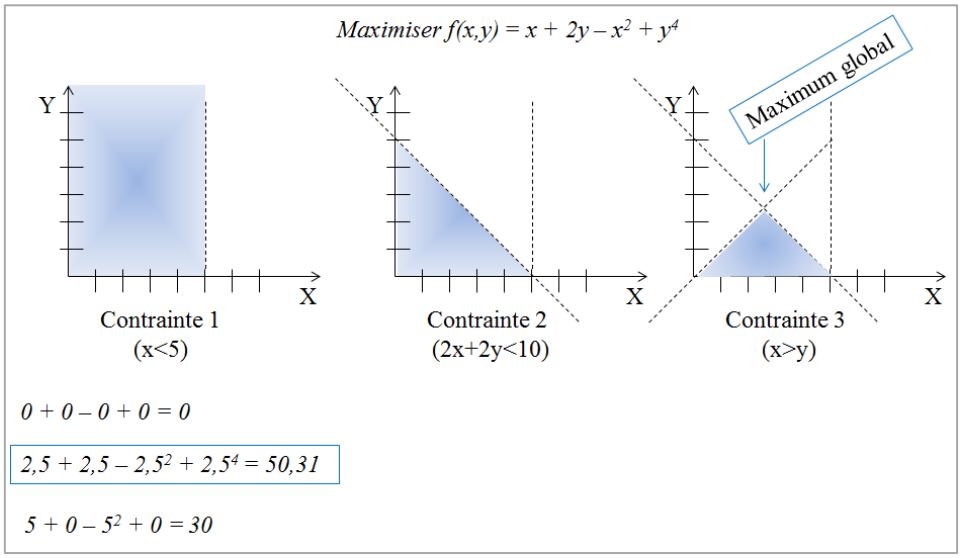


Figure 4 Exemple d'execution de l'algorithme du Simplex

Ces méthodes assurent de trouver l'optimum de la fonction et sont particulièrement efficaces sur des problèmes de recherche locale de petite taille et ne nécessitent aucun paramétrage. Cependant celles-ci présentent certains inconvénients. Elles fonctionnent difficilement sur des problèmes possédant des aspects stochastiques, discrets, dynamiques ou encore impliquant différents types de

variables. D'autre part, leur temps de d'exécution sur des problèmes de type NP-complet de grande taille est exponentiel. Enfin, leur adaptation à un problème spécifique peut être fastidieuse et chronophage.

La seconde catégorie regroupe les méthodes approximatives (également appelées méthodes « probabilistes »). Celles-ci se basent sur des algorithmes itératifs. Parmi ces méthodes, les algorithmes génétiques (Holland, 1992), le recuit simulé (Kirkpatrick et al., 1983) et les essaims particules (Kennedy and Eberhart, 1995) figurent parmi les exemples les plus largement utilisés.

Le fonctionnement probabiliste (basé en partie sur des raisonnements aléatoires) de ces méthodes ne garantit pas de parvenir à l'optimum à l'issue du processus d'optimisation. Cependant, ces méthodes permettent d'optimiser des problèmes complexes appartenant à la classe NP-complets dans des temps raisonnables. Elles sont particulièrement adaptées aux problèmes dont la nature peut être discrète, dynamique, stochastique ou encore non linéaire. De plus, de par leur inspiration naturelle celles-ci sont facilement compréhensibles et adaptables à tout type de problème.

Leurs concepts communs, leurs extensions, ainsi que les principaux algorithmes appartenant à cette catégorie de méthodes d'optimisation, sont présentés plus en détails dans la partie suivante.

2.2 Optimisation par les métaheuristiques

2.2.1. Définition générale

Les métaheuristiques regroupent différents algorithmes inspirés par la nature (biologie, physique, sociologie, etc.) dont l'objectif est la résolution de problèmes complexes (Gendreau and Potvin, 2010; Glover and Kochenberger, 2003; Luke, 2013; Talbi, 2009; Yang, 2008). À la différence des heuristiques qui sont propres à un problème spécifique, les métaheuristiques sont indépendantes du problème étudié. Elles se basent sur des processus itératifs et stochastiques permettant la convergence de la ou des solutions vers l'optimum comme cela est présenté sur la Figure 5 et la Figure 6. Sur ces deux figures, nous pouvons voir que des solutions aléatoires sont proposées lors de la première itération. Au fil des itérations, les solutions les plus mauvaises vont être remplacées par des solutions de meilleures qualités, plus proches de l'objectif recherché. La répétition de ce processus pourra permettre d'atteindre l'optimum recherché. Sur la Figure 5, les différentes couleurs de flèches représentent l'itération durant laquelle la solution a été créée (noir pour l'itération 1, bleu pour l'itération 2, rouge pour l'itération 3). Sur la Figure 6, les zones bleues sont les zones dont les valeurs renvoyées par la fonction d'évaluation sont minimales alors que les zones rouges représentent les valeurs maximales.

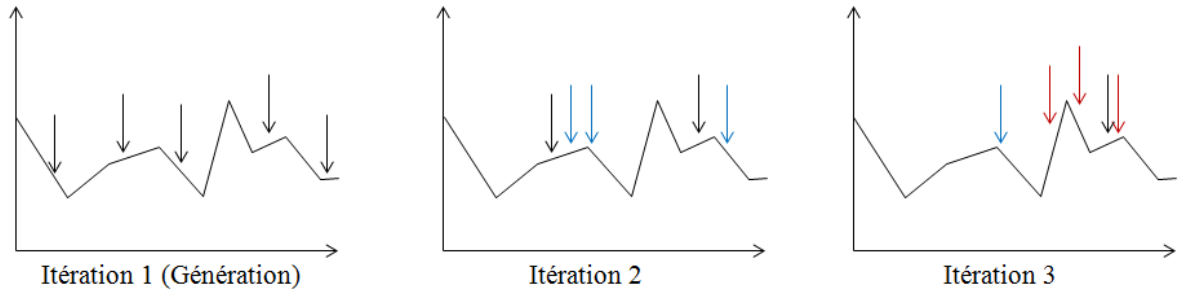


Figure 5 Exemple de maximisation (courbe 2D)

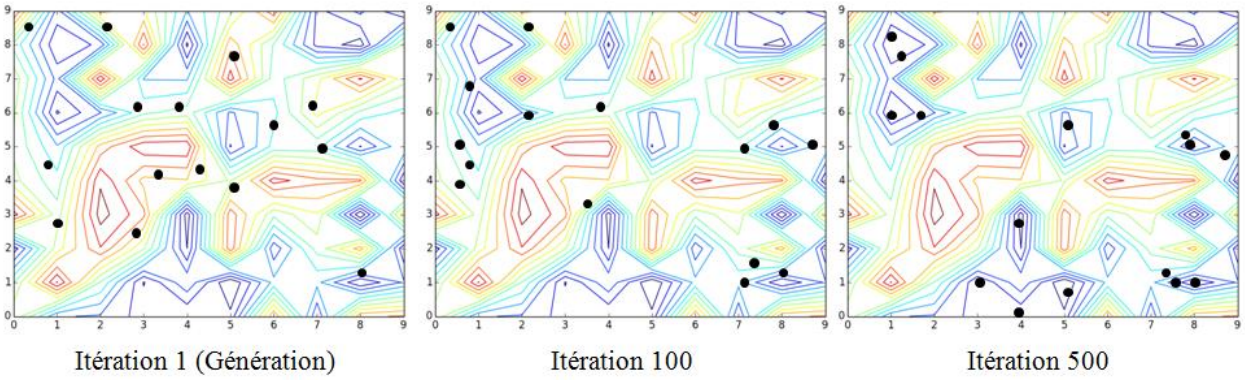


Figure 6 Exemple de minimisation (contours courbe 3D)

L'exécution des métaheuristiques s'articule autour de deux concepts centraux : « la diversification » et « l'intensification ». La « diversification » permet d'explorer les différentes zones de l'espace de recherche pouvant abriter des solutions de qualité. « L'intensification » quant à elle permet d'approfondir et d'affiner les solutions proposées. La part de chacun de ces deux concepts est ajustable en ajustant différents paramètres propres à chaque algorithme et dont le choix dépend de la nature du problème étudié. La configuration idéale doit assurer un juste équilibre entre l'intensification et la diversification. En effet, une intensification trop forte augmentera le risque de bloquer le processus à l'intérieur d'un minimum local alors qu'une diversification augmentera de manière considérable le temps de convergence vers l'optimum.

Utilisées dans différents domaines (médical, militaire, écologique, chimique, économique, architectural et de gestion des risques), la diffusion de ces algorithmes tant sur le plan scientifique qu'industriel est incontestable. Ce succès croissant s'explique par de nombreux progrès qui ont permis l'explosion de la puissance de calcul et la chute des coûts des équipements depuis près d'un demi-siècle.

Il existe de nombreuses classifications des métaheuristiques. La plus utilisée se base sur le nombre de solutions gérées par l'algorithme en distinguant : les « S-Métaheuristique » qui utilisent une seule solution durant leur exécution et les « P-Métaheuristiques » qui elles utilisent un ensemble de solutions.

2.2.2. Représentation des solutions

Pour un problème donné, différentes représentations des solutions peuvent être utilisées. Cependant certaines sont meilleures que d'autres. Les différentes alternatives doivent donc être étudiées afin de choisir celle qui permettra une représentation précise des solutions tout en assurant une convergence rapide de celles-ci vers l'optimum.

En premier lieu, la représentation doit permettre de couvrir l'ensemble de l'espace de recherche. On parle alors de « complétude ». En effet, aucune solution potentielle ne doit être ignorée.

Elle doit ensuite assurer un voisinage cohérent entre les différentes solutions comme cela est démontré dans l'exemple sur la Figure 7. Sur cet exemple, deux représentations du voisinage sont présentes. Leur objectif est de représenter une grille et les liens entre les différentes cellules de cette grille. La représentation de gauche est correcte car celle-ci permet de représenter l'ensemble de cellules voisines de la cellule (2,2). Ce n'est pas le cas de la représentation de droite où seule une partie des cellules voisines de (2,2) est correctement représentée.

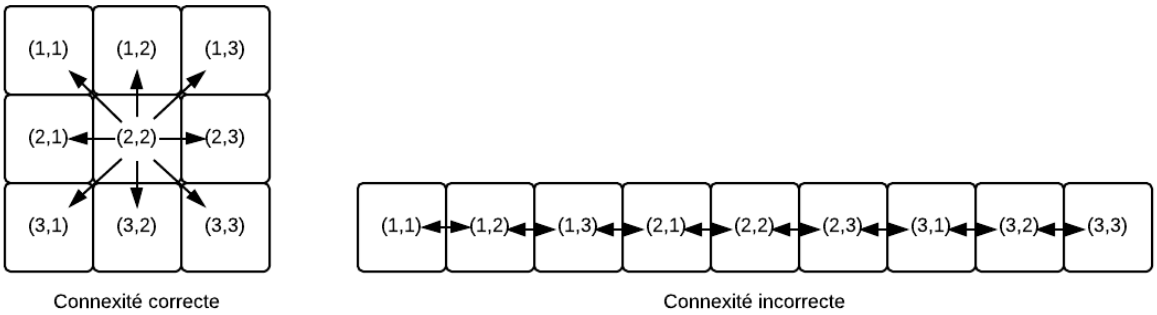


Figure 7 Exemples de connexité

La représentation des solutions doit aussi permettre aux différents opérateurs de la métaheuristique d'accéder au contenu de la solution. Deux approches sont alors possibles comme cela est présenté sur la Figure 8. L'approche « directe » qui représente la solution avec sa structure et son type final. Cette approche est rapide mais nécessite une adaptation des opérateurs de la métaheuristique en fonction des caractéristiques des solutions. L'approche « encodée » ou « indirecte » qui elle, représente la solution en utilisant un type intermédiaire standard (e. g. liste binaire). Cette approche nécessite une phase de traduction de la ou des solutions mais permet cependant d'utiliser des opérateurs génériques dans l'algorithme définissant la métaheuristique.

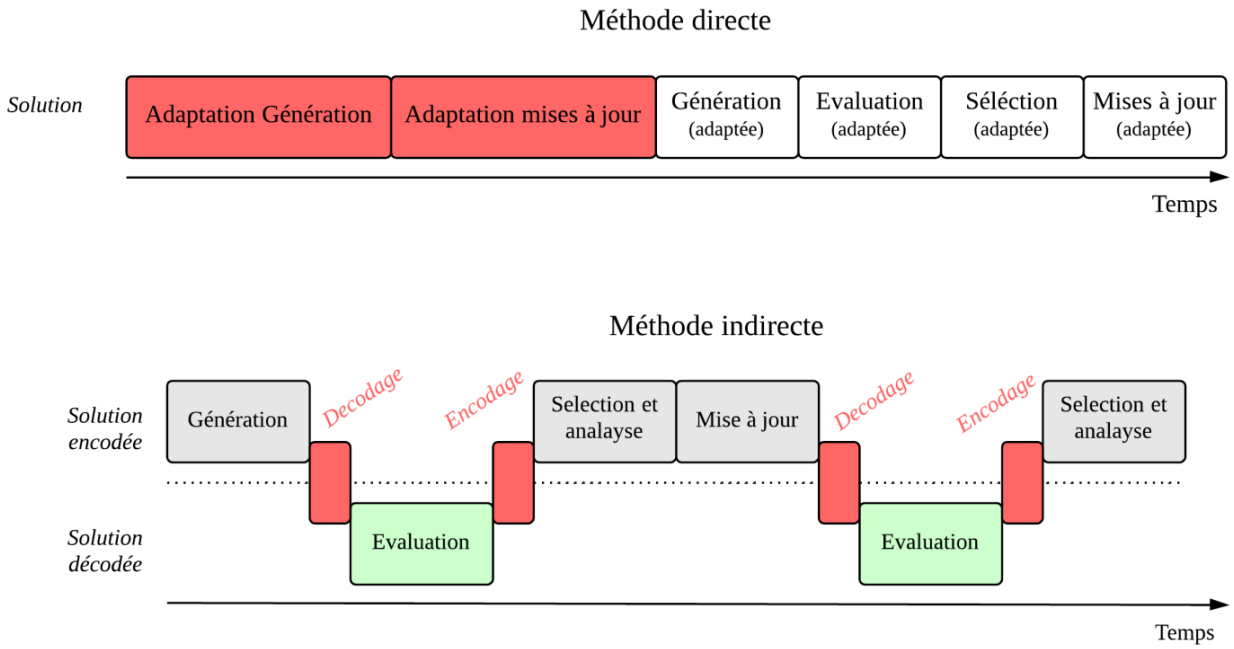


Figure 8 Codage direct et codage indirect

La représentation des solutions est décrite par deux éléments : la structure des données et leur type. Au niveau de la structuration, deux groupes peuvent être isolés. Le premier groupe rassemble les structures dites « linéaires » qui utilisent les structures de données suivantes : vecteurs, listes, règles, matrices. Le second groupe contient les structures dites « non-linéaires » qui utilisent les structures de données suivantes : arbres et graphes. À l'intérieur de ces structures peuvent être présentes différentes sortes de variables de types primitifs (entier, flottant, caractère, binaire ou mixte) de manière homogène mais aussi hétérogène. Bien qu'une nette préférence pour le type binaire soit visible dans la littérature, dans certains cas celui-ci ne représente pas le type le mieux adapté.

Une fois le format de la solution potentielle ou des solutions potentielles choisi les différentes étapes communes à chaque métaheuristique peuvent être exécutées.

2.2.3. Déroulement des métaheuristiques

Bien que de très nombreux algorithmes soient présents dans la littérature, ceux-ci reposent tous sur un socle commun, décomposable en quatre catégories d'actions : des générations aléatoires, des évaluations, des analyses, des mises à jour. Ces différentes actions s'ordonnancent comme cela est visible sur la Figure 9. Celles-ci sont présentées dans les sous parties suivantes.

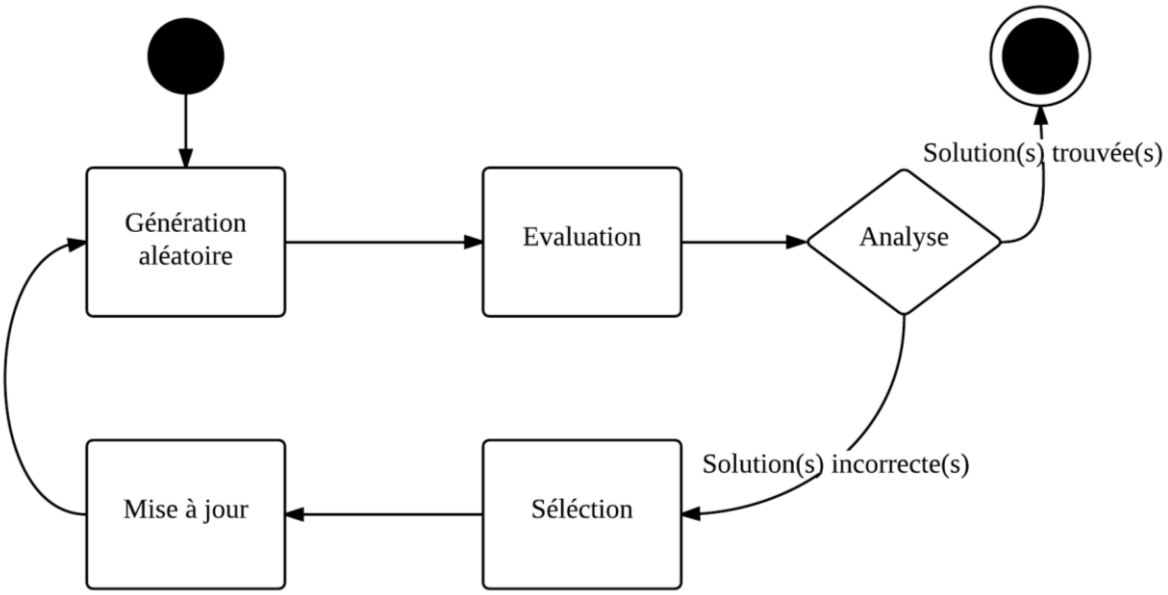


Figure 9 Les étapes d'exécution d'une métaheuristique

2.2.3.1. Générations aléatoires

La génération aléatoire est le point de départ de chaque métaheuristique. Cette étape a comme objectif de générer de manière aléatoire puis de stocker en mémoire une ou plusieurs solutions. La génération aléatoire ne doit en aucun cas restreindre l'espace de recherche mais au contraire être la plus diversifiée possible afin de ne pas ignorer des solutions répondant aux objectifs énoncés. Elle se base sur deux éléments : un générateur de nombres aléatoires et une représentation commune des solutions. La génération aléatoire doit être cohérente avec le format de représentation des solutions.

Le choix d'un générateur aléatoire est une étape cruciale trop souvent négligée. En effet, la diversité des nombres générés par celui-ci a un impact non négligeable sur l'efficacité de l'algorithme choisi. Cependant, seuls des nombres pseudo-aléatoires peuvent actuellement être générés par la majorité des équipements informatiques. Il convient donc de choisir une méthode de génération de nombres aléatoires performante. Deux méthodes sont principalement utilisées de nos jours pour générer des nombres pseudo-aléatoires : l'algorithme de Mersenne Twister et la méthode de Fibonacci. Leur succès s'explique par une prédiction difficile de leur sorties et par la diversité des nombres générés essentielle au bon déroulement des métaheuristiques.

Une fois la phase de génération aléatoire terminée, l'exécution d'une métaheuristique se dirige vers la phase d'évaluation.

2.2.3.2. Evaluation

Cette phase se base sur une « fonction d'évaluation » de la définition formelle d'un processus d'optimisation. Elle permet de connaitre pour chacune des solutions potentielles, la qualité de la réponse proposée par rapport au problème étudié. Il est important de préciser que pour un même problème, différentes fonctions d'évaluation peuvent être disponibles. Un choix doit alors être fait en fonction des priorités qui pourront être : la précision de l'évaluation, la durée de son exécution.

Généralement, l'objectif est soit de trouver la valeur minimale pouvant être générée par la fonction soit à l'inverse de trouver la valeur maximale. Ce choix s'explique généralement par la sémantique de la fonction. Par exemple, si celle-ci renvoie un taux d'erreur l'objectif de l'algorithme sera de minimiser les sorties de la fonction alors que si celle-ci renvoie un score, la finalité sera de maximiser ses sorties. Le Tableau 5 illustre un exemple de fonction d'évaluation sur 5 solutions potentielles. Dans cet exemple, trois variables : X, Y et Z décrivent une solution. L'objectif est d'obtenir une somme de X et Y et la différence de Z égale 0. La fonction d'évaluation qui en résulte représente l'écart absolu de cette somme par rapport à 0.

Solutions potentielles (X, Y, Z)	Evaluation (X+Y-Z=0)	Sortie
X=3 Y=2 Z=2	Abs (3 + 2 – 2)	3
X=2 Y=1 Z=7	Abs (2 + 1 – 7)	4
X=5 Y=5 Z=4	Abs (5 + 5 – 4)	6
X=1 Y=1 Z=5	Abs (1 + 1 – 5)	3
X=2 Y=2 Z=4	Abs (2 + 2 – 4)	0

Tableau 5 Exemple de fonction d'évaluation

Certaines applications nécessitent la prise en charge de contraintes d'égalités ou d'inégalités. Bien que celles-ci puissent être prises en compte durant d'autres étapes (e. g. lors de la mise à jour), il existe différentes méthodes permettant la gestion des contraintes à l'intérieur de la fonction d'évaluation. Parmi elles, nous pouvons citer : les coefficients de pénalisation, le comptage du nombre de contraintes non respectées.

La fonction d'évaluation peut être considérée comme une « boite noire » renvoyant une valeur de sortie. C'est pourquoi celle-ci peut être implémentée sous forme de fonctions mathématiques, de programmes, de modèles ou encore de web-services. Dans la majorité des applications, elle est totalement autonome. Cependant certaines applications nécessitent un guidage supplémentaire afin de la faire évoluer. Ce guidage peut se baser sur une planification des différents objectifs à satisfaire de manière chronologique ou encore être réalisé de manière interactive. L'interactivité implique qu'un utilisateur ou un groupe d'utilisateurs puisse interagir directement avec le processus d'optimisation.

L'analyse des différentes sorties produites par la fonction d'optimisation permet non seulement de connaitre la progression de l'algorithme vers la résolution du problème mais également de repérer la ou les solutions qui s'en rapprochent et doivent donc être utilisées lors de la phase suivante : la phase d'analyse.

2.2.3.3. Analyse

La phase d'analyse permet d'analyser les résultats associés lors de la phase d'évaluation à chacune des solutions. En fonction des résultats obtenus et du nombre d'itérations trois cas sont possibles. Dans le premier cas, si le nombre total d'itération de l'algorithme dépasse un seuil défini en amont l'algorithme s'interrompt. Dans le deuxième cas, celui-ci s'arrête également lorsque

l'évaluation de la solution ou d'une des solutions satisfait le ou les critères définis préalablement. Enfin dans tous les autres cas, l'algorithme itère en se dirigeant vers l'étape de « Sélection ».

2.2.3.4. Sélection

La sélection permet de guider le processus « d'intensification » (Cf 2.2.1.). Les différentes solutions, ainsi que leur voisinage sont mises en concurrence. Le choix entre les solutions utiles et les solutions à supprimer ou à remplacer se base sur l'évaluation de chaque solution réalisée lors de l'étape précédente. Les solutions sont triées en fonction de la qualité de leur évaluation.

A partir de l'évaluation attribuée à chaque solution, la sélection peut se faire de manière probabiliste (e. g. la probabilité des solutions d'être choisies pour les itérations suivantes est proportionnelle à leur évaluation) à partir de l'équation suivante :

$$Pi = \frac{fi}{\sum_{j=0}^{n} fj}$$

où fi désigne l'évaluation d'une solution divisée par les sommes des évaluations de toutes les solutions potentielles.

La sélection peut également être réalisée de manière déterministe (e. g. remplacement de la solution la plus mauvaise ou remplacement de la moitié des plus mauvaises solutions par la moitié des meilleures solutions).

Une fois cette phase de traitement et d'analyse des évaluations terminée, l'étape de mise à jour peut alors avoir lieux.

2.2.3.5. La mise à jour

La mise à jour comme son nom l'indique permet la modification des solutions existantes en vue de les améliorer. Elle peut se matérialiser par des perturbations des solutions précédemment sélectionnées ou encore par la création de nouvelles solutions et la suppression d'anciennes.

Les « S-Metaheuristics » et « P-Metaheuristics » (Cf 2.2.1.) peuvent toutes deux être mises à jour par des modifications aléatoires (e. g. opérateur d'improvisation dans l'algorithme de recherche harmonique). Ces modifications permettent d'assurer la diversification du processus de la recherche et ainsi éviter aux algorithmes un blocage dans des minimums locaux. Elles peuvent également être mises à jour par des perturbations basées sur la notion de « voisinage ». En effet, chaque solution potentielle possède un ensemble de solutions potentielles voisines. Une solution est voisine d'une autre lorsqu'il existe un lien de connexité entre ces deux solutions et que leur distance est inférieure à la métrique utilisée. (e. g. opérateur de voisinage dans la recherche par tabou). Ces modifications permettent quant à elles l'intensification du processus de recherche et permettent ainsi d'explorer en profondeur les zones de l'espace de recherche se rapprochant de la solution recherchée.

Les « P-Metaheuristics » peuvent également être mises à jour par des croisements, des mixages de solutions existantes. L'ensemble des solutions (e. g. composition dans la recherche harmonique) ou une seulement une partie de meilleures solutions (e. g. reproduction dans les algorithmes génétiques) peuvent partager leur description afin de générer de nouvelles solutions pour lesquelles la probabilité d'obtenir une évaluation de qualité sera forte.

Certains problèmes d'optimisation impliquent des contraintes. Elles peuvent être traitées lors de la phase de mise à jour grâce de deux stratégies distinctes. La première dite « stratégie de préservation » limite les mises à jour aux valeurs définies par l'espace de recherche. La seconde dite « stratégie de réparation » ne limite pas les mises à jour à l'espace de recherche défini. Cependant les valeurs obtenues sont « ramenées » dans l'espace de recherche une fois la mise à jour terminée. La Figure 10 présente un exemple pour chacune de ces stratégies. Dans cet exemple, les valeurs sont comprises dans l'intervalle entier [1,5]. On observe dans la « stratégie de préservation » que le mixage de deux solutions est ajusté directement au niveau de l'opérateur de croissement alors que dans la « stratégie de réparation » l'ajustement se fait à la suite de l'usage de l'opérateur. Dans les deux cas les résultats finaux sont identiques.

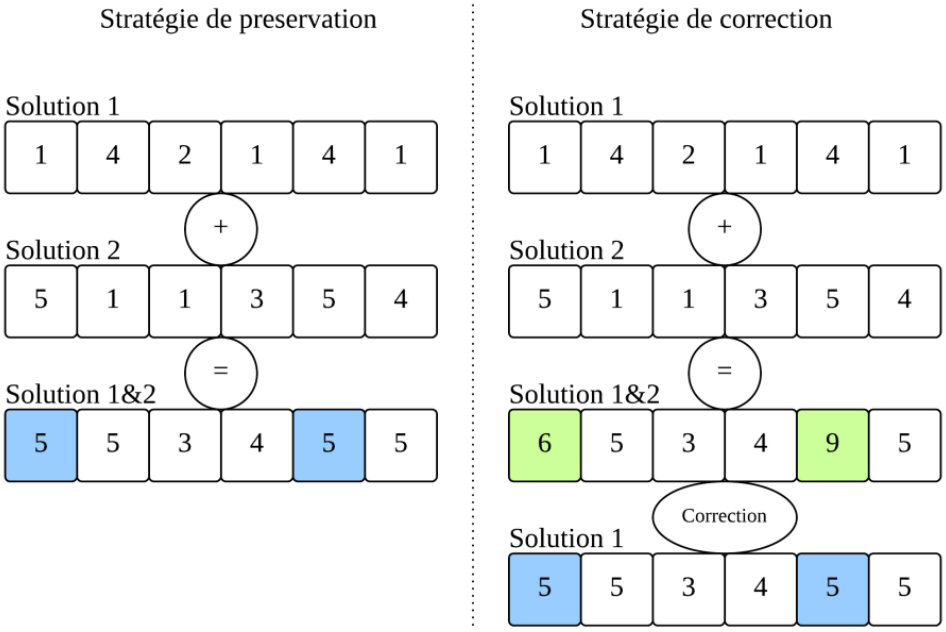


Figure 10 Exemple de traitement des contraintes lors de la mise à jour des solutions

Toutes les étapes décrites précédemment composent chacune des métaheuristiques existantes. Cependant chaque interprétation de ces étapes leur est propre. La question que nous pouvons alors nous poser est : quelle interprétation choisir ?

2.2.4. Choix d'une métaheuristique

Le choix d'une métaheuristique parmi les nombreux algorithmes disponibles est une opération cruciale pour le succès de l'optimisation. Cependant, cette étape est généralement réalisée de manière empirique en fonction des préférences de l'utilisateur. Le théorème « No Free Lunch » (Wolpert and Macready, 1997; Igel, 2014) affirme que pour deux algorithmes testés sur un vaste échantillon composé de différents problèmes : les performances obtenues seront quasi-similaires. En d'autres termes ce théorème démontre qu'il n'existe pas un algorithme supérieur à tous les autres.

Le choix d'une métaheuristique doit donc s'adapter à la nature du problème ainsi qu'aux finalités recherchées. Il ne se limite donc pas à un algorithme. Des paramètres doivent également être choisis ainsi que la variante de l'algorithme.

2.2.5. Les paramètres

Chaque métaheuristique dispose de paramètres qui lui sont propres. (E. g. le taux de mutation pour les algorithmes génétiques et le taux d'improvisation pour la recherche harmonique). Pour chaque paramètre, des intervalles de valeurs sont proposés sans de solides justifications. Ces propositions reposent uniquement sur des ensembles de tests hétérogènes réalisés de manière empirique. Comment savoir alors quel paramètre choisir pour un problème spécifique sans avoir à tester l'ensemble des combinaisons de paramètres possibles ? Pour répondre à cette problématique différentes techniques de paramétrage sont possibles (Eiben and Smit, 2011; Smit and Eiben, 2009). Comme nous pouvons le voir sur la Figure 11, celles-ci peuvent être classées en deux branches : les méthodes « off-line » et les méthodes « on-line » (Talbi, 2009).

Avec les méthodes « off-line », les paramètres sont choisis avant l'exécution du processus d'optimisation. Ces méthodes reposent sur une analyse de la nature du problème étudié et sur la corrélation avec des résultats antérieurs obtenus sur des problèmes analogues. Ces méthodes sont également utilisées pour résoudre différentes instances d'un même problème pour lequel un paramétrage correct avait été trouvé. Cependant, celles-ci de par leur aspect fortement déterministe nécessitent de disposer de temps importants en amont de l'exécution de l'optimisation. De plus, elles imposent des paramètres statiques qui devront rester fixes durant l'ensemble du processus d'optimisation.

Les méthodes « on-line » quant à elles sont plus flexibles. Les paramètres peuvent être mis à jour de manière dynamique durant le processus de recherche. Ces mises à jour peuvent être aléatoires ou déterministes. Une autre approche « on-line » consiste à intégrer les différents paramètres dans la description des solutions générant ainsi un phénomène d'apprentissage, d'auto-adaptation.

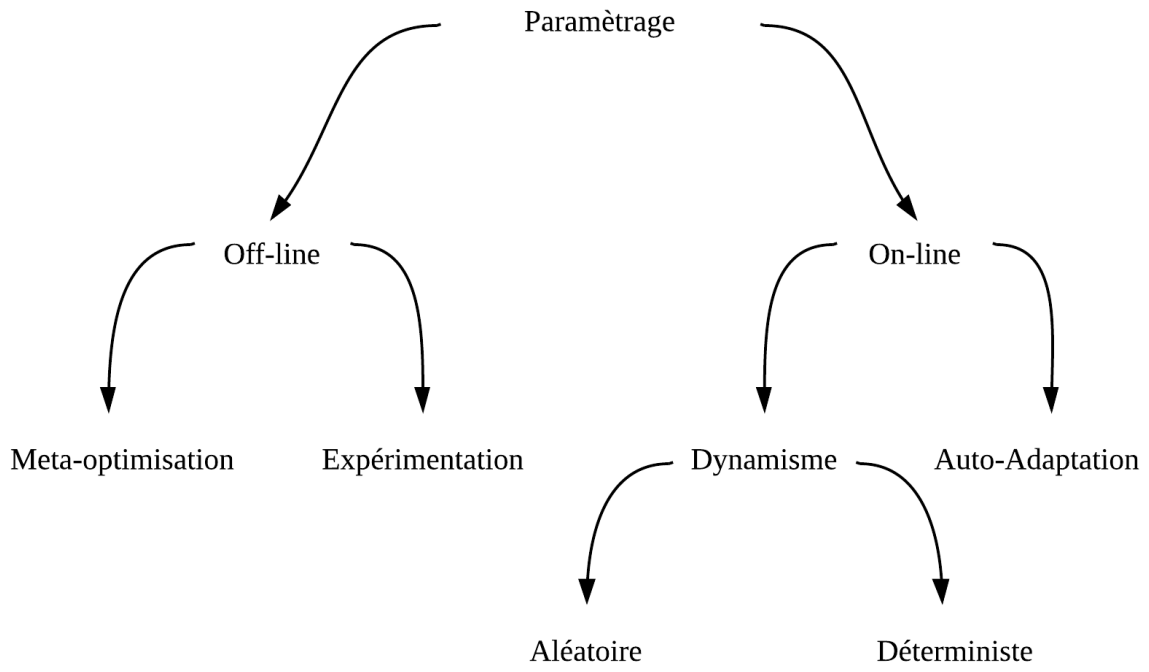


Figure 11 Classification des techniques de paramètrage

2.2.6. Les variantes

Pour répondre aux besoins de performance croissants, différents concepts sont venus étendre les métaheuristiques : l'analyse multicritères, la parallélisation, l'hybridation ainsi que le concept d'hyperheuristique.

2.2.6.1. Métaheuristiques multicritères

Rares sont les problèmes du monde réel se limitant à un seul critère. En effet, la majorité des problèmes scientifiques et industriels actuels présentent de multiples aspects et objectifs. Bien qu'ils puissent être complémentaires, ils sont généralement antagonistes (e. g. maximiser la production d'énergie d'une centrale électrique tout en limitant la pollution émise). Des compromis doivent alors mis en place.

En ce sens, l'optimisation multicritère ne se limite pas à minimiser ou maximiser une unique fonction d'évaluation. Elle consiste soit à minimiser un ensemble de fonctions, soit à maximiser un ensemble de fonctions ou enfin à minimiser certaines fonctions et en maximiser d'autres. Dans sa définition formelle, l'évaluation multicritère se base sur deux espaces : l'espace de variables décisionnelles à optimiser et l'espace des évaluations obtenues. Comme nous pouvons le voir sur l'exemple de la Figure 12, ces espaces peuvent être de dimensions différentes. En effet, une même variable peut être associée à différents critères et inversement. Sur cet exemple, les deux variables représentées par un point vert et un point bleu possèdent chacune une évaluation par rapport à trois critères.

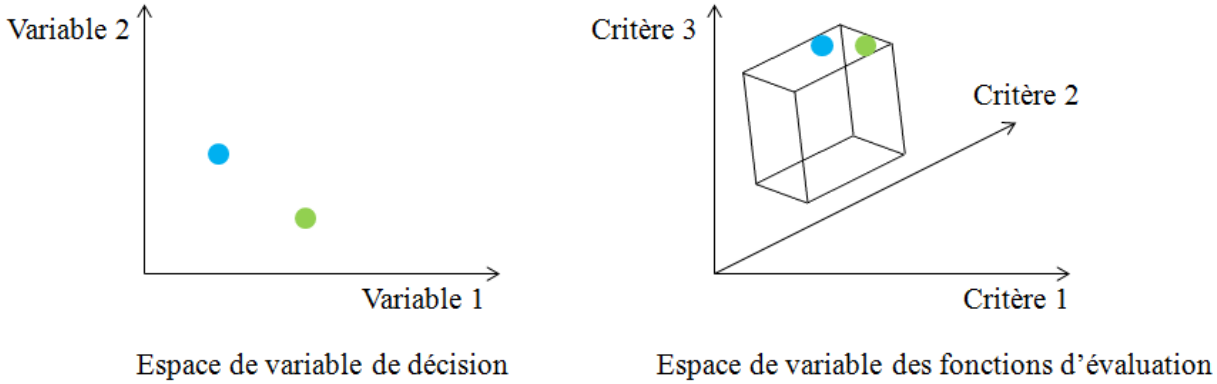


Figure 12 Espace de variables et espace de décisions

Une décision multicritère peut être décomposée en deux étapes. La première est l'optimisation des différentes fonctions objectives et la seconde, le processus de décision pouvant impliquer des compromis choisis par le décideur. A travers l'ordre d'exécution de ces deux étapes, les méthodes d'optimisation multicritères peuvent être classées dans trois catégories distinctes : les méthodes à « priori », les méthodes à « posteriori » et les méthodes « progressives » (Talbi, 2009).

Les méthodes à « priori » consistent à définir l'importance des différents critères étudiés préalablement au processus d'optimisation (e. g. un problème d'optimisation de la production pour lequel la qualité du produit sera prioritaire sur la quantité fabriquée). En d'autres termes, la décision est prise « avant » le processus d'optimisation. Parmi ces méthodes nous pouvons citer les méthodes d'ordonnancement ainsi que les méthodes d'agrégations linéaires et non linéaires. Le principe des méthodes d'agrégations est de réduire un problème multicritère en un problème classique. Celles-ci

sont rapides à mettre en œuvre. Cependant elles limitent fortement le nombre de solutions potentielles et réduisent ainsi fortement les choix du décideur.

Les méthodes à « posteriori » quant à elles inversent cet ordre. La recherche des différentes alternatives est la plus large possible. Ce n'est qu'ensuite que le processus de décision est effectué. Ce n'est plus une valeur optimale qui est recherchée mais un ensemble de valeurs contenant les différentes combinaisons représentant les compromis les plus adaptés aux objectifs énoncés. (e. g. lister les différents itinéraires de livraison possibles en fonction des horaires souhaités par les clients tout en limitant la consommation de carburant). Cet ensemble de compromis se nomme « ensemble de Pareto ». Les meilleures solutions présentes sur cet ensemble forment ce que l'on appelle : « la frontière de Pareto ». Chacune de ces solutions est dite « Pareto optimale » car il n'existe aucune alternative à la solution qui pourrait améliorer la réponse à un critère spécifique sans compromettre la qualité de la réponse pour les autres critères. Les autres solutions présentes appartiennent au sous-ensemble des « solutions dominées ». La Figure 13 illustre ces concepts dans un exemple où deux objectifs doivent être minimisés. Chaque cercle représente l'évaluation d'une solution d'après deux fonctions d'évaluation : A et B.

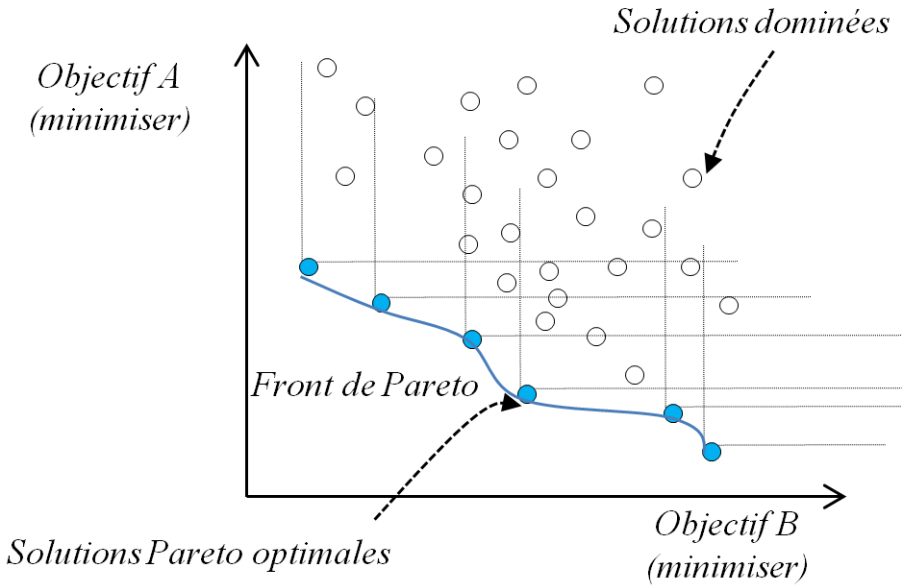


Figure 13 Exemples d'ensembles de Pareto

Enfin une troisième catégorie de méthodes multicritères regroupe les métaheuristiques exécutées de manière coopérative et mixtes (e. g. le décideur définit une fourchette de qualité pour ces produits, pouvant être ajustée durant le processus d'optimisation). Dans ce cas-là, la décision est prise « pendant » le processus d'optimisation.

De nombreuses métaheuristiques (Ficici, 2008) ont été modifiées afin de prendre en charge des problèmes multicritères. Parmi elles nous pouvons citer les exemples suivants: « Mutli-Objective Genetic Algorithm »(MOGA) (Murata and Ishibuchi, 1995), « Nondominated sorting Genetic Algorithm » (NSGA) (Srinivas and Deb, 1994), « Niched-Pareto Genetic Algorithm » (NPGA) (Horn et al., 1994), « Mutli-objective harmony search » (Ricart et al., 2011).

2.2.6.2. Métaheuristiques parallèles

Après des années d'augmentation de la fréquence des processeurs, nous assistons depuis une dizaine d'années à la multiplication du nombre de cœurs présents sur les équipements informatiques permettant la parallélisation des processus. L'usage de la parallélisation apparait de plus en plus comme incontournable pour les métaheuristiques (Alba, 2005). Elle permet en effet une convergence plus rapide des solutions dans le temps qui jusqu'alors pouvait être dissuasif pour certaines applications.

Dans la littérature, une distinction est faite entre l'utilisation de processus parallèles pour les métaheuristiques à solution unique (S-Metaheuristic) et les métaheuristiques à solutions multiples (P-Metaheuristic).

Les méthodes se basant sur une solution unique peuvent utiliser la parallélisation de trois manières. La première consiste à lancer simultanément plusieurs processus avec des paramètres identiques ou différents et pouvant progresser de manière totalement autonome ou ponctuellement collaborative. La seconde consiste quant à elle à calculer simultanément les différentes variations possibles de la solution courante lors des phases de mise à jour. Enfin, la troisième a comme objectif de diviser la fonction d'évaluation en différentes sous-fonctions indépendamment les unes des autres. Ces trois concepts sont résumés dans la Figure 14.

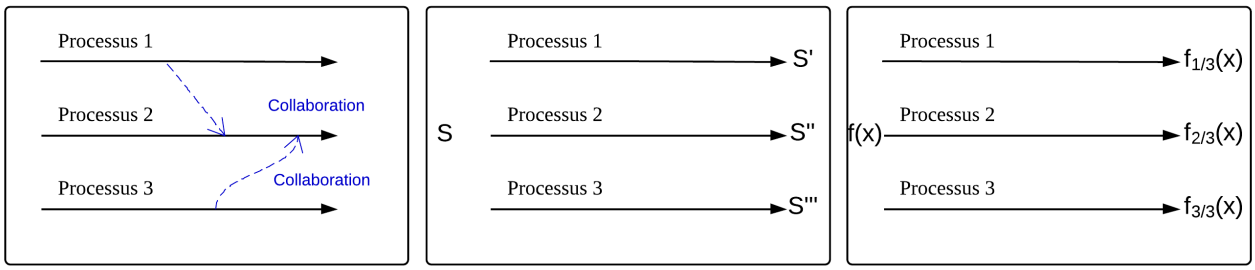


Figure 14 Parallèlisation de métaheuristiques à solution unique (S-Metaheuristic)

Les métaheuristiques qui utilisent plusieurs solutions, peuvent utiliser la parallélisation de deux manières. La première consiste à exploiter l'aspect unitaire de chaque solution. Certaines opérations : génération, évaluation, perturbations aléatoires, peuvent être réalisées de manière isolée. De plus, l'évaluation de manière isolée peut être fortement utile dans le cadre d'applications multicritères (e. g évaluer une même solution sur un modèle quantitatif et sur un modèle qualitatif distincts). La deuxième approche consiste quant à elle à diviser la population en sous-groupes également appelés ilots et évoluant de manière indépendante et ponctuellement collaborative. Ce principe est renforcé par la prolifération des services des web-services (e. g. p2p) et du cloud (e. g. grid computing systems).

Ces deux concepts sont illustrés dans les deux exemples présents sur la Figure 15. Dans l'exemple de gauche, nous pouvons voir que pour chacune des solutions : A, B, C, D, E, F un processus est associé pour les opérateurs unaires. Dans l'exemple de droite, nous pouvons voir que trois ilots distincts contenant chacun deux solutions et des paramètres pouvant être différents sont utilisés et peuvent partager de manière ponctuelle et non systématique des informations.

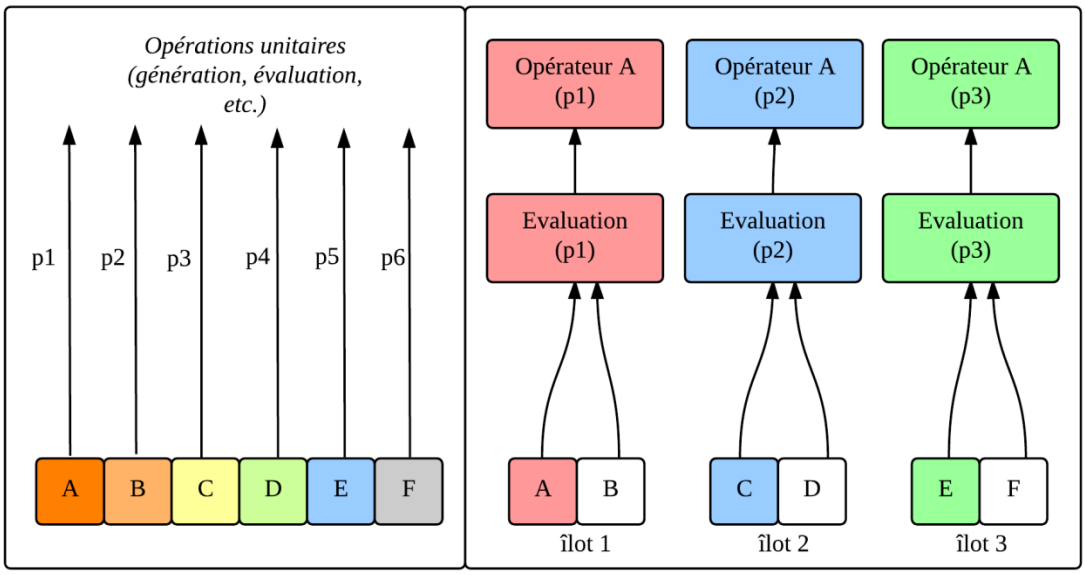


Figure 15 Parallèlisation de métaheuristiques à solutions multiples (P-Metaheuristics)

2.2.6.3. Métaheuristiques hybrides

Plus récemment, depuis les années 80, différents travaux cherchent à améliorer les performances des métaheuristiques classiques par des approches hybrides (Raidl, 2006). Pour cela, une métaheuristique peut être combinée à d'autres techniques d'optimisation tel que cela est présenté sur la Figure 16.

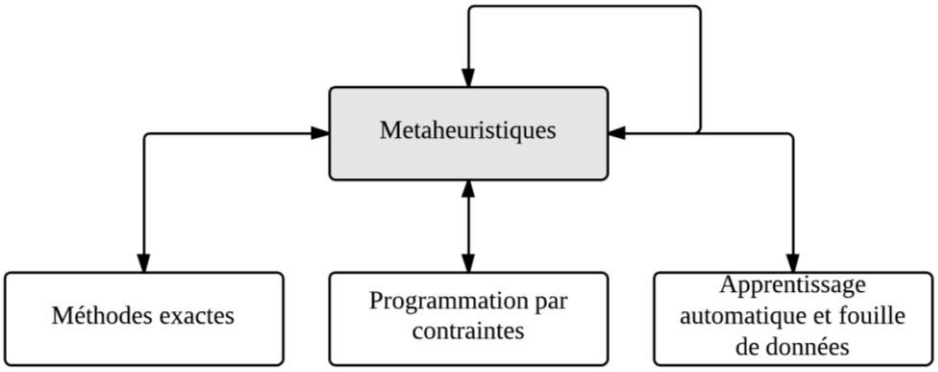


Figure 16 Les différentes catégories d'hybridation des métaheuritiques

Les méthodes exactes peuvent être utilisées pour des phases d'intensification comme par exemple lorsque les résultats de l'algorithme sont extrêmement proches de l'objectif recherché. La programmation par contraintes quant à elle se base sur des algorithmes de propagation et sur la division du problème en sous-problèmes. Son utilisation permet de réduire rapidement l'espace de recherche du problème à optimiser en limitant certaines explorations inutiles. De plus, elle permet une description de haut niveau et déclarative du problème pouvant être facilement abordée par les décideurs. Enfin, la fouille de données se base sur l'extraction de connaissances grâce à des outils d'apprentissage supervisé ou indépendant, de statistiques, d'échantillonnages, de classifications, de reconnaissances de motifs. L'utilisation de tels outils nécessite de disposer de volumes de données importants. Elle permet une meilleure adaptation de la métaheuristique au problème (e. g. auto paramétrage, choix des opérateurs adéquats).

Une métaheuristique peut également être considérée comme hybride lorsque plusieurs instances d'un même algorithme sont utilisées avec des paramètres et des opérateurs hétérogènes. (e. g. algorithmes génétiques sous formes d'ilots utilisant des taux de mutations et des croisements différents)

Face aux nombre grandissant de métaheuristiques hybrides deux classifications sont proposées comme cela est présenté sur la Figure 17.

La première approche permet de classer les différents algorithmes de manière « hiérarchique ». Cette classification isole les méthodes dites de « bas niveau » pour lesquelles l'hybridation est réalisée de manière interne à l'algorithme d'optimisation et les méthodes dites de « haut niveau » pour lesquelles l'hybridation se manifeste par des interfaces entre les différentes méthodes qui conservent alors leur autonomie. Elle isole également deux types d'exécutions : relais et équipe. En mode « relais », les différentes technologies d'optimisation sont utilisées de manière concurrente. En mode « équipe », les différentes entités d'optimisation s'exécutent en parallèle et de manière collaborative par le partage et l'échange d'informations.

La seconde approche permet une description des métaheuristiques de manière déstructurée, « à plat ». Une distinction est faite entre les méthodes homogènes utilisant une seule métaheuristique avec différents paramètres et opérateurs et les méthodes hétérogènes utilisant différents algorithmes. La coopération entre les différents composants de l'algorithme est également décrite. Celle-ci est globale quand l'ensemble des modules d'optimisation disposent des mêmes informations et partielle quand l'information est répartie entre les différents modules. Enfin, l'approche à plat distingue les méthodes génériques pouvant être appliquées facilement à tout type de problèmes et les méthodes spécifiques applicables à une certaine catégorie de problèmes (e. g. métaheuristiques hybrides spécialisées dans le problème du voyageur de commerce).

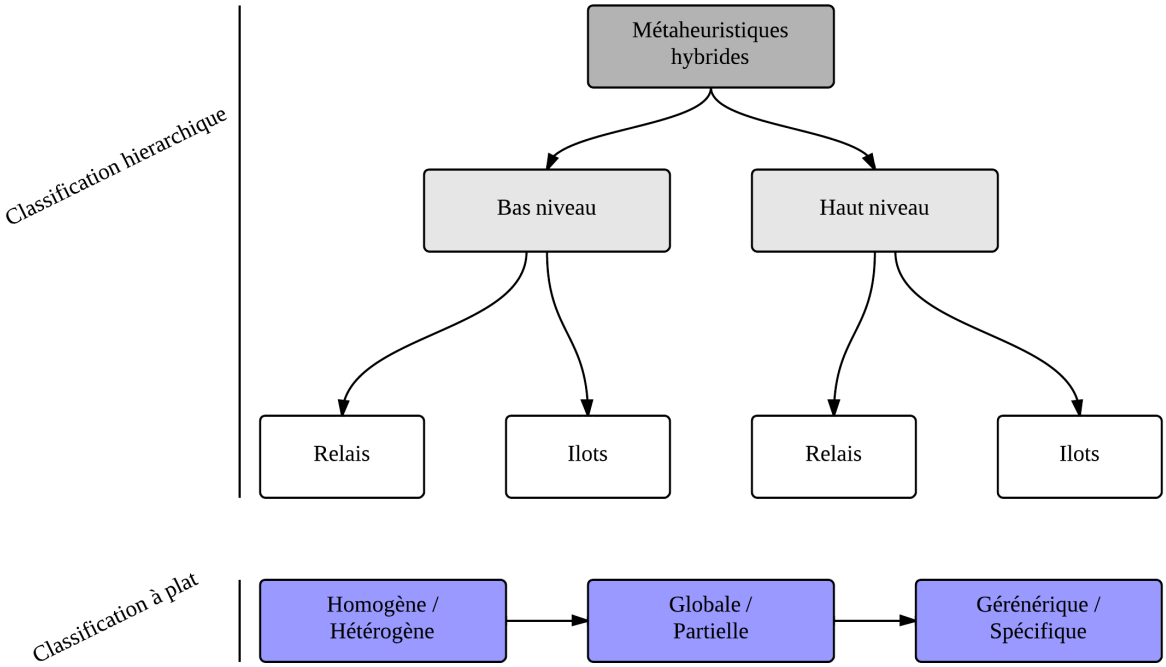


Figure 17 Classification des métaheuristiques hybrides (Talbi, 2009)

2.2.6.4. Hyperheuristiques

La dernière variante regroupe les « hyperheuristiques » (Cowling et al., 2001). Le terme « Hyperheuristiques » peut être trompeur. Il ne désigne pas une nouvelle classe d'algorithmes d'optimisation plus efficaces que les métaheuristiques mais l'utilisation des métaheuristiques pour choisir de manière automatique les heuristiques ou métaheuristiques pour un problème spécifique. En ce sens, l'objectif des hyperheuristiques n'est donc pas de solutionner un problème, mais de produire l'outil pour y parvenir (Burke et al., 2013; Chakhlevitch and Cowling, 2005; Özcan et al., 2008).

Les hyperheuristiques se placent à un niveau conceptuel de généricité supérieur à celui des métaheuristiques qui bien qu'applicables à tout type de problèmes, n'en demeurent pas moins difficiles à mettre en œuvre dans certains cas.

Comme cela est visible sur la Figure 18, l'élaboration d'une hyperheuristique peut se baser sur différentes sources de données. Elle peut se structurer à partir d'un raisonnement aléatoire ou sur un raisonnement plus avancé impliquant des techniques d'apprentissages. Cet apprentissage peut être « On-line » et produire des méthodes de résolution au fil de l'exécution ou être « Off-line » et produire des méthodes de résolutions préalablement à l'exécution. Pour cela des règles, des jeux de tests et de classification doivent être disponibles. L'hyperheuristique générée peut se présenter sous deux formes. Elle peut se présenter sous la forme d'une sélection de méthodes adaptable à tout type de problèmes et pouvant être construite ou modifiée durant l'optimisation. Elle peut également se présenter sous la forme d'une méthode spécifique générée en fonction du problème étudié. Cette génération peut se bâtir progressivement en fonction des besoins ou être générée une seule fois tout en étant modifiable.

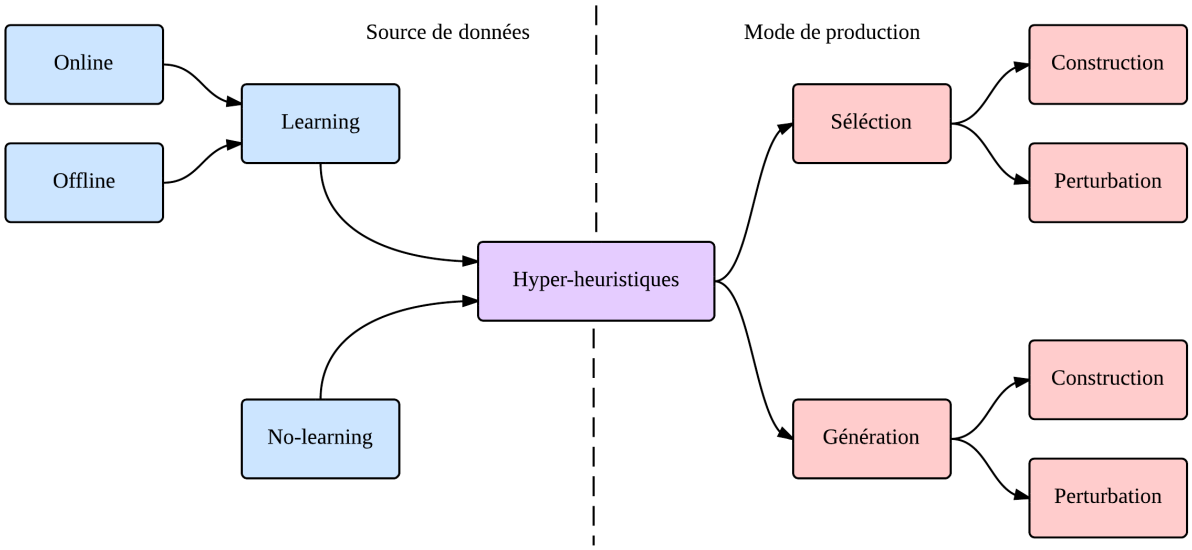


Figure 18 Caractérisation des hyperheuristiques

2.3 Principaux algorithmes

De très nombreux algorithmes sont présents dans la littérature. Leur classification peut se faire en fonction de différents critères tels que : le nombre et la représentation des solutions qu'ils utilisent, la source d'inspiration sur laquelle ils se basent. Nous avons choisi d'utiliser la classification la plus largement utilisée à savoir : le nombre de solutions utilisées lors du processus d'optimisation. Celle-ci isole deux groupes : « les métaheuristiques à solution unique » et les « métaheuristiques à solutions multiples ». La Figure 19 qui se base sur cette classification présente une liste non exhaustive des principales catégories de métaheuristiques actuellement utilisées. Pour chacune de ces catégories, nous allons présenter le fonctionnement algorithmique d'une des méthodes et illustrer son déroulement par des exemples simples.

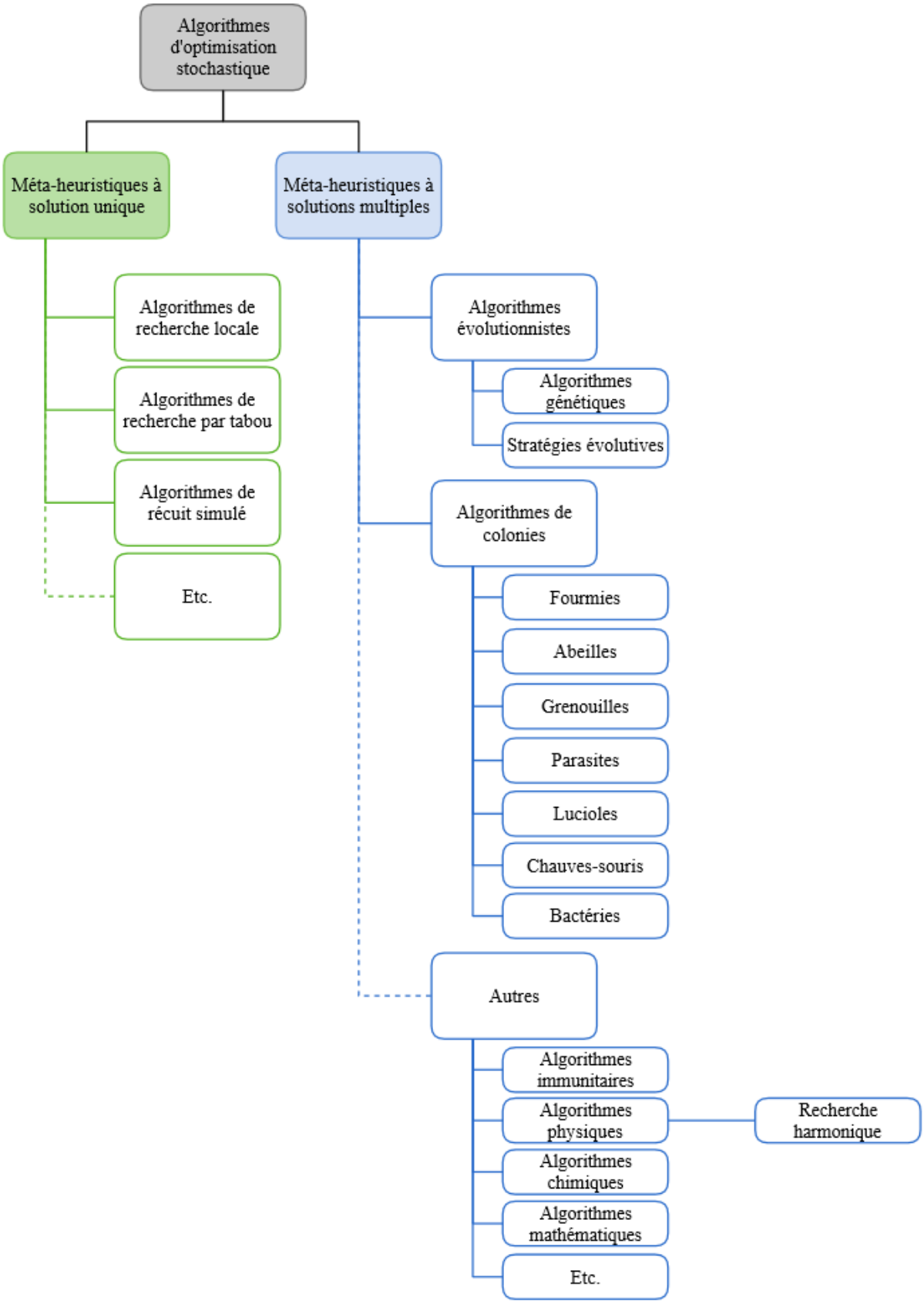


Figure 19 Classfication des principales catégories de métaheuristiques

2.3.1. Les métaheuristiques à solution unique (S-Metaheuristics)

Parmi la multitude de « métaheuristiques à solution unique » nous avons choisi de focaliser notre étude sur les plus couramment utilisées à savoir : la recherche locale, la recherche par tabou et le recuit simulé.

2.3.1.1. La recherche locale

Parmi les métaheuristiques à solution unique, la recherche locale est la plus ancienne. Son fonctionnement est extrêmement simple à appréhender et à implémenter. L'idée principale est de parcourir l'espace de recherche par les solutions voisines de la solution potentielle en choisissant celle ayant la meilleure évaluation. Son déroulement est décrit dans l'Algorithme 1.

```
1  Début
2  s* = générerSolutionAléatoire()
3  répéter
4          s' = générerVoisin(s)
5          Si f(s') > f(s)
6                  s* = s'
7  Jusqu'à solutionTrouvée()
8  Retourner s
9  Fin
```

Algorithme 1 Recherche locale (Local Search)

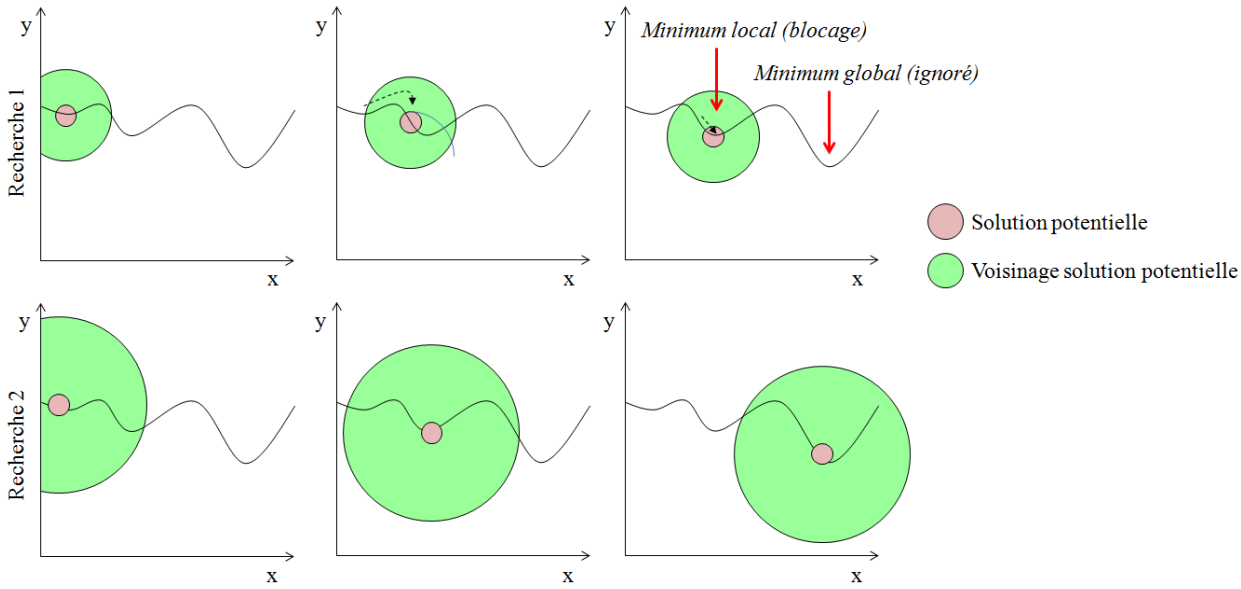


Figure 20 Exemples de recherches locales avec différents niveaux de voisinage

Cette méthode est largement utilisée pour des problèmes de nature quasi-linéaire de petite taille. Bien que celle-ci soit considérée comme la première métaheuristique, son usage est de moins en moins répandu. Cela s'explique par un risque important de blocage dans des minimums locaux. En effet, seules les modifications vers des solutions de meilleure qualité sont acceptées. D'autres méthodes ont donc été développées pour contourner cette limite.

2.3.1.2. La recherche par tabou

L'algorithme de recherche par tabou (F Glover, 1990; Fred Glover, 1990; Glover, 1989) peut être considéré comme une extension de la recherche locale. Il se base sur l'utilisation d'une ou plusieurs mémoires permettant à l'algorithme de s'extraire des minimums locaux et ainsi diversifier la recherche afin de parcourir l'ensemble de l'espace de recherche. Son déroulement est décrit dans l'algorithme 2.

```
1   Début
2   mémoire = []
3   s*  = générerSolutionAléatoire()
4   répéter
5           s' = trouverMeilleureAdmissible(générerVoisins(s))
6           Si f(s') > f(s*)
7                   s* = s'
8                   MiseAJour(mémoire)
9   Jusqu'à solutionTrouvée()
10  Retourner s*
11  Fin
```

algorithme 2 Recherche par tabou (Tabu Search)

La première mémoire utilisée par l'algorithme est la « mémoire à court terme » (short-term memory). Son utilisation est indispensable. Elle contient les dernières solutions potentielles visitées lors des précédentes itérations. Cette mémoire utilise une structure de données de type FIFO (First In First out). L'utilisation de cette mémoire contraint l'algorithme à proposer des solutions qui n'ont pas été visitées (présentes dans la liste). Des solutions de qualité moindre peuvent donc être acceptées ce qui évite à l'algorithme de rester bloqué dans des minimums locaux. Le choix du nombre de valeurs stockées dans la liste est une opération délicate. En effet, une taille trop petite augmentera considérablement le risque de blocage cyclique dans un minimum local alors qu'une taille trop grande sera trop restrictive et éloignera la recherche des régions de l'espace de recherche à fort potentiel. Un juste milieu entre interdiction et autorisation doit donc être trouvé. L'exemple de la Figure 21 illustre ce principe.

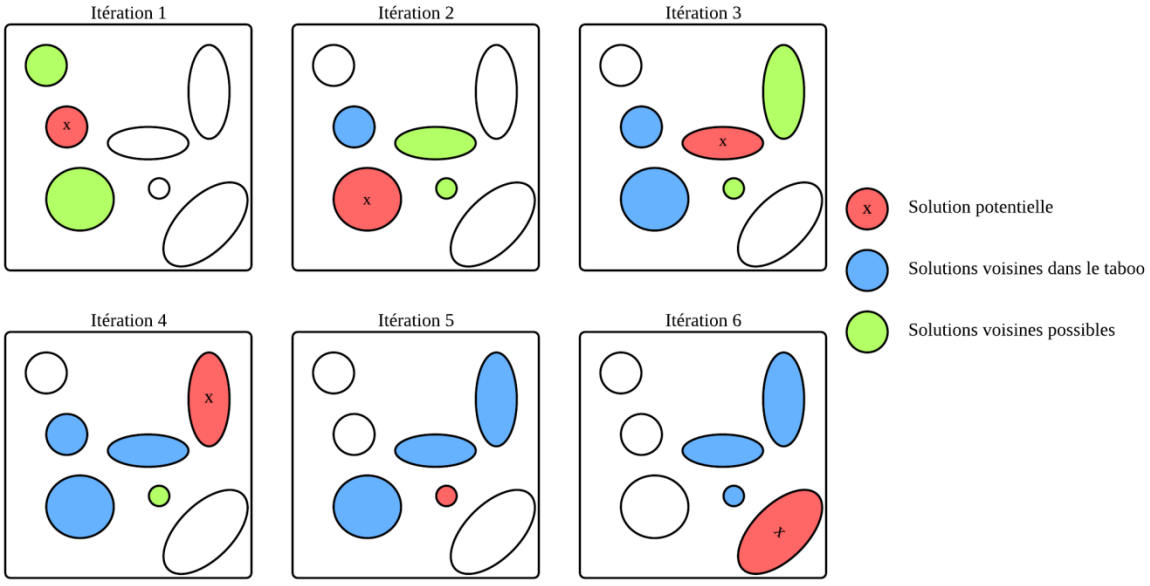


Figure 21 Exemple de recherche du cercle d'aire maximale par tabou (mémoire de 3)

Dans cet exemple, l'objectif est de trouver le cercle ayant la plus grande surface. Lors des itérations 4 et 5, l'intérêt de la liste de solutions taboues est visible. Nous remarquons que la solution choisie parmi le voisinage n'est pas la meilleure mais celle-ci n'appartenant pas à l'ensemble de valeurs présentes dans la liste. Il est à noter que le nombre de solutions explorées et présentes dans cette liste peut être fixe ou flexible (Battiti et al., 1993).

Un deuxième type de mémoire peut être utilisé par l'algorithme : la mémoire à moyen terme (medium-term memory). Celle-ci est facultative. Elle permet de conserver en mémoire les meilleures solutions rencontrées lors du processus de recherche. En ce sens, cette mémoire est moins volatile que la mémoire à court terme. Son objectif consiste à analyser les caractéristiques des meilleures solutions en vue d'orienter le processus de recherche vers celles-ci. Elle permet donc de renforcer le principe d'intensification.

Un troisième type de mémoire peut également être intégré dans l'algorithme : la mémoire à long terme (long-term memory). Elle permet de mémoriser le parcours de l'espace de recherche par l'algorithme durant une grande partie, voir l'ensemble de la recherche. Son objectif est de cibler les zones de l'espace de recherche qui n'ont pas été exploitées et permettre ainsi à l'algorithme de se diriger vers celles augmentant le niveau de diversification.

2.3.1.3. Le recuit simulé

L'algorithme du recuit simulé (Kirkpatrick et al., 1983; Laarhoven and Aarts, 1987) se construit à partir d'une analogie entre le processus physique de refroidissement des métaux et la recherche de solutions présentée dans le Tableau 6. En effet, au niveau physique lorsqu'un matériau est chauffé, les différents atomes qui le composent vont se déplacer rapidement. A l'inverse, quand celui-ci est refroidi, les différents atomes qui le composent vont se stabiliser. Si ceux-ci sont refroidis rapidement la structure obtenue sera irrégulière et de mauvaise qualité. A l'inverse si ceux-ci sont refroidis progressivement la structure obtenue sera homogène et de bonne qualité (e. g. la fabrication d'objets en cristal). Ce concept de refroidissement a été adapté pour l'optimisation comme cela est visible sur l'Algorithme 3.

Optimisation	Inspiration
Espace de recherche	Ensemble des organisations d'atomes possibles
Solution	Structure d'un matériau
Représentation	Vecteur
Evaluation	Organisation des atomes
Intensification	Processus de refroidissement
Diversification	Processus d'échauffement

Tableau 6 Analogies du récuit simulé

```
1  Début
2  t ← températureInitiale()
3  s* ← s
4  Répéter
5          s^i= générerSolutionVoisine()
6          t = nouvelleTemperature()
7          Si f(s^i) ≤ f(s)
8                  s ← s^i
9                  Si f(s*) < f(s^i)
10                         s* ← s^i
11         Sinon
12                 Si (δ< e^( (-f(s*)- f(si)) / t))
13                         s* ← s^i
14 Jusqu'à solutionTrouvée()
15 Retourner s*
16 Fin
```

Algorithme 3 Recuit simulé (Simulated Annealing)

Comme on peut le voir dans l'algorithme, le mouvement vers des solutions de qualités moindres est autorisé permettant ainsi de s'extraire de certains minimums locaux. Pour cela, le raisonnement utilise à chaque itération le « critère de Metropolis » basé lui-même sur la « probabilité de Bolzman » et se calcule par l'équation suivante afin de déterminer la probabilité d'acceptation d'une solution de qualité moindre:

$$P(E) = e^{(\frac{\Delta E}{ti})}$$

Dans cette équation, ΔE représente l'écart entre l'évaluation de la meilleure solution et de la solution proposée par l'itération actuelle et t_i représente la température de l'itération actuelle. Un nombre δ compris entre 0 et 1 est généré aléatoirement à partir d'une distribution uniforme. Si celui-ci est inférieur à la probabilité calculée, la solution actuelle remplace alors la meilleure solution. Comme cela est illustré dans les exemples du Tableau 7 Exemples de remplacement dans l'algorithme du recuit simulé et de la Figure 22, la probabilité d'accepter une solution d'évaluation inférieure est corrélée avec la température tout en conservant une part de raisonnement aléatoire. Dans cet exemple d'approximation d'un cercle par le tracé de huit points nous pouvons observer les conséquences du refroidissement de la température sur le nombre de perturbations, chacune représentée par une flèche rouge.

Evaluation S1	Evaluation S2	δ (aléatoire)	t (température)	$E^{(\Delta E / t)}$	S2 remplace S1
10	8	0.855	20	0.90483741	Vrai
10	8	0.554	20	0.90483741	Vrai
10	6	0.911	20	0.81873075	Faux
10	6	0.801	20	0.81873075	Vrai
10	8	0.999	2	0.36787944	Faux
10	8	0.321	2	0.36787944	Vrai
10	6	0.456	2	0.13533528	Faux
10	6	0.233	2	0.13533528	Faux

Tableau 7 Exemples de remplacement dans l'algorithme du recuit simulé

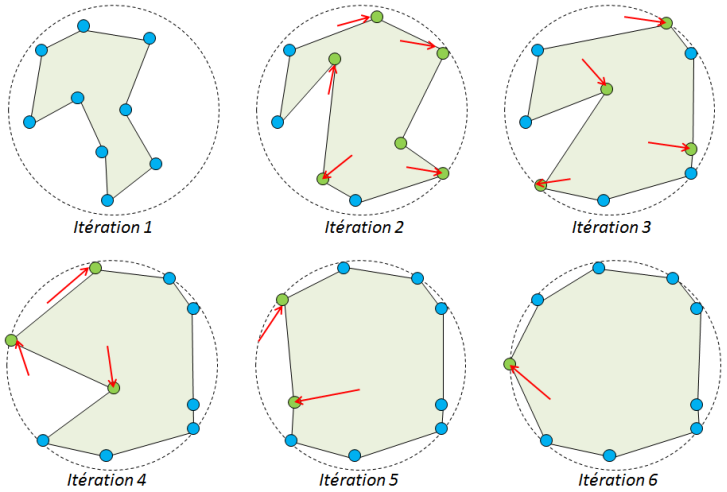


Figure 22 Execution de l'algorithme de récuit simulé pour l'approximation d'un cercle

2.3.2. Métaheuristiques à solutions multiples (P-Metaheuristics)

Contrairement aux métaheuristiques à solution unique, les métaheuristiques à solutions multiples se basent sur un ensemble de solutions évoluant simultanément et de manière collaborative. Bien que le nombre de métaheuristiques à solution unique soit considérable, celui-ci est largement dépassé par le nombre de métaheuristiques à solutions multiples. Dans ce contexte, nous avons choisi de réaliser notre étude de l'existant sur les catégories les plus populaires : les algorithmes évolutionnistes, les algorithmes immunitaires, les algorithmes de colonies et algorithmes d'inspirations diverses.

2.3.2.1. Les algorithmes évolutionnistes

L'évolution des espèces décrite par C. Darwin et observée de nos jours par l'étude de l'ADN est un excellent exemple d'optimisation. Durant les années 70, de nombreux algorithmes d'optimisation s'en inspirant sont donc apparus (Simon, 2013). Ceux-ci se basent sur les analogies présentes dans le Tableau 8.

Optimisation	Inspiration
Espace de recherche	Ensemble des gênes possibles
Solution	Individu
Représentation	Vecteur, Arbre
Evaluation	Adaptation
Intensification	Sélection naturelle
Diversification	Mutations génétiques

Tableau 8 Analogies des algorithmes évolutionnistes

Parmi ces algorithmes évolutionnistes, les algorithmes génétiques (Holland, 1992) sont la sous-catégorie la plus largement diffusée tant sur le plan industriel que scientifique. Le comportement classique d'un algorithme génétique est présenté dans l'algorithme 4.

```
1  Début
2  P = générerSolutionsAléatoires(taillePopulation)
3  évaluer(P)
4  Répéter
5        Pour i ← 1 à taillePopulation
6              R ← sélectionnerReproducteurs(n,P)
7              N ← créerNouveaux(m,R)
8              N ← muterNouveaux(N)
9              évaluer(N)
10             remplacerAncienParNouveaux(P,N)
11       Fin pour
12 Jusqu'à solutionTrouvée()
13 Retourner meilleurSolution(S)
14 Fin
```

algorithme 4 Algorithmes génénétiques (Genetic Algorithms)

Ce comportement est illustré dans l'exemple présent dans la Figure 23. L'objectif de ce problème est d'optimiser une suite de cinq nombres en vue d'obtenir une somme nulle. Comme on peut le voir, suite à l'évaluation de chacune des solutions potentielles, seulement quatre solutions sont séléctionnées et vont être utilisées lors de la phase de croisement. Chaque couple de solutions séléctionnées génère à partir de sa description un couple de nouvelles solutions. Les nouvelles solutions sont ensuite mutées afin d'assurer la diversification de la recherche comme c'est le cas par exemple dans la nouvelle solution [2, 1, -3, -5, 1] dont la deuxième valeur est remplacée par 5 produisant la solution potentielle [2, 5, -3, -5, 1] dont la somme des nombres est bien égale à la valeur recherchée : 0.

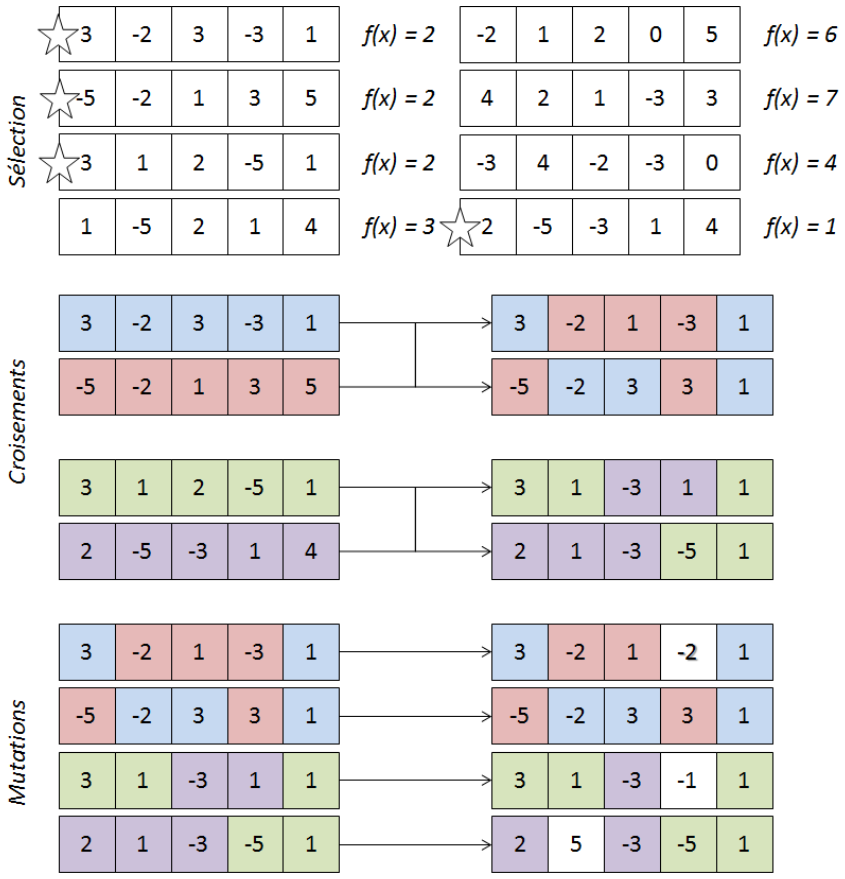


Figure 23 Exemple d'exécution des algorithmes génétiques

2.3.2.2. Les algorithmes immunitaires

Tout comme l'évolution, l'observation et les récentes découvertes concernant le système immunitaire sont une source d'inspiration pour la création d'algorithmes dédiés à l'optimisation. En effet, le système immunitaire fait preuve d'une intelligence remarquable. Il permet aux êtres vivants de se défendre contre des cellules étrangères (e. g. virus) grâce à l'utilisation de différentes cellules (e. g. lymphocytes B et lymphocytes T) douées de remarquables propriétés d'adaptation. Le Tableau 9 illustre le raisonnement analogique permettant d'adapter ces concepts à des algorithmes destinés à l'optimisation.

Optimisation	Inspiration
Espace de recherche	Ensemble des anticorps possibles
Solution	Anticorps
Représentation	Vecteur
Evaluation	Affinité des anticorps avec un virus
Intensification	Clonage
Diversification	Hyper-mutation

Tableau 9 Analogies des algorithmes immunitaires

L'algorithme 5 représente l'une des méthodes immunitaire la plus fréquemment utilisée dans le cadre de processus d'optimisation : l'algorithme de clonage par sélection (De Castro and Von Zuben, 2000). Un exemple d'exécution est présenté sur la Figure 24.

```
1   Début
2   systèmeImmunitaire = créerAnticorpsAléatoires()
3   évaluer(systèmeImmunitaire)
4   normaliserEvaluation(systèmeImmunitaire)
5   Répéter
6           clones ← créerClones(systèmeImmunitaire)
7           clones ← hyperMuterClones(systèmeImmunitaire)
8           évaluer(clones)
9           normaliserEvaluation(clones)
10          nouveaux ← créerNouveaux()
11          évaluer(nouveaux)
12          normaliserEvaluation(nouveaux)
13          intégrer(clones, nouveaux, systèmesImmuniatire)
14  Jusqu'à solutionTrouvée()
15  Retourner meilleurSolution(systèmeImmunitaire)
16  Fin
```

algorithme 5 Séléction par clonage (Clonal Selection Algorithms)

Dans cet exemple de la Figure 24, différentes solutions sont générées aléatoirement lors du démarrage de l'algorithme. Ces solutions sont ensuite évaluées et normalisées. Le processus itératif commence et permet à chaque solution de créer un nombre de clones proportionnel à la valeur de son évaluation normalisée. À ce stade, les clones sont la copie conforme de l'anticorps d'origine. Chacun de ces clones va alors être hyper-muté. Le taux de mutation sera quant à lui inversement proportionnel à la qualité de son évaluation. Les clones de bonne qualité seront légèrement modifiés alors que les clones de qualité moyenne seront fortement modifiés. Théoriquement, il n'existe pas de clones de mauvaise qualité, les anticorps de mauvaise qualité n'étant pas clonés. Une fois cette étape terminée chaque clone va à son tour être évalué et son évaluation normalisée. Afin d'assurer un certain niveau de diversité de nouveaux anticorps seront générés aléatoirement et évalués de la même manière. L'étape d'intégration pourra alors débuter. Celle-ci consiste à conserver un nombre limité d'anticorps

choisis parmi les originaux, les clones et les nouveaux. Cette sélection s'effectue en fonction de la qualité de l'affinité des anticorps.

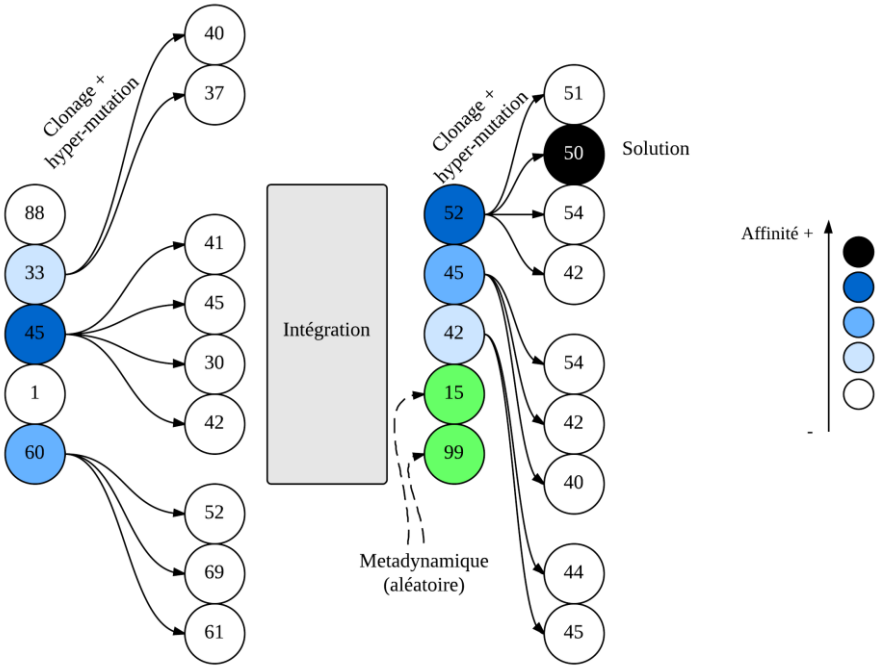


Figure 24 Recherche de la valeur 50 par l'algorithme de clonage par séléction

2.3.2.3. Les algorithmes de colonies

De même que pour le système immunitaire, des algorithmes d'optimisation se sont inspirés du comportement des colonies d'animaux et d'insectes présentes dans la nature. En effet, les différents agents composants ces colonies peuvent interagir et générer à partir de comportements individuels simples un comportement collectif d'une intelligence remarquable. Leur nombre étant considérable nous avons choisi de centrer notre étude sur la première métaheuristique parue et inspirée par ces concepts : les colonies de fourmis (Dorigo and Blum, 2005; Dorigo et al., 1996). En effet, les colonies de fourmis font preuve d'une intelligence remarquable lors de leur processus de recherche de nourriture. Cette intelligence se base sur l'émission de phéromones le long des chemins parcourus menant à une source de nourriture. Plus le chemin vers une source de nourriture sera court et plus la quantité de phéromone déposée sera importante. Chaque fourmi évaluera régulièrement la quantité de phéromones présentes afin de choisir de manière probabiliste et progressive quel chemin emprunter. La répétition de ce processus permettra de faire émerger un chemin dominant contentant la plus grande partie des phéromones. Celui-ci représentera la solution au problème. Cette méthode se base sur les analogies présentes dans le Tableau 10 et le raisonnement de l'algorithme 6.

Optimisation	Inspiration
Espace de recherche	Environnement naturel d'une fourmi
Solution	Déplacement
Représentation	Vecteur
Evaluation	Coût du déplacement
Intensification	Emission de phéromones
Diversification	Evaporation de phéromones

Tableau 10 Analalogies des colonies de fourmis

```
1  Début
2  P ← initialiserPhéromones()
3  F ← initialiserFourmis()
4  C ← initialiserChemin()
5  meilleurChemin = ø
6  Répeter
7          Pour chaque fourmi fi dans F:
8                  Ci ← ChoisirUnChemin(P)
9          Fin pour
10         Pour chaque chemin Ci dans C :
11                 évaluer(Ci)
12         Fin pour
14         Pour chaque phéromone Pi dans P :
15                 évaporationPhéromones(Pi)
16         Fin pour
18         meilleurCheminIteration ← meilleur(C)
19         Si estMeilleur(meilleureCheminIteration, meilleureChemin) :
20                 meilleurChemin ← meilleureCheminIteration
21         Fin si
22         Pour chaque phéromone Pi dans P :
23                 deposerPhéromones(Pi, meilleureChemin)
24                 déposerPhéromones(Pi, meilleureCheminIteration)
25         Fin pour
26 Jusqu'à solutionTrouvée()
27 Retourner meilleurSolution(C)
28 Fin
```

algorithme 6 Colonies de fourmis (Ant Colony Optimization)

La Figure 25 présente un exemple d'exécution de l'algorithme afin de trouver le chemin le plus court entre deux points présents sur une carte. Lors du lancement de l'algorithme chaque chemin a la même probabilité d'être choisi. Au fil des itérations, la quantité de phéromones déposées sur le chemin le plus court augmente, augmentant par la même occasion la probabilité pour celui-ci d'être choisi. À terme, on remarque que les chemins les moins utilisés ont perdu leurs phéromones par l'évaporation alors que les chemins les plus utilisés ont une quantité importante de phéromones.

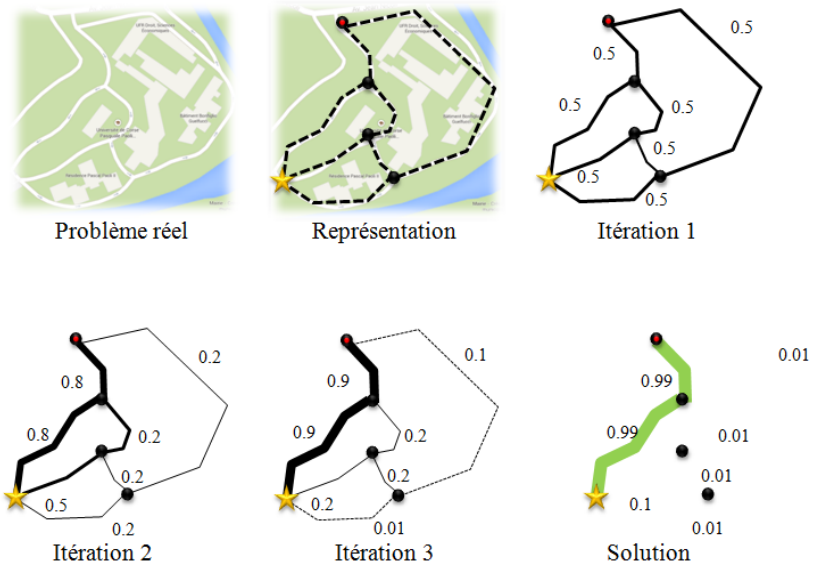


Figure 25 Exemple d'exécution de l'algorihtme d'optimisation par colonies de fourmis

2.3.2.4. Les algorithmes physiques

D'autres méthodes plus difficiles à classer se basent sur diverses branches de la physique telles que l'acoustique ou l'astronomie (Rashedi et al., 2009). En effet, la recherche d'harmonie musicale par l'étude des ondes sonores est un excellent exemple de processus d'optimisation. La création de nouvelles harmonies par les musiciens repose sur deux éléments : des connaissances (e. g. solfège, compositions connues) et des improvisations. C'est à partir de 2001 que ce processus est exploité par le monde de l'optimisation par des analogies présentes dans le Tableau 11 donnant naissance à la métaheuristique nommée « recherche harmonique » (Geem, 2009; Zong Woo Geem et al., 2001) .

Espace de recherche	Ensemble des notes et accords possible
Solution	Une composition
Représentation	Vecteur
Evaluation	Niveau d'harmonie
Intensification	Considération des harmonies précédentes
Diversification	Ajustement et improvisation

Tableau 11 Analogies de la recherche harmonique

Comme cela est explicité dans l'algorithme 7, un ensemble d'harmonies sont générées aléatoirement lors de l'initialisation de l'algorithme chacune représentant une solution potentielle. Trois paramètres sont utilisés par l'algorithme. Le premier : « taux de considération » décrit la probabilité de prise en compte des harmonies précédentes lors de la création de nouvelles harmonies. Le second : « taux d'ajustement » décrit la probabilité de perturbation vers le voisinage des notes choisies dans les harmonies précédentes. Le troisième paramètre est le taux d'improvisation, celui-ci se produit lorsque les harmonies précédentes ne sont pas prises en compte.

```
1  Début
2  nombreVariables ← définirNombreVariables()
3  nombreHarmonies ← définirNombreHarmonies()
4  HM ← initialiserHarmonies(nombreVariables, nombreHarmonies)
5  tauxConsidération ← initialiserTauxConsidération()
6  tauxAjustement ← initialiserTauxAjustement()
7  Répeter
8  nouvelleHarmonie ← ø
9  i ← 0
10         Tant que i < nombreVariables
11                 Si (tirerNombreAléatoire() < tauxConsidération)
12                         ajouterValeurDeMémoire(HM, nouvelleHarmonie)
13                         Si (tirerNombreAléatoire() < tauxAjustement)
14                                 ajusterValeurDeMémoire(HM, nouvelleHarmonie)
15                         Fin Si
16                 Sinon
17                         ajouterValeurAléatoire(nouvelleHarmonie)
18                 Fin Si
19         Fin tant que
20 Si estMeilleure(nouvelleHarmonie, plusMauvaiseHarmonie(HM))
21         remplacer(plusMauvaiseHarmonie(HM), nouvelleHarmonie)
22 Fin Si
23 Jusqu'à solutionTrouvée()
24 Fin
```

algorithme 7 Recherche Harmonique (Harmony Search)

Un exemple d'exécution de l'algorithme est présenté sur la Figure 26. Dans cet exemple l'objectif est d'obtenir un groupe de cinq couleurs proches les unes des autres. Lors de la première itération différentes harmonies sont générées de manière aléatoire puis évaluées. Une nouvelle harmonie est alors générée à partir de ces solutions potentielles. De même que les autres harmonies, celle-ci est évaluée et obtient le score de 4. Elle remplace alors l'harmonie ayant le score le plus faible (la 3^{ième}). Lors de la deuxième itération, une nouvelle harmonie est générée de la même manière. Cependant la 5^{ième} couleur de celle-ci s'inspire d'une valeur présente en mémoire sur laquelle elle effectue un ajustement. L'algorithme permet alors d'obtenir cinq couleurs proches les unes des autres et satisfait l'objectif recherché.

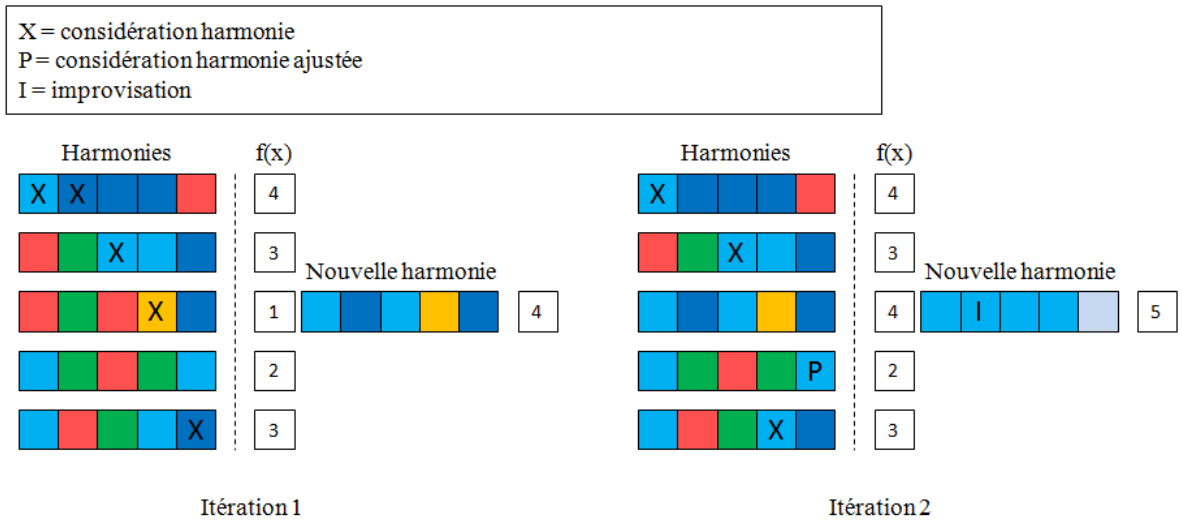


Figure 26 Exemple d'exéction de l'algorithme de recherche harmonique

2.4 Conclusion

Comme nous avons pu le voir dans ce chapitre, l'optimisation est un domaine de recherche pluridisciplinaire, très vaste et en constante évolution. Elle concerne de très nombreux problèmes et propose un nombre considérables de méthodes. Son poids dans la communauté informatique se renforce depuis une quarantaine d'années notamment par l'intermédiaire des nombreux travaux en intelligence artificielle. En effet, l'observation de la nature semble être une source intarissable de méthodes d'optimisation. Des phénomènes et des entités tels que l'évolution, le système immunitaire, les colonies d'animaux ou d'insectes, l'harmonie des ondes sonores ont d'ores et déjà été reproduits au sein de méthodes appelées « métaheuristiques ». Ces algorithmes ont été adaptés avec succès pour la résolution de problèmes d'optimisation

Cependant, ce succès présente une contrepartie. Face à la multitude d'algorithmes développés de nouvelles problématiques émergent. Quel(s) algorithme(s) choisir ? Comment évaluer précisément et rapidement les solutions proposées afin de valider l'optimisation sur des applications réelles ? La réponse à ces questions est beaucoup plus compliquée qu'il n'y parait.

Concernant le choix d'un algorithme, de nouveaux algorithmes publiés régulièrement présentent « sur le papier » des résultats qui surpassent ceux de leurs prédécesseurs. Nous assistons à une réelle course au « meilleur algorithme ». Malgré cela, la croyance en un algorithme qui surpasserait tous les autres apparait de plus en plus en plus utopique. En effet, certains algorithmes peuvent présenter des résultats inférieurs à leurs concurrents pour de nombreux problèmes et pourtant être les plus performants pour un problème de nature très spécifique. Aucun algorithme ne doit donc être ni ignoré ni glorifié.

Concernant l'évaluation, nous savons que le succès de tous les algorithmes que nous avons cités, repose sur de nombreuses évaluations des solutions proposées qui doivent être les plus précises et les plus cohérentes avec le monde réel. Il semble donc que l'outil adéquat pour réaliser ces différentes évaluations soit la modélisation du problème et sa simulation en fonction des solutions proposées. Ces concepts sont présentés dans le chapitre suivant.

Chapitre 3

De la modélisation et simulation à l'optimisation de systèmes à évènements discrets

---

« Etudie, non pour savoir plus, mais pour savoir mieux »

Sénèque

Ce chapitre présentera les principaux concepts de la modélisation et de la simulation de systèmes à évènements discrets ainsi que les intégrations antérieures de méthodes d'optimisation, en vue de situer l'approche que nous proposons.

Dans la première partie, nous présenterons les principaux concepts de la modélisation et de la simulation. Pour cela nous définirons les notions générales et les différentes entités : le système, le modèle et le simulateur. Nous aborderons également les différents processus associés à ces entités : la modélisation et la simulation.

Une fois ces concepts décrits, dans la seconde partie, notre analyse se portera sur la modélisation et la simulation de systèmes à évènements discrets que nous avons choisis d'utiliser dans le cadre de nos recherches. Nous présenterons les principales méthodes et formalismes actuellement disponibles à travers des exemples concret.

Nous nous développerons ensuite les concepts du formalisme DEVS que nous avons choisi d'utiliser. Nous expliquerons le choix de ce formalisme et présenterons son fonctionnement à travers les deux entités qu'il utilise : les modèles atomiques et les modèles couplés. Ces concepts seront illustrés dans un exemple concret. Par la suite présenterons un inventaire des principales extensions du formalisme et réaliserons un inventaire des différentes implémentations existantes.

Enfin, nous terminerons ce chapitre par une étude des différents couplages de méthodes d'optimisation avec le formalisme DEVS. Nous introduirons en premier lieu le domaine plus général de l'OvS (Optimisation Via Simulation) dans lequel ces travaux se situent. Nous proposerons ensuite une classification de ces approches en deux groupes : « les approches spécifiques » et les « approches génériques » et nous mettrons en évidence leur principaux apports et limites.

3.1 Concepts généraux

La modélisation et la simulation trouvent leur origine dans divers domaines allant de la théorie systémique, l'analyse numérique, les sciences informatiques (e. g. conception, programmation, intelligence artificielle), les mathématiques (e. g. équations différentielles) et la physique.

Comme nous pouvons le voir sur la Figure 27, modélisation et simulation sont deux processus permettant la manipulation de trois entités (Wainer, 2009) : « le système réel », « le modèle », « le simulateur » que nous allons définir.

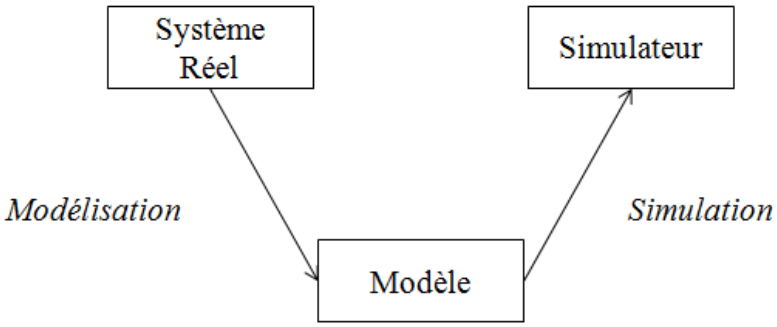


Figure 27 Modélisation et simulation

3.1.1. Le système réel

Le système réel définit une partie observée du monde réel (Cassandras and Lafortune, 2008). Différentes caractéristiques ont été mises en évidence, permettant ainsi de classer les différents systèmes réels dans plusieurs groupes en fonction de leurs caractéristiques intrinsèques comme cela est résumé dans la Figure 28.

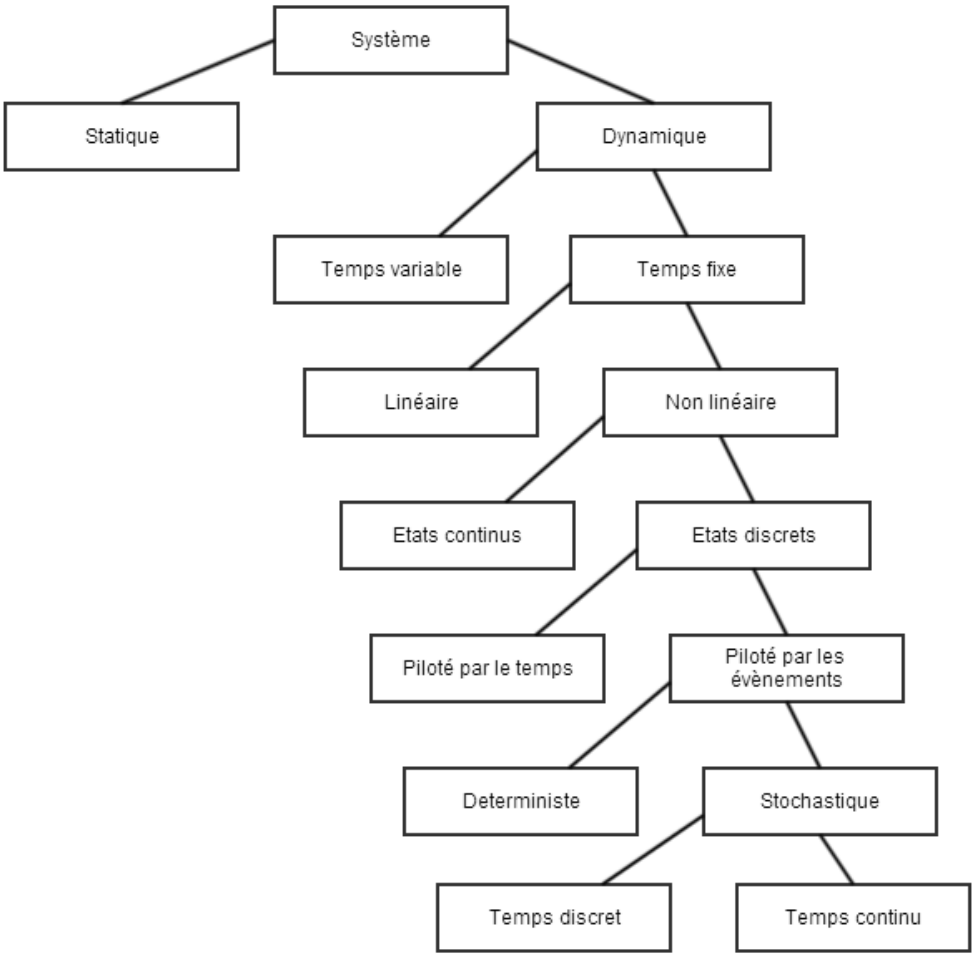


Figure 28 Classification des systèmes

Les systèmes statiques sont des systèmes pour lesquels les sorties générées sont indépendantes des entrées précédemment collectées contrairement aux systèmes dynamiques. Les systèmes dynamiques peuvent être classés en deux groupes. Le premier regroupe les systèmes sur lesquels les variations de temps n'entrainent pas de variations des états et le second, les systèmes sur lesquels des variations d'états sont visibles sur le temps. Ces systèmes dynamiques peuvent être de nature linéaire ou non linéaire. Dans le cadre des systèmes non linéaires, leurs états peuvent être continus (définis sur un intervalle de valeurs acceptées) ou discrets (définis sur une ensemble de valeurs possibles). Ces systèmes à évènements discrets peuvent être pilotés par le temps (l'état change à chaque changement de temps) ou par des évènements (l'état change uniquement lors de l'occurrence d'évènements). Les systèmes pilotés par les évènements peuvent produire des sorties identiques pour chaque entrées et être alors qualifiés de déterministes ou à l'inverse, produire différentes sorties pour des entrées toujours identiques. Ils sont alors qualifiés de stochastiques. Enfin, ces évènements peuvent se produire dans le cadre d'une représentation du temps continue ou discrète.

Parmi les systèmes réels existants, la grande majorité peut être qualifiée de systèmes complexes (Boccara, 2010; Goti, 2010). Un système complexe peut être défini comme un ensemble de composants physiques ou logiques. Chacun de ces composants possède un comportement propre et des informations limitées. Ceux-ci peuvent interagir les uns avec les autres directement ou indirectement et de manière décentralisée. C'est alors qu'ils génèrent un comportement au niveau global qui est loin de se limiter à la simple somme des comportements de ses composants. On parle alors « d'émergence » de comportement général du système.

Fortement inspirés à leur origine par la physique et la biologie, une multitude de domaines scientifiques décrivent actuellement des systèmes complexes. Parmi ces systèmes, nous pouvons citer quelques exemples tels que : la gestion des ressources hydrauliques, électriques, naturelles (pétrolières, minérales, végétales, animales), mais également les réseaux de transport (maritimes, routiers, ferroviaires, aériens) et d'échanges (internet, système nerveux).

Tout système réel peut être représenté sous la forme d'un modèle suite à une phase de modélisation.

3.1.2. La modélisation

La modélisation est un processus d'analyse et de structuration formelle du système réel en vue de créer une représentation simplifiée (Fishman, 2001) : « le modèle ». Il peut se présenter sous une forme numérique ou symbolique. Ce niveau de simplification aussi appelé « abstraction du système étudié » est dépendant de l'application et des finalités poursuivies par les futurs utilisateurs. En effet, le système réel est accessible lors de la phase d'observation. Cette observation peut être réalisée à différentes échelles. Pour cela, le comportement et les différents attributs temporels et spatiaux se doivent d'être choisis et représentés avec la plus grande rigueur sans omettre certains éléments essentiels en évitant de surcharger le modèle. Les différentes étapes du processus de modélisation sont présentées sur la Figure 29.

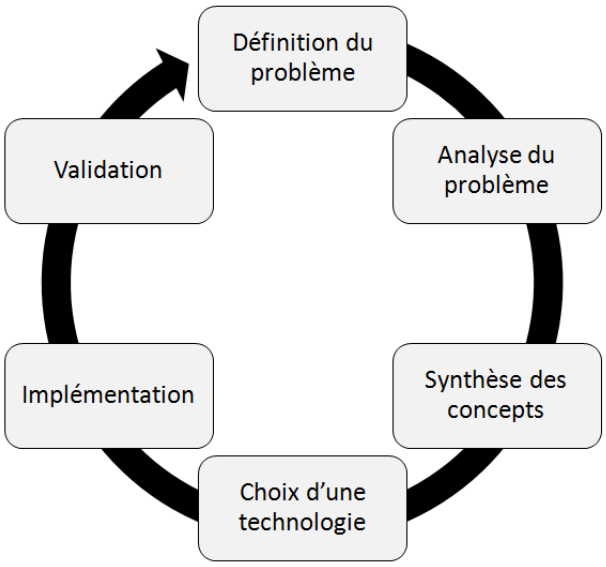


Figure 29 Etapes du processus de modélisation

La première étape : « la définition du problème » consiste à identifier les différentes réponses que le modèle devra fournir ainsi que les différents scénarios auxquels il sera soumis. La deuxième étape : « analyse du problème » a pour objectif la bonne compréhension du système étudié grâce à des documents textuels, statistiques ou encore grâce à des d'échanges verbaux avec les spécialistes des domaines concernés. Une « synthèse des concepts », pouvant être considérée comme une simplification, une abstraction du problème ou un filtrage des informations observées sera ensuite effectuée afin de recentrer la modélisation, de manière à ce que celle-ci soit cohérente avec les objectifs définis précédemment. Une fois cette étape terminée, le modélisateur devra choisir une méthode de modélisation : « choix d'une technologie ». Pour cela différents facteurs pourront être pris en compte tels que : le type d'utilisateurs, l'envergure du projet, la réutilisabilité des modèles développés, la rapidité du développement et la nature du système modélisé.

Le modèle sera ensuite conçu en respectant le format et les spécificités de la méthode de modélisation choisie. Il sera enfin validé à partir de données expérimentales collectées lors de la phase d'analyse.

3.1.3. Le modèle

Le modèle est une représentation du problème résultant du processus de modélisation (Leemis and Park, 2006). Il est une représentation simplifiée du système réel qui conserve uniquement un certain nombre de caractéristiques essentielles pour une application donnée (Zimmer et al., 2013). Sa qualité est estimée à partir de la comparaison entre les données qu'il génère lors de sa simulation et les données observables sur le système réel. Il doit être le plus possible complet et correct.

Comme cela est représenté sur la Figure 30, un modèle est caractérisé par trois éléments : des entrées, des états et des sorties. Les entrées du modèle sont les différentes conditions expérimentales auxquelles le modèle pourra être soumis lors des simulations. Les états quant à eux décrivent les caractéristiques du modèle lors de la phase de simulation et leurs éventuelles évolutions. Ils peuvent être modifiés de manière autonome ou être influencés par les entrées reçues. Les sorties du modèle sont quant à elles des informations que le modèle génère et souhaite externaliser vers d'autres modèles avec lesquels des relations sont définies.

Figure 30 Structure classique d'un modèle

Les modèles sont généralement visualisés par l'utilisateur final comme des boites noires qui produisent des sorties en fonction d'états et d'entrées diverses. Cette production est possible grâce à la phase de simulation.

3.1.4. La simulation

La simulation est le processus d'exécution du comportement du modèle. Elle permet l'évolution de son état dans le temps en fonction d'un contexte spécifique : « le scénario » (Sokolowski and Banks, 2010). Tout processus de simulation est amorcé par un processus d'initialisation durant lequel l'état initial du modèle est configuré. La simulation est avant tout un processus générateur de données. La Figure 31 présente un exemple de résultats de simulation de propagation de fumées obtenu par le logiciel « ForeFire »(Filippi and Pialat, 2013).

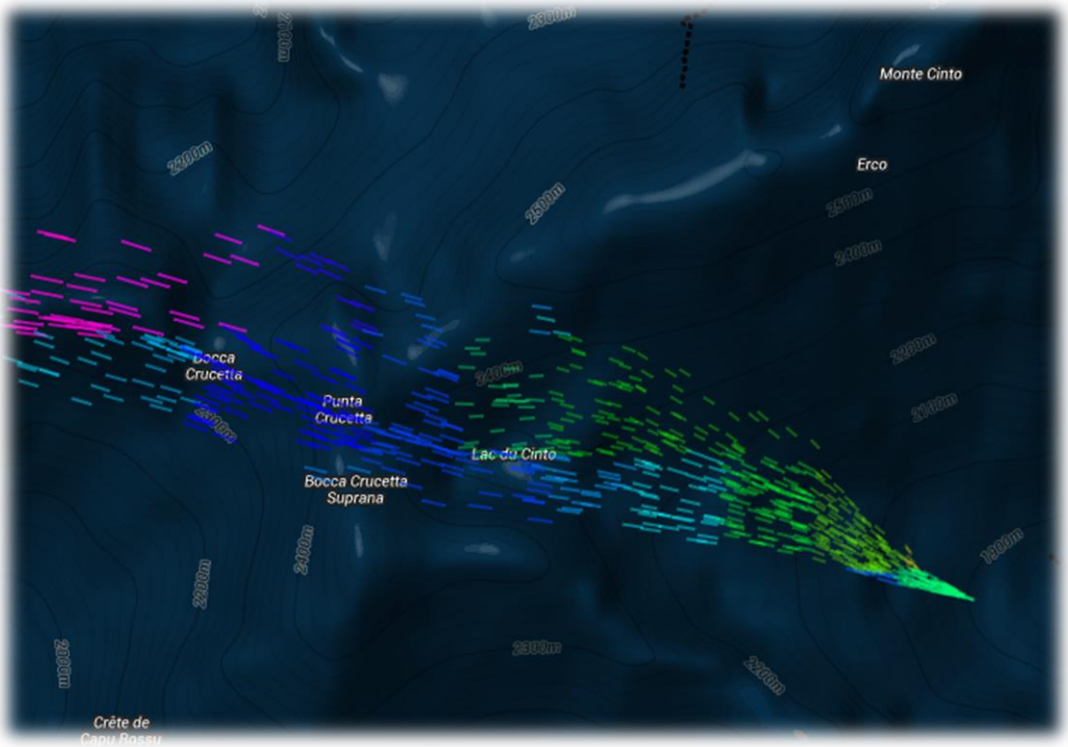


Figure 31 Simulation de la propagation de fumées d'incendi en fonction du vent et du relief

Elle offre de nombreux avantages. Elle permet d'analyser des conditions exceptionnelles sur des systèmes comme par exemple lors de la simulation de catastrophes naturelles sur une ville. Elle permet également l'observation et l'analyse de systèmes qui n'existent pas dans le monde réel. C'est le cas par exemple pour le test de prototype dans le monde aéronautique. Elle permet aussi de contourner certains risques en choisissant la simulation pour des scénarios dont l'impact économique et social peut être important. C'est le cas par exemple pour la simulation d'un incendie dans un bâtiment de grande taille. Elle permet également de s'affranchir des contraintes temporelles et spatiales inhérentes à certains systèmes. C'est par exemple le cas pour l'étude de phénomènes géologiques dont l'échelle de temps est le million d'années sur des surfaces géographiques considérables. Enfin, elle contribue à la mise en œuvre de démarches qualitatives en facilitant la détection de fautes ou d'erreurs en amont des processus de production. Son exécution est abritée par le simulateur.

3.1.5. Le simulateur

Il est important de préciser qu'un même modèle peut être simulé avec différents simulateurs. Deux types de simulations sont présents dans la littérature : les simulations continues, les simulations discrètes. Nos travaux étant centrés sur l'aide à la décision, nous nous sommes orientés vers les systèmes à évènements discrets. Dans un premier temps, ceux-ci présentent une représentation cohérente des décisions, qui par nature peuvent être considérées comme des évènements. De plus, ils permettent une modélisation rapide du problème en autorisant l'usage de différents niveaux d'abstraction. Ils facilitent les échanges entre les spécialistes du domaine étudié et les modélisateurs grâce à des représentations graphiques des différents états du système étudié. Enfin ils offrent comme autre avantage des temps d'exécution des simulations nettement inférieurs à ceux des approches continues.

3.2 Les systèmes à évènements discrets

3.2.1. Les évènements discrets

Les systèmes à évènements discrets sont pilotés par le déclanchement d'évènements dans le temps (Choi and Kang, 2013; Sokolowski and Banks, 2010). Ils se basent sur une représentation du temps et des états discrétisés ayant un ensemble limité de valeurs comme cela est visible sur la Figure 32.

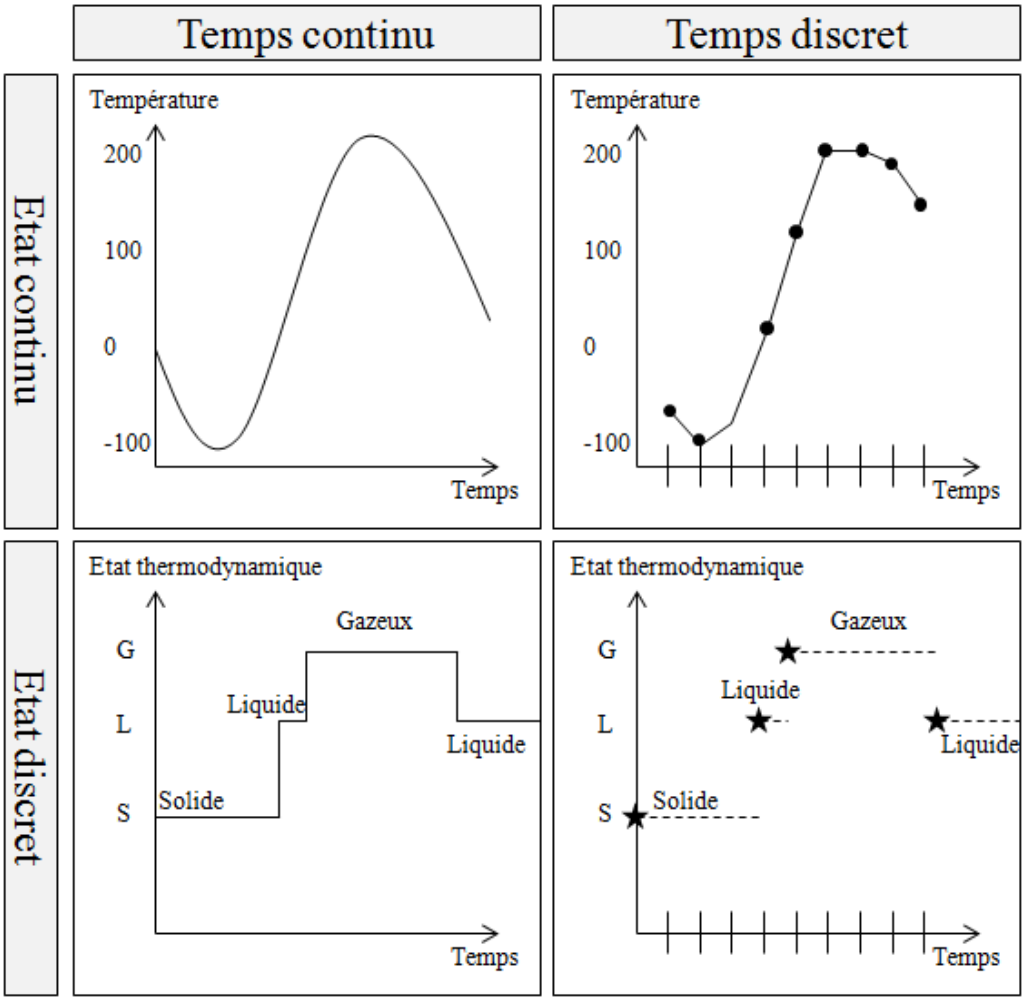


Figure 32 Représentation du temps et des états (Wainer, 2009)

Les systèmes à évènements discrets sont présents dans de nombreux domaines scientifiques. Ils sont par exemple utilisés pour la M&S de systèmes électroniques, économiques, sociologiques, physiques, chimiques et biologiques. Diverses approches sont présentes dans la littérature.

3.2.2. Les différentes approches

De nombreuses approches permettent la modélisation et la simulation. Les plus utilisées sont : les automates à état fini, les réseaux de Pétri, les systèmes multi-agents et le formalisme DEVS.

3.2.2.1. L'approche « automates à état fini »

Les automates à états finis (Hopcroft et al., 2006) peuvent être définis de la manière suivante :

Automate à état fini = (X, U, Y, F, G, x_0)

- X est un ensemble fini d'états du système
- U est un ensemble fini des différentes entrées du système
- Y est un ensemble fini des différentes sorties du système
- F représente les fonctions de transition qui d'après un état et d'une entrée produisent l'état suivant appartenant à X.
- G représente les fonctions de sortie qui d'après un état et d'une entrée produisent une sortie appartenant à Y.
- X_0 représente l'état initial du système.

La Figure 33 et le Tableau 12 illustrent les concepts énoncés ci-dessus. Cet exemple représente les différents états de l'eau X = {x1(solide), x2 (gazeux), x3 (liquide)} ainsi que les différentes transitions possibles F = {f1 (condensation), f2 (sublimation), f3 (fusion), f4 (vaporisation), f5 (liquéfaction), f6 (solidification)}. Les entrées U = {u1, u2, u3} quant à elles représentent les différentes températures auxquelles est soumises l'eau. Une seule sortie est présente dans cet exemple : y1 ∈ Y. Elle représente une fuite d'eau du système lorsque celle-ci est dans l'état gazeux.

Etat courant	Condition	Transition	Etat suivant
x_1 (solide)	$u_1 > 100$	f_2 (sublimation)	x_2 (gazeux)
x_1 (solide)	$u_1 > 0$ et $u_1 < 100$	f_3 (fusion)	x_3 (liquide)
x_2 (gazeux)	$u_2 < 0$	f_1 (condensation)	x_1 (solide)
x_2 (gazeux)	$u_2 < 100$ et $u_2 > 0$	f_5 (liquéfaction)	x_3 (liquide)
x_3 (liquide)	$u_3 > 100$	f_4 (vaporisation)	x_2 (gazeux)
x_3 (liquide)	$u_3 < 0$	f_6 (solidification)	x_1 (solide)

Tableau 12 Transition des états de l'eau

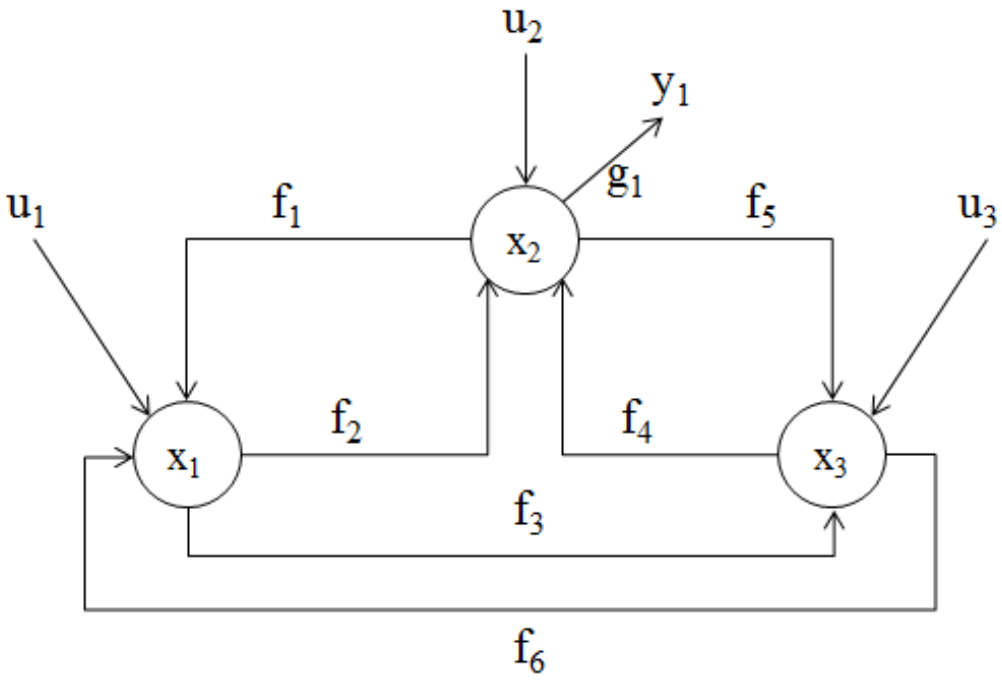


Figure 33 Modélisation des différents états de l'eau sous forme d'automate à états finis

3.2.2.2. L'approche « réseaux de Pétri »

Les réseaux de Pétri (Peterson, 1981; Cassandras and Lafortune, 2008; Reisig, 2013) se construisent également autour d'une représentation graphique du système étudié. Ils peuvent être décrits de la manière suivante :

Réseau de Pétri = (P, T, F, W)

- P est un ensemble d'états (Places) des réseaux représentés graphiquement sous la forme de cercles
- T est un ensemble de transitions d'états (Transition) représentées graphiquement sous force de rectangles allongés.
- F est un ensemble de liens (Flow relation) représentés graphiquement sous la forme d'arc orientés.
- W est un ensemble de poids attribués à chaque lien.

Durant leur simulation, différents « marqueurs » (Token) sont disposés sur les différents états et permettent ainsi de décrire l'état général du système.

La Figure 34 est un exemple de réseau de Pétri représentant le processus d'embarquement de passagers dans un aéroport. Les différents états P = {p1, p2, p3, p4, p5} représentent les différentes positions des passagers (p1 : hall de l'aéroport, p2 : salle d'embarquement, p5 : avion) ainsi que l'état de l'embarquement (p3 : embarquement ouvert ou fermé) et l'état de l'accès à l'avion (p4 : autorisé ou non-autorisé). Par défaut les poids W ont comme valeur 1 à l'exception du lien entre p2 et t6 qui a comme valeur 10. Cela signifie qu'il est nécessaire de disposer de minimum 10 passagers en salle d'embarquement et que l'équipage de l'avion soit prêt afin de permettre aux passagers d'accéder à l'avion. On remarque également grâce aux marqueurs, qu'à l'état courant : 1 passager se trouve dans le hall de l'aéroport, 3 dans la salle d'embarquement et 10 sont déjà dans l'avion et que l'embarquement est ouvert et l'équipage de l'avion prêt.

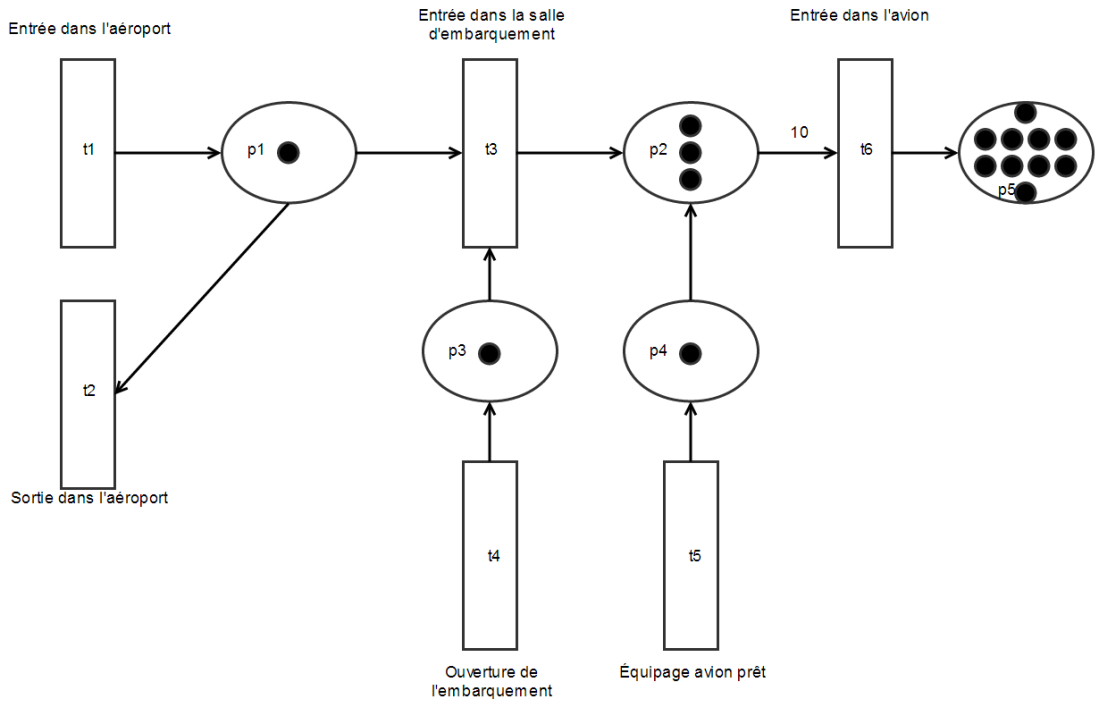


Figure 34 Exemple de réseau de Pétri (gestion de flux de passagers)

3.2.2.3. L'approche SMA (systèmes multi-agents)

Les systèmes multi-agents (Michel et al., 2009) peuvent être définis de la manière suivante :

Système multi-agents = (A, I, E, O)

- A est un ensemble d'agents ayant chacun des objectifs qui lui sont propres
- I est un ensemble d'interactions entre ces différents agents
- E est un ensemble permettant la description de l'environnement (généralement sous forme de cellules $\{E_1, E_2, \ldots, E_n\}$
- O est un ensemble de contraintes appliquées aux interactions inter-agents et agent-environnement.

Inspirée par la théorie des jeux, des processus de décision markoviens, de l'intelligence artificielle et de la stigmergie, cette approche permet la construction de systèmes complexes composés de différents agents autonomes ayant chacun des connaissances partielles du système général.

3.2.2.4. L'approche DEVS

Enfin un dernier formalisme nommé « Discrete Event System Specification » (DEVS) (Zeigler et al., 2000) permet également la modélisation et la simulation de systèmes à évènements discrets. Il propose une méthode mathématiquement formelle. Parmi les différents formalismes énoncés, le formalisme DEVS est celui qui offre la représentation des modèles la plus intuitive. En effet les systèmes modélisés peuvent être traduits facilement vers une représentation graphique et hiérarchique sous forme de boites noires interconnectées et imbriquées. Cette représentation permet de facilité les échanges entre le modélisateur et les spécialistes du système étudié. Fortement inspiré par les concepts de la programmation orientée objet, il répond ainsi à nos besoins de généricité, de compatibilité et de réutilisabilité.

De plus, le formalisme DEVS peut aujourd'hui être considéré comme un « formalisme multi paradigme ». Grace à différentes extensions et intégrations, celui-ci englobe d'autres formalismes tels que les équations différentielles (D'Abreu and Wainer, 2003) pour la simulation de systèmes continus ou encore les réseaux de Pétri (Boukelkoul and Redjimi, 2013) pour la simulation de systèmes à évènements discrets. Il se positionne à un niveau d'abstraction supérieur à celui de ses concurrents (Zeigler and Vahie, 1993; Vangheluwe et al., 2002) et permet d'effectuer des transformations d'un formalisme à un autre (Lara and Vangheluwe, 2002a). Enfin, il offre aussi la possibilité de réaliser des simulations de systèmes hétérogènes.

Toutes ces qualités nous ont conduits à choisir le formalisme DEVS pour le développement de notre outil d'aide à la décision. Celui-ci est présenté de manière approfondie dans la partie suivante.

3.3 Le formalisme DEVS

3.3.1. Présentation du formalisme

Créé durant les années 70 par le professeur Bernard Zeigler, le formalisme DEVS offre la possibilité de modéliser et de simuler des modèles informatiques et mathématiques dans un cadre formel (Zeigler et al., 2000; Vangheluwe, 2001; Zeigler, 2003; Wainer and Mosterman, 2010; Casas, 2013) et pose les bases de la théorie de la modélisation et de la simulation des systèmes à évènements discrets. Utilisés à l'origine pour l'étude de circuits électroniques, les concepts du formalisme ont rapidement été étendus à de nombreux domaines d'applications.

Le formalisme se base sur un ensemble de spécifications. Il peut être considéré comme une approche standard et universelle de modélisation et de simulation (Vangheluwe et al., 2002) dont le principal avantage est la séparation explicite entre la conception du système (modélisation), son utilisation (simulation) et son analyse (interprétation des résultats). La modélisation s'articule autour de deux catégories de modèles : les « modèles atomiques » et les « modèles couplés »(Molter, 2012). La simulation quant à elle est basée sur une arborescence de modèles et de coordinateurs construits automatiquement et sur laquelle différents messages circulent.

3.3.2. Les modèles atomiques

Les modèles atomiques permettent de décrire le comportement du système réel de manière modulaire et réutilisable. Comme on peut le voir sur la Figure 35, ce comportement est influencé à la fois par des évènements externes et internes qui pourront avoir deux conséquences : des transitions d'états ou des générations de sorties elles-mêmes considérées comme des évènements.

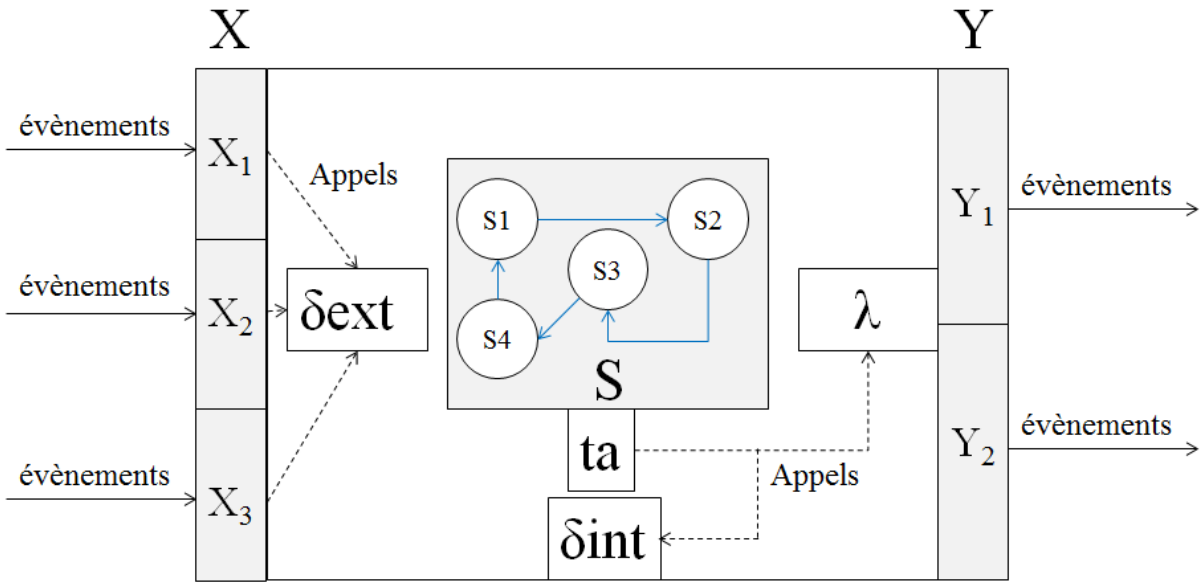


Figure 35 Modèle atomique DEVS

La définition d'un modèle atomique se présente sous la forme suivante :

$$ModèleAtomiqueDEVS = < X, Y, S, ta, \delta int, \delta ext, \lambda >$$

- X désigne l'ensemble des ports d'entrées du modèle et les valeurs d'entrées admissibles sur chacun de ces ports.
- Y désigne l'ensemble des ports de sorties du modèle et les valeurs de sorties admissibles sur chacun de ces ports.
- S désigne l'ensemble des états possibles du modèle. L'état d'un modèle est composé au minimum d'une première valeur décrivant l'état initial du modèle et d'une seconde valeur décrivant la durée de cet état.
- ta désigne la fonction d'avancement dans le temps du modèle. Elle peut être considérée comme l'horloge du modèle. C'est elle qui gère l'appel des fonctions de transitions internes et de sorties.
- δint désigne la fonction de transition interne. Celle-ci détermine le prochain état du modèle en fonction de son état courant. Elle est déclenchée lorsque le temps passé par un modèle dans un état (elapsed time) est supérieur ou égal à la valeur retournée par la fonction ta.
- δext désigne la fonction de transition externe. Celle-ci est invoquée lors de la réception d'une valeur sur l'un des ports d'entrée X.
- λ désigne la fonction de sortie. Celle-ci est appelée lorsque la durée d'un état est échue. Elle transmet alors les valeurs de sorties sur les ports de sorties Y. Son exécution terminée, la fonction δint est appelée.

Il est important de préciser que l'utilisation de modèles atomiques permet une modélisation rapide et fiable. Les modèles complexes peuvent ainsi être décomposés en sous-modèles faciles à implémenter et pouvant être facilement réutilisés. Une fois implémentés, ces modèles peuvent être testés de manière unitaire et ainsi assurer un développement de qualité en facilitant la détection et la localisation des erreurs. Une fois les différents modèles atomiques réalisés ceux-ci pourront être regroupés dans des structures nommées : « modèles couplés ».

3.3.3. Les modèles couplés

Les modèles couplés permettent de décrire la structure d'un système de manière hiérarchique sous forme d'un réseau de modèles atomiques ou couplés, interconnectés les uns aux autres. Comme cela est représenté sur la Figure 36, ces connexions forment un réseau de modèles à l'intérieur duquel circulent des messages représentant des évènements du système.

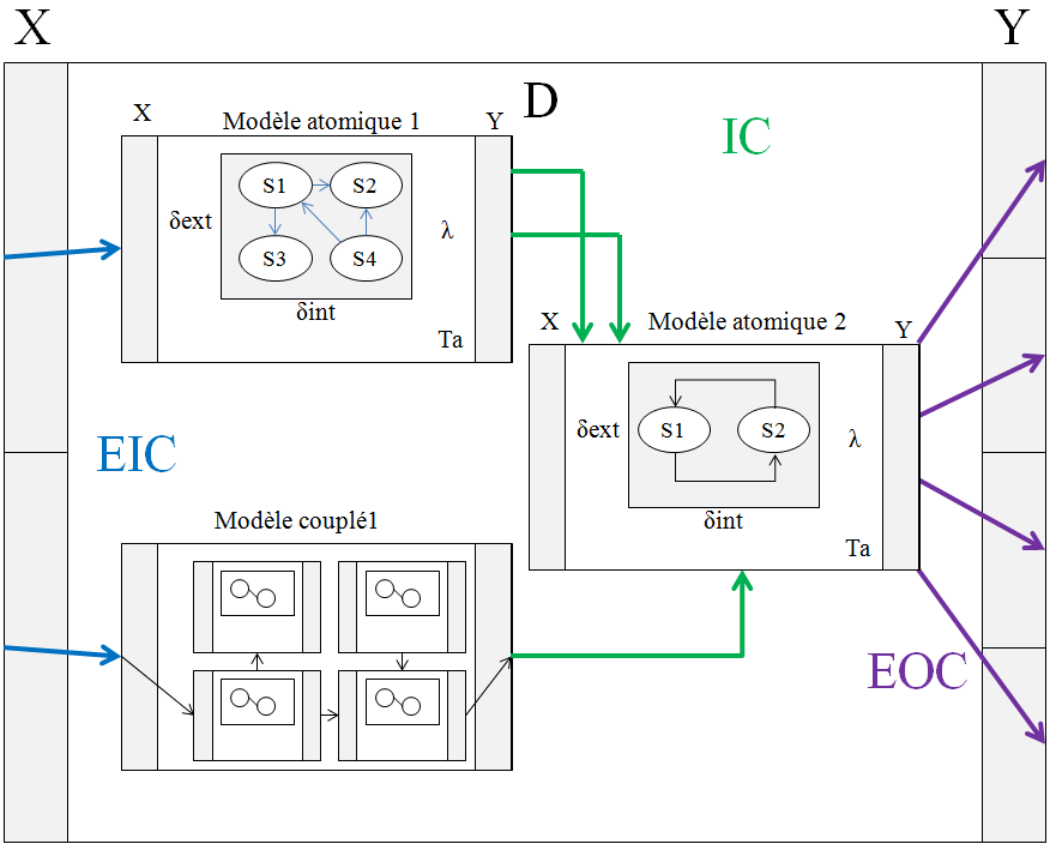


Figure 36 Modèle couplé DEVS contenant deux modèles atomiques et un modèle couplé

La définition d'un modèle couplé est décrite de la manière suivante :

ModèleCouplédDEVS = < X, Y, D, EIC, EOC, IC, select>

- X désigne l'ensemble des ports d'entrées du modèle couplé (à l'exclusion des ports d'entrées des composants du modèle)
- Y désigne l'ensemble des ports de sorties du modèle couplé (à l'exclusion des ports de sorties des composants du modèle)
- D désigne la liste des composants présents à l'intérieur du modèle couplé
- EIC désigne l'ensemble des branchements entre les ports d'entrées du modèle couplé et les ports d'entrées de ses composants.
- EOC désigne l'ensemble des branchements entre les ports de sorties du modèle couplé et les ports de sorties de ses composants.
- IC désigne l'ensemble des branchements entre les composants du modèle couplé. Chacun de ces branchements a pour origine un port de sortie du composant vers le port d'entrée d'un autre composant.
- Select est la fonction de sélection. Celle-ci est utilisée en cas de conflits entre deux modèles qui souhaitent s'exécuter en même temps et définit pour cela un ordre d'exécution. Elle se base pour cela sur une liste de priorités définissant l'ordre d'exécution des modèles.

Une fois les modèles atomiques et couplés définis ceux-ci vont pouvoir être simulés de manière automatique.

3.3.4. Simulateur DEVS

Comme nous l'avons dit précédemment le formalisme DEVS repose sur une séparation explicite de la phase de modélisation et de la phase de simulation. Le simulateur est généré de manière automatique en se basant sur la description des modèles et de leur structure comme cela est présenté dans la Figure 37.

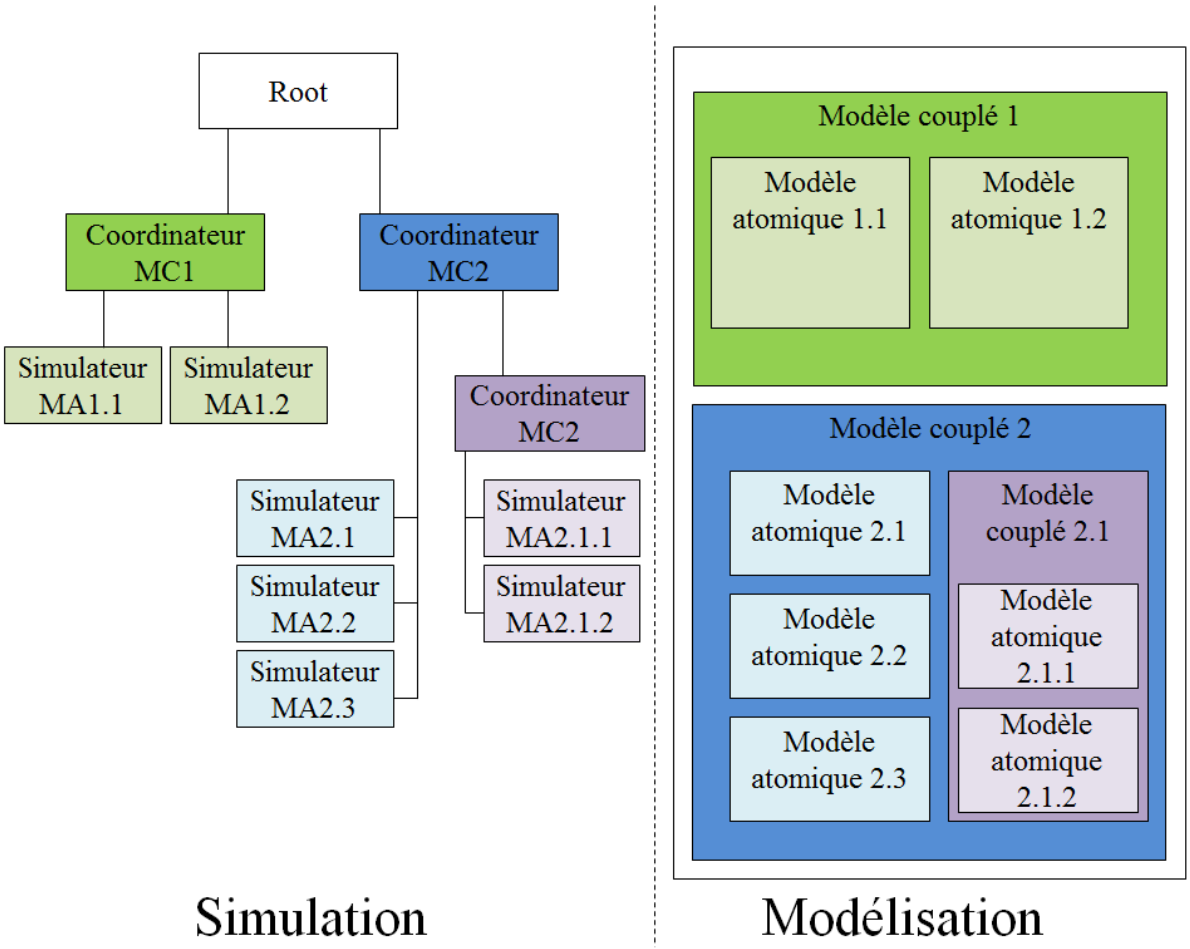


Figure 37 Correspondance simulation et modélisation DEVS

Comme nous pouvons l'observer sur la Figure 37, à partir de la description hiérarchique des modèles, une arborescence de processus est construite ayant comme coordinateur principal la racine de l'arbre : « Root ». Le formalisme isole deux groupes de processus. Le premier groupe représente les différents « simulateurs » associés à chaque modèle atomique. Le second groupe contient les différents « coordinateurs » associés à chaque modèle couplé. Il est important de préciser qu'un coordinateur pourra lui-même gérer des coordinateurs de niveau inférieur et être géré par des coordinateurs de niveau supérieur.

Durant la simulation, différents types de messages vont être échangés entre les simulateurs et les coordinateurs. Des « i messages » permettant l'initialisation des modèles, des « * messages » signalant le déclenchement d'évènements internes, des « x messages » contenant des informations liées à des évènements externes (entrées), des « y messages » contenant les sorties du modèle et enfin

des « messages d'acquittement » informant le coordinateur de niveau supérieur qu'un modèle (atomique ou couplé) a terminé son exécution.

La progression de la simulation dans le temps se base sur deux variables. La variable « TimeNext » qui contient le temps auquel le prochain évènement va être déclenché et la variable « TimeLast » qui mémorise le temps auquel le dernier évènement a été déclenché. Ces variables permettent aux différents coordinateurs de connaitre l'évènement le plus proche parmi les éléments (coordinateurs ou simulateurs) qui le composent.

Nous avons choisi d'illustrer les concepts de modèle atomique, modèle couplé et simulation par un exemple concret.

3.3.5. Exemple de modèle DEVS « Eclairage »

3.3.5.1. Modélisation du système « Eclairage »

Cet exemple présente un système de gestion d'éclairage d'une cage d'escalier. Celui-ci est composé de deux éléments : un interrupteur et une ampoule. Lorsque qu'une personne presse l'interrupteur celui-ci se positionne sur « ON » pendant une durée de 20 secondes après quoi il se repositionne en position « OFF ». Si l'interrupteur est pressé alors que celui-ci est déjà en position « ON » celui-ci reste de nouveau sur « ON » pendant 20 seconde. Si l'interrupteur est alimenté en énergie électrique il transmettra alors l'énergie à l'ampoule, à l'inverse si l'interrupteur n'est pas alimenté il ne transmettra à l'ampoule aucune énergie quelle que soit sa position.

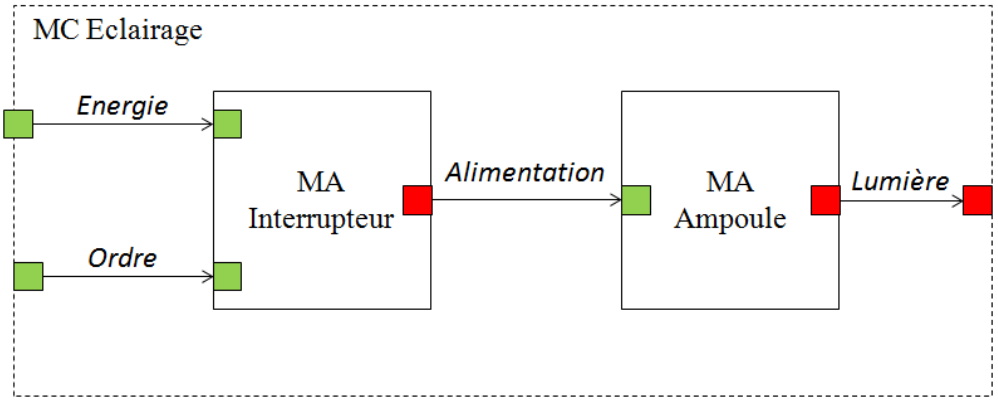


Figure 38 Modèle couplé « Eclairage »

La description du modèle atomique « MA Interrupteur » est la suivante :

Eléments DEVS	Description
X	{Energie, Ordre}
Y	{Alimentation}
S	{position ϵ {ON OFF}, tempsAllumage ϵ [0-20], alimenté ϵ {OUI, NON}}
δext (S, X)	Si message.port = X.Energie alimenté ← OUI

	Si message.port = X.Ordre position←ON tempsAllumage ← 10
δint(S)	tempsAllumage ← tempsAllumage – 10 alimenté ← NON Si tempsAllumage = 0 position ← OFF
λ(S)	transmettreEnergie ()
ta(S)	Si position = ON & alimenté = OUI Retourner ← 10 // pas de temps choisi Sinon Retourner ← Infini

Tableau 13 Description du modèle atomique « Interrupteur »

Description du modèle « MA Ampoule » est la suivante:

Eléments DEVS	Description
X	{Alimentation}
Y	{Lumière}
S	{alimenté ϵ {OUI, NON}
δext (S, X)	alimenté ← OUI
δint(S)	alimenté ← NON
λ(S)	émettreLumière()
ta(S)	Retourner 10 // pas de temps choisi

Tableau 14 Description du modèle atomique « Ampoule »

La description du modèle couplé « MC Eclairage » est la suivante :

Eléments DEVS	Description
X	{Energies, Ordre}
Y	{Lumière}
D	{MA Interrupteur, MA Ampoule}
EIC	{((MC Eclairage, Energie), (MA Interrupteur, Energie)); ((MC Eclairage, Ordre), (MA Interrupteur, Ordre))}
EOC	{((MC Eclairage, Lumière), (MA Ampoule, Lumière))}
IC	{((MA Interrupteur, Alimentation), (MA Ampoule, Alimentation))}
Select	{MA Interrupteur, MA Ampoule}

Tableau 15 Description du modèle couplé « Eclairage »

3.3.5.2. Simulation du système « Interrupteur »

Nous présentons maintenant un exemple d'exécution des modèles précédemment définis. Pour cela, la Figure 39 décrit les évènements se produisant sur les différents ports ainsi que l'état des différentes variables d'états des modèles sur une simulation d'une durée de cent secondes.

Nous raisonnons sur le scénario où l'interrupteur est pressé aux temps : t=30, t=60, t=80. Durant ce scénario, une coupure de courant est visible (t=70). Le modèle ampoule produit de la lumière à la condition unique où l'interrupteur est alimenté et en position « ON » (intervalle de temps 30-50, intervalle de temps 60-70, intervalle de temps 80-100)

Comme on peut le voir à l'intervalle de temps 70-80, l'interrupteur est bien en position « ON » cependant celui-ci n'est pas alimenté. Il en résulte une non-alimentation en énergie du modèle « AM Interrupteur » vers le modèle « AM Ampoule ». Celui-ci n'émet alors plus de lumière.

Concernant le modèle « AM Ampoule », celui-ci peut être éteint dans deux cas : non alimentation électrique de l'interrupteur ou interrupteur en position « OFF ». Cependant le système étant décrit de manière modulaire le modèle « AM Ampoule » ne connait pas la raison de sa non-alimentation. Il se contente de traiter l'énergie reçue et de la transformer en lumière.

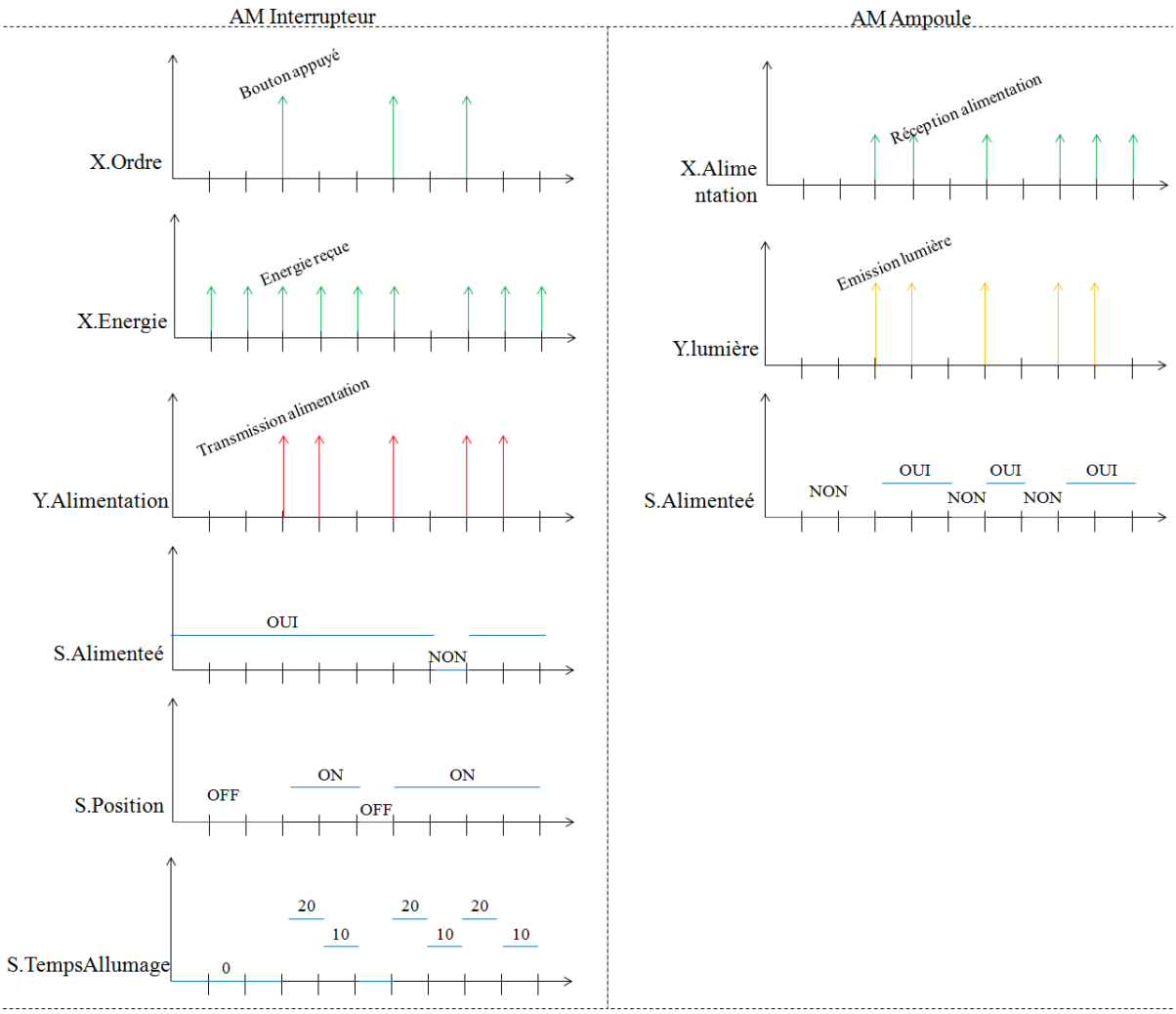


Figure 39 Simulation de l'exemple

Comme nous pouvons le voir dans l'exemple de la Figure 39, le formalisme DEVS permet une modélisation rapide. Toutefois, certains systèmes ne peuvent pas être pris en compte par le formalisme d'origine. Des extensions ont donc été apportées pour compléter le formalisme de B. Zeigler.

3.3.6. Les extensions du formalisme

Malgré une généricité incontestable du formalisme d'origine, des limites sont présentes. Des extensions ont donc été développées par différentes équipes de recherche afin d'utiliser le formalisme pour modéliser et simuler des systèmes dont la nature n'avait pas été prises en compte initialement.

Parmi elles, nous pouvons citer le formalisme « Parallel DEVS » (P-DEVS) (Chow and Zeigler, 1994) dont l'objectif est de permettre la modélisation et la simulation de systèmes en parallèle. Cette extension permet de ne plus avoir à utiliser une liste de priorités pour l'exécution des modèles qui pouvait jusqu'alors générer d'importantes disparités des résultats obtenus en fonction des priorités choisies. Cette liste de priorité est remplacée par une fonction de confluence permettant la gestion des conflits. De plus, cette extension permet de réduire le temps d'exécution total des simulations par le regroupement d'évènement se produisant simultanément dans des « bag » et en permettant de répartir la charge sur différentes ressources.

A partir de celle-ci une autre extension : « Cell-DEVS » (Liu and Wainer, 2007; Ntaimo and Khargharia, 2006; Wainer and Giambiasi, 1997, 2001) a été développé. Cette extension quant à elle est utilisée pour des systèmes pouvant être représentés sous la forme de grilles et ce de manière beaucoup moins fastidieuse qu'avec le formalisme d'origine avec lequel le voisinage de chaque cellule devait être configuré manuellement par le modélisateur et simulé de manière séquentielle. Pour cela, le système est défini comme une grille de cellules interconnectées, chacune de ces cellules étant un modèle atomique. Cette extension est fortement liée au formalisme P-DEVS. En effet, celui-ci permet une gestion parallèle des cellules plus cohérente avec le monde réel dans lequel plusieurs cellules peuvent s'activer simultanément. Cette extension a notamment permis l'intégration d'automates cellulaires au formalisme (Wainer et al., 2010).

D'autres extensions permettent des changements de structures des modèles durant la simulation telle que le formalisme « Dynamic Structure Discrete Event System Specification » (DSDEVS) (Barros, 1995). Ce dynamisme apparait lors de l'ajout, de la suppression et de la modification de liens et de modèles. Pour cela, l'arbre de simulation repose sur une structure dynamique pouvant évoluer dans le temps (Hu et al., 2005).

Il existe également une extension « Real-Time DEVS » (RTDEVS) (Hong et al., 1997; Moallemi et al., 2010). Comme son nom l'indique celle-ci permet de simuler le système en fonction du temps réel contrairement aux autres extensions qui elles s'exécutent sur un temps logique. En d'autres termes, cette extension permet de simuler un système à une échelle de temps identique à la réalité et ainsi faciliter la validation en direct par l'observation des évènements simulés et observés.

Le formalisme « Stochastic DEVS » (STDEVS) (Castro and Kofman, 2006; Castro et al., 2010) permet la modélisation de systèmes stochastiques contrairement au formalisme initial dont le raisonnement se base sur des opérateurs déterministes. Grâce à ce formalisme, les notions d'incertitude, imprévisibilité et d'espace de confiance des événements peuvent être simulées. Pour cela l'auteur se base sur la théorie des probabilités.

Les formalismes « Fuzzy DEVS » (FDEVS)(Kwon et al., 1996) et « Imprecize DEVS » (I-DEVS) (Bisgambiglia et al., 2009) permettent également l'intégration des notions d'incertitude et d'imprévisibilité. Cependant ils ne se basent pas sur la théorie des probabilités mais sur celle des ensembles flous.

Le formalisme « Symbolic DEVS » (SDEVS) (Zeigler and Chi, 1992) quant à lui propose une représentation symbolique du temps. Ce concept permet une analyse approfondie des modèles et permet de tracer l'ensemble des transitions possibles ou nécessaires pour parvenir à un état donné. Il est essentiellement utilisé pour la détection de fautes et la simulation de scénarios impliquant des contraintes de temps.

Enfin l'extension « G-DEVS » (Giambiasi et al., 2001; Kofman and Junco, 2001; Zeigler and Lee, 1998) permet la modélisation et la simulation de systèmes complexes continus. Il se base sur l'intégration des équations différentielles et la mise en œuvre d'un « quantizer ». Le rôle de celui-ci est de limiter le nombre d'évènements en fonction d'une valeur mesurant l'ampleur d'un changement d'état appelé « Quantum ». Seuls les évènements générant des changements d'états significatifs sont déclenchés ce qui permet de réduire la charge de calcul sans pour autant compromettre le niveau de précision des résultats choisis.

Enfin d'autres extensions moins connues sont présentes dans la littérature telles que « Multilayered DEVS » (Broutin et al., 2010; Zeigler, 1984) permettant la modélisation et la

simulation de systèmes à différentes échelles (états, temporalité) ainsi que « Concurrent DEVS » (Capocchi et al., 2005) permettant de réaliser des simulations comparatives et concurrentes sans pour autant nécessiter la mise en parallèle des processus de simulation.

Au-delà de ces différentes extensions, le formalisme DEVS a conduit à la conception d'un langage graphique de méta modélisation nommé « System Entity Structure » (SES) que nous présenterons dans la partie suivante.

3.3.7. Système entité structure

Dans l'objectif de permettre une conception performante et simple du système étudié, le formalisme DEVS s'est enrichi d'une représentation graphique des modèles nommée « System Entitiy Structure » (SES) (Zeigler and Hammonds, 2007). Les arbres générés permettent la description modulaire et hiérarchique des différents modèles qui composent le système à modéliser. Ce langage graphique se structure autour de différents axiomes que sont : « les entités », « les aspects », « les aspects multiples », « les spécialisations » et enfin « les variables ». Les « entités » représentent un composant (modèle atomique ou couplé) du système étudié sous la forme de « variables » typées et pouvant être bornées. Les « aspects » quant à eux définissent une relation hiérarchique de composition unitaire entre un nœud parent et un nœud enfant. Les « multi-aspects » représentent des relations hiérarchiques multiples entre un nœud parent et plusieurs nœuds enfants. Enfin les « spécialisations » permettent la représentation d'un choix alternatif ou hiérarchique d'un nœud parent vers un unique nœud enfant.

De par sa structure fortement hiérarchique, les modèles représentés par le formalisme SES sont très facilement convertibles au format XML et permettent ainsi un haut niveau de standardisation. De plus, son utilisation permet de simplifier la structure du modèle en appliquant des méthodes d'élagage (pruning) facilitant la définition des modèles couplés ainsi que la hiérarchie générale du modèle.

Un exemple d'utilisation du SES est présenté sur la Figure 40. Cet exemple permet de réaliser le méta modèle d'un avion. Celui-ci se compose de trois « entités » : les ailes, la cabine et les soutes. Dans cet exemple nous pouvons voir que les ailes ont une « spécialisation ». Elles peuvent supporter une propulsion à hélices ou une propulsion à réacteur. La cabine quant à elle peut avoir deux configurations : « Cargo » et « Tourisme ». Lorsqu'elle est spécialisée pour du tourisme, elle dispose de « multiples » sièges. Ce n'est pas le cas dans la configuration « Cargo ». Le composant « Soutes » quant à lui dispose de plusieurs compartiments pouvant être configurés soit pour recevoir des chargements conteneurisés soit des chargements en vrac.

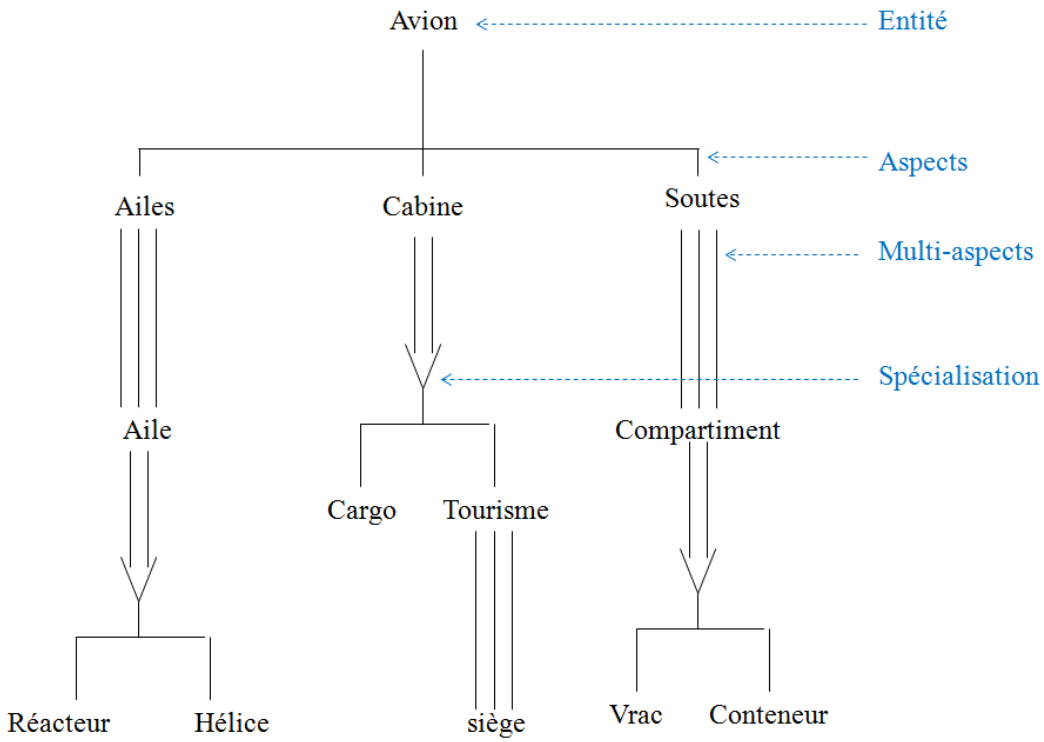


Figure 40 Exemple de SES pour la modélisation d'un avion

Une fois la phase de conception des modèles terminée ceux-ci peuvent être implémentés grâce à des outils logiciels ou encore des libraires spécifiques au formalisme DEVS.

3.3.8. Logiciels et librairies DEVS

Fort de son succès (Sarjoughian and Cellier, 2001), ce formalisme s'est rapidement imposé dans la communauté scientifique dans le monde entier et a permis la modélisation et la simulation de nombreux systèmes. Son utilisation est possible dans différents logiciels référencés dans le Tableau 16. Il est également accessible dans plusieurs librairies présentées dans le Tableau 17.

Logiciels	Pays	Licences	Spécificités
DEVSimPy (Capocchi et al., 2011)	France	Open source	Collaboratif Concurrent Multi-échelle Dynamique Choix de l'algorithme de simulation Plugins Librairies de modèles Visualisation avancée Web-services
VLE (Virtual Laboratory Environment) (Quesnel et al., 2009)	France	Libre	Multi-formalisme Systèmes dynamiques Aide à la décision Apprentissage
MS4ME (Modeling and Simulation for me) (Zeigler and Sarjoughian, 2012)	Etats-Unis	Propriétaire	Collaboratif Fouille de données Visualisation avancée « Store » de modèles
Atom 3 (A Tool for Multi-formalism and Meta-Modeling) (Lara and Vangheluwe, 2002b)	Canada	Open source	Multi-paradigme Meta-modélisation Modèle transformation Programmation linéaire (solveur)
Power DEVS (Bergero and Kofman, 2011)	Argentine	Open source	Systèmes hybrides Simulation temps réel
DEVSim++ (Kim, 1994)	Canada, Corée du Sud	Open source	Temps réel

JAMES II (JAva-based Multipurpose Environment for Simulation II) (Himmelspach and Uhrmacher, 2007) (Suite James II)	Allemagne	Open source	Multi-formalisme Plugins Optimisation via simulation Choix de l'algorithme de simulation Parallélisation Fouille de données Rapports de simulation Création de langages de modélisation
SmallDEVS (Janoušek and Kironskỳ, 2006)	République Tchèque	Open source	Simulation interactive Réutilisation des modèles Choix de l'algorithme de simulation

Tableau 16 logiciels DEVS

Librairies	Pays	Langage
DEVS-JAVA (Zeigler and Sarjoughian, 2003)	Etats-Unis	JAVA
DEVSPython (Bolduc and Vangheluwe, 2001)	Canada	Python
CD++ (Wainer, 2002)	Carleton University & Universidad de Buenos Aires	C++
DEVSim++ (Kim et al., 2011)	Corée du Sud	C++
ADEVS (Nutaro, 2011)	Etats-Unis	C++
DEVS/C++ (B.P. Zeigler et al., 1996)	Etats-Unis	C++
JDEVS (Filippi and Bisgambiglia, 2004)	France	JAVA
DEVS C# (Hwang, 2007)	Etats-Unis	C# (.NET)

DEVS-Ruby (Franceschini Romain and Bisgambiglia, 2014)	France	Ruby

Tableau 17 Librairies DEVS

Le formalisme DEVS, ses différentes extensions, ses logiciels et librairies ont été utilisés dans le cadre de processus d'optimisation via simulation. Les principes d'intégration de la simulation dans les processus d'optimisation ainsi que les différentes approches DEVS sont présentées dans la partie suivante.

3.4 DEVS et l'optimisation

3.4.1. Principes de couplage de l'optimisation et de la simulation (OvS)

Les différentes méthodes d'optimisation que nous venons de présenter dans le chapitre précédent ont été appliquées à de nombreux problèmes modélisés et simulés. Ce succès s'explique à la fois par la possibilité offerte par la simulation d'étudier rapidement de nombreux scénarios et par la multitude de systèmes modélisés disponibles.

Le nombre considérable de travaux réalisés a permis l'émergence d'un domaine de recherche à part entière nommé : « Optimisation via Simulation » (OvS) (Fu et al., 2000). L'objectif de l'OvS est de trouver la ou les solutions optimales en simulant le problème à optimiser. En ce sens, elle peut être considérée comme un lien entre les deux domaines de recherches que sont : « l'optimisation » et « la modélisation et la simulation ». Comme nous pouvons le voir sur la Figure 41, pour chacun de ces deux domaines de très nombreux concepts sont disponibles et peuvent être combinés. Des travaux ont permis de mettre en évidence six catégories distincts de méthodes (Fu et al., 2005).

La première catégorie regroupe les approches de « classement et sélection » (Ranking & Selection). Ces approches sont utilisées pour des applications dont l'ensemble des alternatives possibles peuvent être simulées dans des temps raisonnables.

La seconde catégorie regroupe les approches de « surface de réponse » (Response Surface Methodology). Ces méthodes se basent sur des outils statistiques tels que les réseaux de neurones. Leur objectif est de trouver par approximation, une fonction de relation optimale entre les entrées et les sorties de la fonction d'évaluation.

La troisième catégorie regroupe les procédures basées sur la notion de « gradient » (Gradient-Based Procedure) et « d'approximation stochastique » (Stochastic Approximation). Elles utilisent des analyses par perturbations et diverses procédures d'estimation du gradient.

Le quatrième groupe réuni les procédures basées sur la « recherche aléatoire » (Random Search). Celles-ci exploitent les notions de voisinage et de parcours probabiliste de l'espace de recherche.

Le cinquième groupe quant à lui, rassemble les méthodes dites de « parcours par échantillonnage » (Sample Path Optimization). Ces méthodes effectuent des ensembles de simulations afin de générer un échantillon de résultats. Une fois cet échantillon obtenu elles tentent d'estimer à partir d'outils mathématiques quelles sont les meilleurs solutions possibles.

Enfin, le sixième et dernier groupe couvre les méthodes dites « Métaheuristiques » (April et al., 2003). Ces méthodes d'optimisation également appelées méthodes modernes permettent l'optimisation par la simulation de problème NP-complets et NP-difficiles, pour lesquels le nombre de combinaisons de variables est considérable et qui engendreraient des temps d'exécution avec les autres méthodes pouvant aller jusqu'à plusieurs mois.

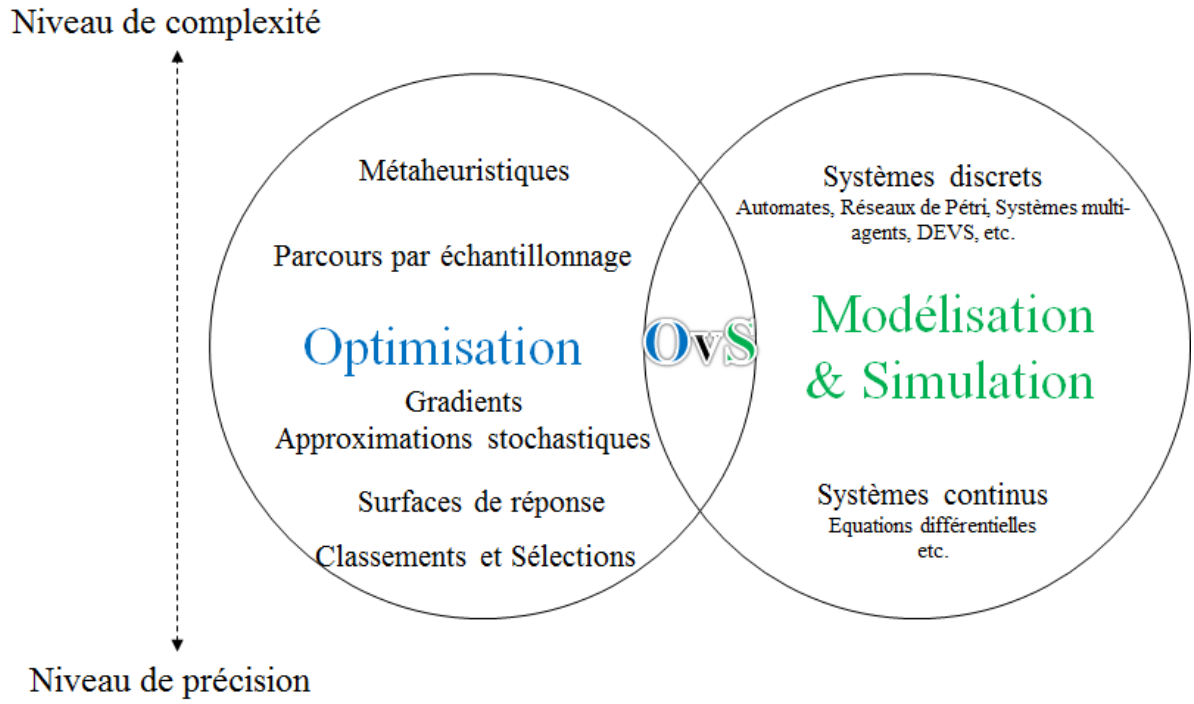


Figure 41 L'optimisation via simulation (OvS)

Chacun des deux domaines qui composent l'OvS, possède sa propre terminologie. Comme nous pouvons le voir sur la Figure 42, du point de vue de l'optimisation, les données émises sont des « solutions », les données reçues des « évaluations » et l'entité manipulée le « problème ». Du point de vue de la modélisation et la simulation les données reçues sont des « entrées », les données générées sont des « sorties » et l'entité traitée est le « système ».

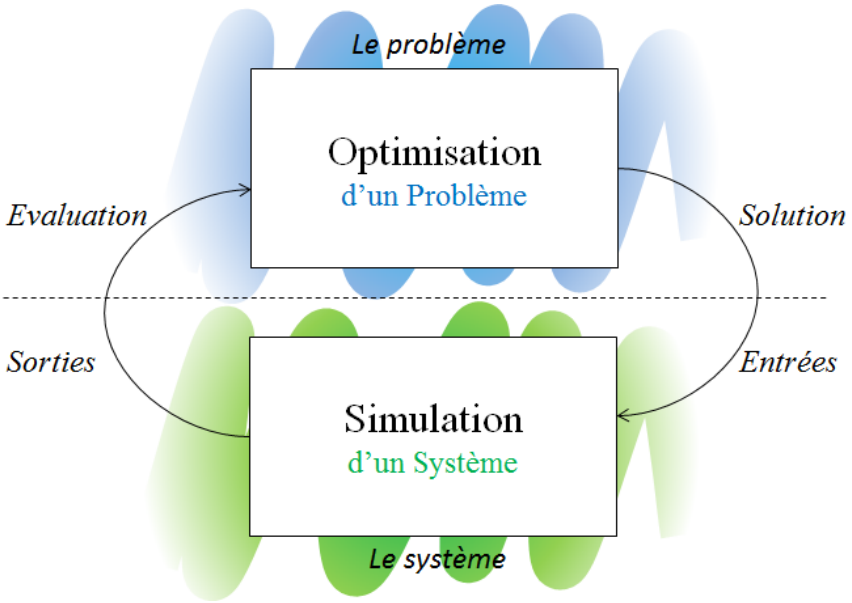


Figure 42 Terminologie de l'OVS

Un certain nombre de travaux d'OvS de système complexes se sont basés sur l'utilisation du formalisme DEVS et de différentes méthodes d'optimisation. Ces différents travaux sont présentés dans la partie suivante.

3.4.2. Approches applicatives DEVS

Dès 1996 B. Zeigler utilise les métaheuristiques dans le cadre de la modélisation d'un système d'infiltration. Dans cette application l'usage des métaheuristiques permet d'optimiser la représentation d'ensembles flous à partir d'une phase préalable d'entrainement (B. P. Zeigler et al., 1996).

Durant l'année 2008, plusieurs travaux intègrent des méthodes d'optimisation à DEVS. L'auteur (Lee et al., 2008) propose l'utilisation du formalisme DEVS et de techniques du type « branch and bound » pour l'optimisation des paramètres d'une passerelle réseau de type « Link-11 » (Lee et al., 2008). A travers l'utilisation d'un générateur intelligent, différentes configurations sont générées puis simulées de manière parallèle et permettent d'obtenir la meilleure taille de trame en fonction de scenario précis. La même année (Ntaimo et al., 2008) un outil : DEVS-FIRE est proposé. Celui-ci s'inspire du formalisme CELL-DEVS et permet la simulation d'incendie et l'optimisation des stratégies de lutte mises en œuvre (e. g. minimisation de la quantité de ressources utilisées et maximisation de leur efficacité). L'incendie est simulé à partir d'une description géographique et météorologique de la zone. La technique d'optimisation utilisée se base sur la méthode « Vost Plus Net Value Change» (C + NVC) inspirée elle-même par les concepts de la programmation linéaire. En 2009 les extensions du formalisme DEVS : P-DEVS (Parallel - Discrete Event System Specification) et DEVS/SOA(Discrete Event System Specification / Service Oriented Architecture)[1] (Mittal et al., 2009) sont proposées pour exécuter de manière parallèle et décentralisée deux algorithmes d'optimisation multi-objectifs : « NSGA-II » et « SPEA2 » (Risco-Martín et al., 2008). L'application consiste à optimiser le typage d'un nombre important de variables et ainsi permettre de minimiser les accès en mémoire pour des équipements mobiles dont la mémoire et l'autonomie énergétique sont limitées.

En 2011, des travaux proposent l'utilisation d'algorithmes génétiques pour l'optimisation d'ensembles flous de modèles respectant le formalisme « DEVSFIS » (Discrete Event System Specification Fussy Inference Systems) (Bisgambiglia et al., 2011). Deux ans plus tard, l'utilisation du formalisme DEVS et la modélisation de techniques d'optimisation sont proposées pour l'optimisation de la gestion d'un réseau hydraulique. L'objectif de cette recherche est de permettre le choix de dates optimales pour les différents modes de fonctionnement du réseau et ce, en vue d'optimiser la production d'une microcentrale électrique qui génère des revenus financiers mais aussi de minimiser le temps de pompage entre les différents sites qui génère des coûts d'exploitation importants (Capocchi and Santucci, 2013).

3.4.3. Des approches génériques DEVS

3.4.3.1. L'utilisation de cluster

Basé sur le concept de « diviser pour mieux régner », J. Kim et B. Zeigler envisagent dès 1996 l'utilisation de la modélisation et de la simulation pour l'optimisation de problèmes complexes (Kim and Zeigler, 1996). A partir d'une conception nommée « Hierarchical Distruted Genetic Algorithms » (HDGA) l'espace de recherche est analysé à différentes échelles formant une hiérarchie de clusters eux-mêmes répartis dans différentes couches comme cela est présenté sur la Figure 43. Comme on peut le voir sur la Figure 44, les couches de haut niveau ont une vision d'ensemble mais peu précise de

[1] L'article était alors en processus de relecture et n'est pas paru que l'année suivante.

l'espace de recherche. Leur rôle consiste à sélectionner différentes sous zones à fort potentiel et de transmettre la recherche sur ces zones sélectionnées en créant des clusters dans la couche de niveau inférieur. En d'autres termes, les couches de haut niveau permettent la diversification de la recherche alors que les couches de bas niveau permettent l'intensification en parallèle des recherches.

Bien que cette approche permette de réduire considérablement les temps d'exécution de l'optimisation, elle se focalise sur une unique méthode d'optimisation dont le paramétrage reste difficile à gérer. Il en est de même pour la création des différents segments de l'espace de recherche étudié. De plus, elle nécessite un important travail d'intégration du problème étudié.

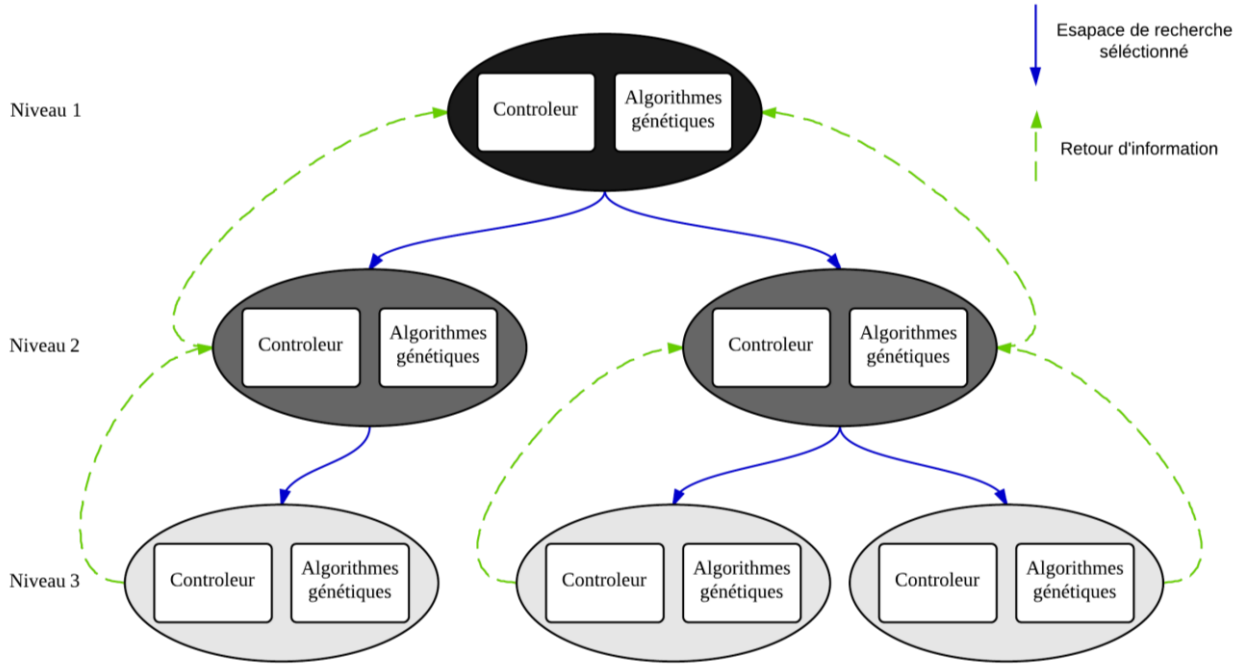


Figure 43 Conception DEVS Hierarchical distributed genetic algorithms

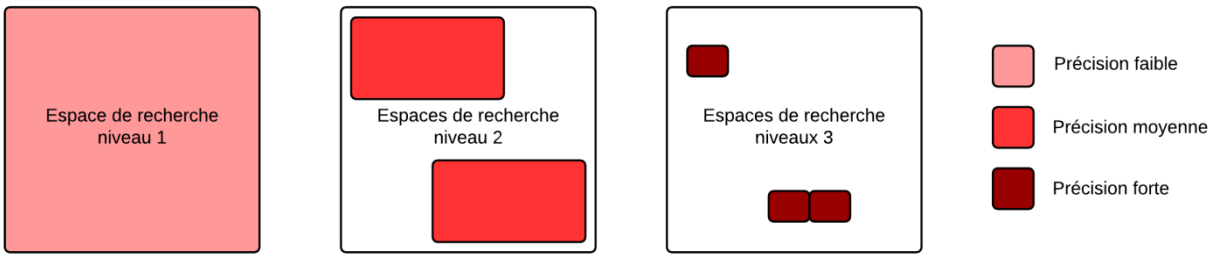


Figure 44 Segmentation de l'espace de recherche

3.4.3.2. L'utilisation d'un tableau noir

En 2003, une nouvelle approche est proposée (Iassinovski et al., 2003). Elle se base sur la création d'un outil unique d'aide à la décision intégrant la modélisation et la simulation de modèles DEVS. À travers un langage unique, l'auteur permet l'utilisation de différentes méthodes d'optimisation : des méthodes stochastiques (métaheuristiques) et des méthodes exactes (séparation et évaluation). Pour cela il est proposé la création d'une zone de mémoire partagée sur laquelle les échanges entre outils d'optimisation, de simulation et visualisation sont centralisés. Un exemple

d'application est donné pour la gestion des commandes pour différents sites de distribution géographiquement répartis. Grâce à l'aspect de modélisation hiérarchique offert par le formalisme DEVS, trois niveaux sont définis. Au premier niveau, les algorithmes génétiques sont utilisés afin d'optimiser la répartition des commandes sur les différents sites. Au second niveau, une heuristique est utilisée pour la gestion des commandes et leurs attributions aux différents véhicules de livraison. Enfin au troisième niveau l'élaboration du planning de transport utilise des méthodes exactes. Le concept de hiérarchie des modèles du formalisme DEVS permet donc ici l'utilisation de méthodes d'optimisation hybrides.

Tout comme l'approche précédente, celle-ci repose sur une configuration statique de l'optimisation. Elle implique également de fortes dépendances entre la modélisation du problème et son optimisation.

3.4.3.1. Optimisation structurelle

En 2009, O. Hagendorf propose d'élargir le champ d'application de l'optimisation par la simulation dans le formalisme DEVS (Hagendorf, 2009). Pour l'auteur, l'optimisation ne doit pas se limiter à un aspect uniquement paramétrique. En effet, certaines applications nécessitent de nombreuses modifications du modèle simulé en vue de répondre aux finalités recherchées. Pour permettre l'optimisation paramétrique et structurelle, une nouvelle architecture est proposée. Celle-ci est visible sur la Figure 45. Elle se décompose en trois modules : « le module de contrôle », « le module de modélisation et simulation » et « le module d'optimisation ».

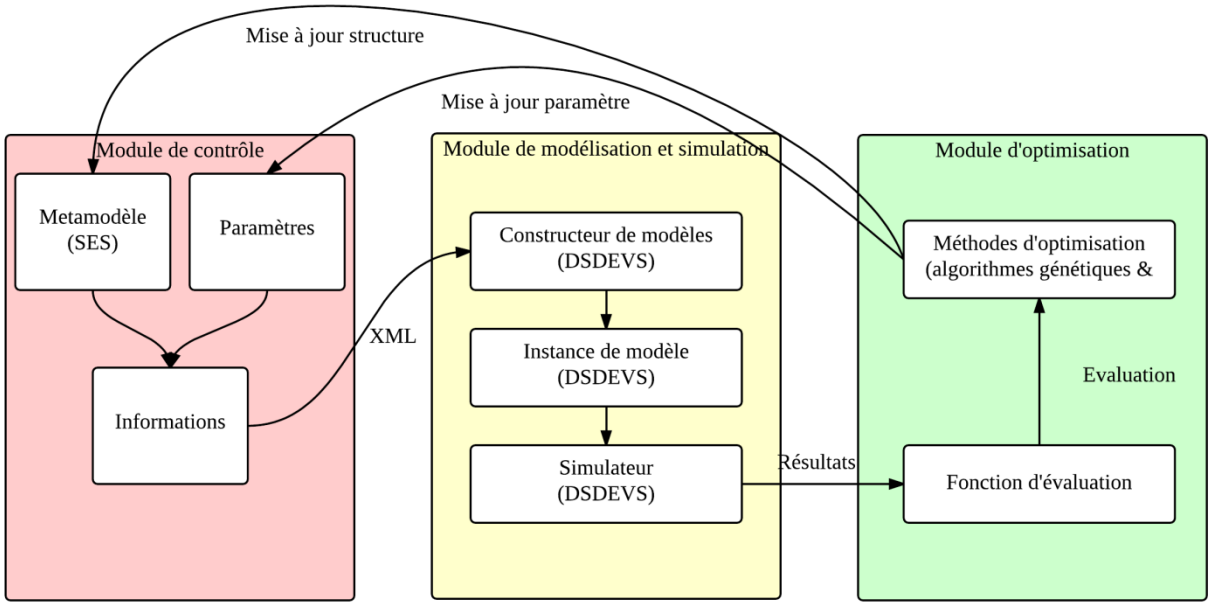


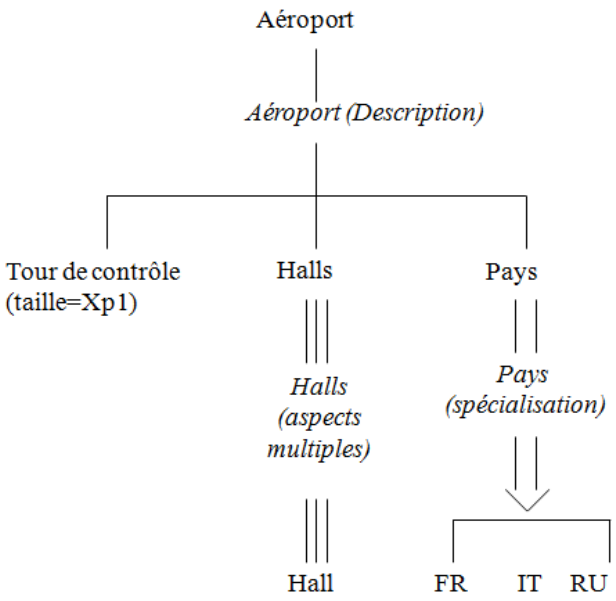
Figure 45 Optimisation paramétrique et structurelle (HagenDorf)

Le module de contrôle contient la représentation du système étudié sous la forme d'un méta-modèle. Ce méta-modèle est décrit à partir du format graphique de conception « System Entity Structure » SES proposé par B. Zeigler. Il se présente sous la forme d'un arbre représentant la hiérarchie des modèles composant le système étudié. Comme cela est présenté sur la Figure 46, lors d'un parcours en largeur ou en profondeur, l'arbre proposé va être traduit sous la forme d'un vecteur de nombres en vue d'être compatible avec le module d'optimisation. Lors de cette phase de traduction, deux espaces de recherche vont être générés. Le premier décrivant les différents paramétrages possibles et le second décrivant les différents choix de structures possibles. A partir de ces deux

ensembles, des propositions vont être élaborées comme cela est visible dans l'exemple présenté sur la Figure 47.

Une fois que le modèle contenant la description de la structure et les différents paramètres est terminé, celui-ci va être transcrit au format XML vers le module de modélisation et de simulation. Il sera alors instancié grâce à l'extension DSDEVS permettant alors l'utilisation de modèles DEVS dynamiques. Une fois cette étape terminée la simulation du modèle proposé sera exécutée et génèrera des sorties.

Cette approche est la première à proposer une optimisation structurelle de systèmes modélisés dans le formalisme DEVS et présente un réel intérêt. Cependant elle se focalise sur l'aspect modélisation et simulation sans avoir approfondi l'optimisation qui une fois de plus, est statique et restreinte à un seul type d'algorithme.



Variables (Xp)
Xp1 ϵ DomaineTaille | DomaineTaille=[0,100]
Structures (Xs)
Xs1 ϵ NombreHall | DomaineNombreHall = {1,2,3,4,5,6} Xs2 ϵ DomainePays = {1,2,3}
X = (Xp, Xs)

Figure 46 Métamodèle SES « aéroport »

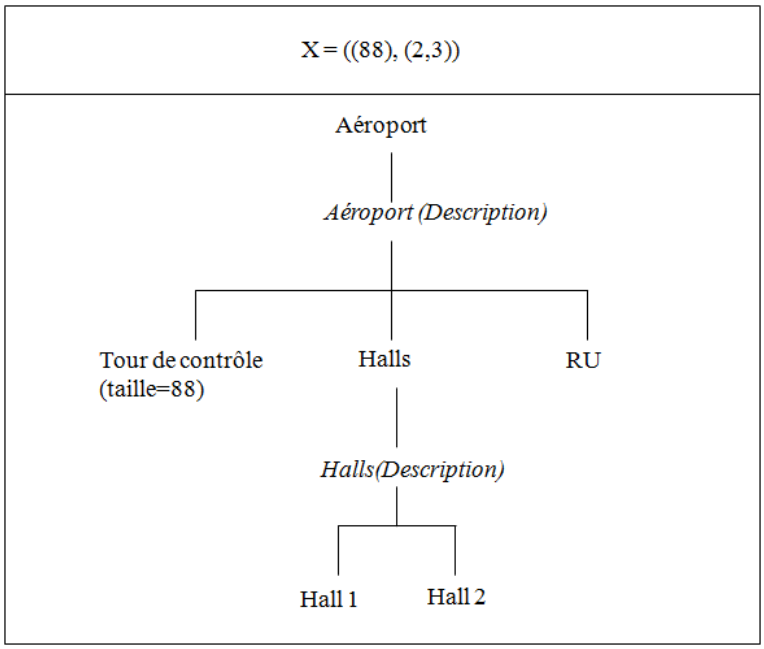


Figure 47 Exemple d'instance du métamodèle « aéroport »

3.4.3.2. Le framework SIMEON

En 2011 deux auteurs (Halim and Seck, 2011) proposent un espace de travail nommé « SImulation-based Multi-objective Evolutionnary OptimizatioN » (SIMEON) permettant l'optimisation de problèmes multicritères représentés sous la forme de modèles DEVS. Quatre objectifs sont définis.

- Proposer un outil générique qui isolera la modélisation du problème et sa résolution afin de maximiser la réutilisation des différents composants respectant ainsi le concept de « boite noire ».
- Garantir un niveau d'efficience suffisant pour permettant l'obtention de solutions de qualité dans des temps raisonnables.
- Considérer le problème dans son ensemble et la prendre en considération les différents aspects qui peuvent le composer.
- Assurer une interaction facile, intuitive et transparente avec les différents utilisateurs grâce à une interface graphique sans que ceux-ci n'aient besoin d'accéder directement au code source décrivant le système et les techniques d'optimisation proposées.

Comme cela est présenté sur la Figure 48, le Framework s'articule autour de trois entités : le cadre expérimentale, le problème étudié et la méthode d'optimisation.

3.4.3.2.1. Le cadre expérimental

A l'intérieur du cadre expérimental sont présents trois composants. Le générateur permet la génération et la transmission de variables environnementales vers les entrées du modèle représentant le problème étudié. Ces variables représentent les circonstances du problème sur lesquelles aucun contrôle n'est possible. Le second composant est le traducteur. Il permet la collecte et l'analyse de données simulées. Il joue le rôle d'intermédiaire, de filtre entre la technique d'optimisation et les

sorties du modèle. Un troisième composant facultatif est présent. Il s'agit de l'accepteur. Celui-ci permet la vérification pour les problèmes impliquant des contraintes.

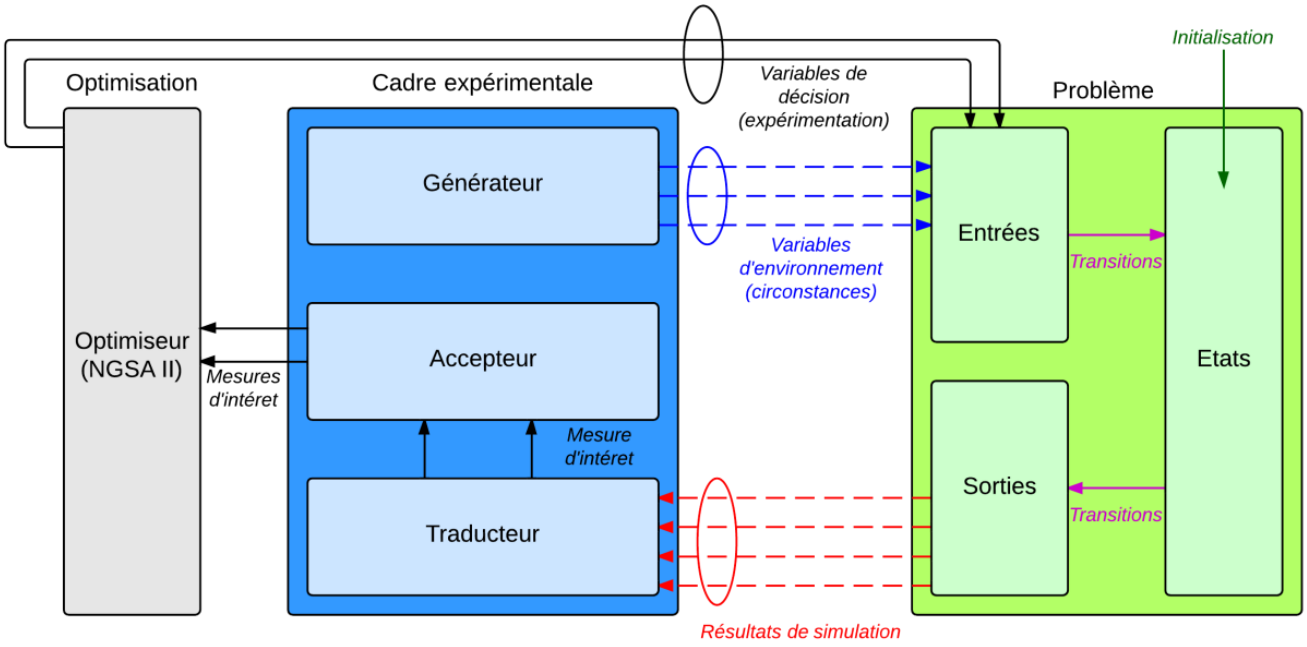


Figure 48 Les composants du framework SIMEON (R-H Halim)

3.4.3.2.2. La description du problème

La description du problème quant à elle se présente sous la forme d'un modèle DEVS (atomique ou couplé) sur lequel sont présentes différentes entrées qui pourront générer des transitions d'états et d'éventuelles sorties.

3.4.3.2.3. La méthode d'optimisation

Enfin, l'entité d'optimisation se base sur la métaheuristique multicritère « nondominated sorting genetic algorithm II » (NSGA-II). Différentes représentations des variables de décisions sont possibles : binaire, réelle, entier, type mixte et permettent ainsi une adaptation à l'ensemble des problèmes numériques.

Les différentes sorties seront collectées par le module d'optimisation. Elles seront analysées dans leur ensemble par une fonction d'évaluation. Une fois cette analyse terminée, un résultat sera transmis vers une méthode d'optimisation de type métaheuristique (e. g. algorithmes génétiques). Les méthodes d'optimisation une fois exécutées proposeront alors au module de contrôle différentes mises à jour des paramètres et de la structure en vue d'obtenir de meilleures évaluations lors des itérations suivantes.

Ces travaux centralisent l'ensemble des travaux précédents et offrent un espace de travail robuste d'optimisation via simulation DEVS. Malgré une maturité incontestable de l'approche, celle-ci se concentre sur l'utilisation du formalisme comme source d'évaluation. Les apports potentiels de la simulation pour la partie optimisation ne sont pas exploités et une seule métaheuristique est utilisée.

3.4.4. Limites actuelles

Ces différentes approches posent les bases du couplage de deux domaines que sont « la modélisation et la simulation de systèmes à évènements discrets » et « l'optimisation de problèmes complexes ». Les résultats obtenus sont encourageants et démontrent l'intérêt de l'intégration des deux domaines. Cependant des limites sont présentes concernant le choix de la méthode d'optimisation ainsi que la généricité de l'outil. Nous allons les développer.

3.4.4.1. Une optimisation statique et standard

Les différentes approches présentées précédemment utilisent un nombre très limité de techniques d'optimisation. Le choix de ces techniques d'optimisation n'est jamais justifié et ne permet pas la comparaison des performances par rapport à d'autres alternatives alors que nous savons que des temps d'optimisation très variables peuvent être obtenus pour un même problème avec différentes méthodes ou même seulement avec différents paramétrages d'une même méthode.

Dans ces approches, les techniques d'optimisation implémentées utilisent la simulation uniquement durant la phase d'évaluation des solutions potentielles. Bien que cette conception assure une certaine qualité de codage, elle isole les deux concepts et ne permet donc pas à l'optimisation de profiter d'un éventuel dynamisme offert par la simulation.

3.4.4.2. Une optimisation fastidieuse

Bien que ces approches couplent des techniques d'optimisation avec des simulations DEVS, ce couplage reste fastidieux pour des non-spécialistes et nécessite d'importants temps d'adaptation. En effet, nous avons observé deux cas de figure. Dans le premier cas, la méthode d'optimisation est adaptée aux spécificités du problème étudié ce qui ne permet pas sa réutilisation pour d'autres applications ultérieures. Dans le second cas, la méthode d'optimisation est totalement indépendante du modèle étudié. C'est alors celui-ci qui doit s'adapter aux spécificités de la méthode utilisée. Dans les deux cas, les phénomènes de dépendance entre les différents modules sont importants et nécessitent des temps d'adaptation considérables et de bonnes connaissances informatiques.

3.4.4.3. Une optimisation chronophage

La simulation de certains systèmes nécessite des temps de calcul considérables. Indépendamment de cet aspect, l'optimisation de certains problèmes complexes nécessite elle aussi des temps de calcul importants. Le regroupement de processus de simulation et d'optimisation peut alors être générateur de temps de calcul dissuasifs pour de nombreux décideurs. Prenons l'exemple d'un modèle ayant comme entrées cinq variables décisionnelles, pouvant chacune prendre une valeur parmi une ensemble de vingt nombres entiers. En considérant que la durée de simulation de chaque configuration est d'environ cinq secondes et que l'algorithme d'optimisation ne teste que 5% de l'ensemble des combinaisons possibles le temps total de l'optimisation sera supérieur à neuf jours.

3.5 Conclusion

Ce chapitre a permis de définir les concepts généraux de la modélisation et de la simulation. Parmi la multitude d'approches et de systèmes pouvant être modélisés nous nous sommes intéressés en particulier aux systèmes à évènements discrets qui sont de nos jours très largement utilisés. Pour cela, nous avons référencé les principales méthodes et approches associées (automates à états finis, réseaux de Pétri, systèmes multi-agents, DEVS) afin d'en choisir une.

En adéquation avec les finalités recherchées, à savoir l'optimisation par la simulation, nous avons choisi de baser nos travaux sur un formalisme de haut niveau, robuste et extensible : le formalisme DEVS. Nous avons alors effectué une analyse critique des rares travaux d'optimisation via simulation impliquant celui-ci. Ces différentes approches, qu'elles soient génériques ou spécifiques ont permis l'ouverture d'une voie de recherche plus que prometteuse.

Cependant de nombreuses pistes restent à explorer tant au niveau du dynamisme, de la généricité que des performances. Les problématiques qui en découlent ont servi de socle à notre architecture de modèle.

Celle-ci devra permettre : le dynamisme de l'optimisation, l'arrêt anticipé de certaines simulations, une adaptation « sur mesure » automatique au problème étudié, la réutilisation de l'ensemble des composants et ce de manière efficiente et intelligente. Les concepts développés ainsi que leur formalisation DEVS sont présentés dans le chapitre suivant.

Chapitre 4
Architecture proposée pour DEVS & Métaheuristiques

« La grande leçon de la vie c'est que parfois ce sont les fous qui ont raison »

Winston Churchill

L'étude bibliographique présentée dans le chapitre précédent montre qu'il existe peu d'approches génériques d'optimisation via simulation avec le formalisme DEVS. De plus, celles-ci sont difficilement adaptables à différents problèmes, nécessitent des temps d'exécutions importants et se basent sur des méthodes d'optimisation statiques qui ne profitent pas du dynamisme offert par la simulation.

L'approche que nous proposons dans ce chapitre vise à développer une nouvelle architecture d'optimisation par la simulation « orientée modèle », intégrant les métaheuristiques sous la forme de modèles DEVS simulés.

Dans un premier temps, nous introduirons de manière générale l'architecture de modèles que nous proposons pour la réalisation d'optimisations paramétriques. Celle-ci se concentre sur deux points : « l'optimisation intelligence » et « l'évaluation externalisée ».

« L'optimisation intelligente » se construit à partir de deux modèles : le « contrôleur » et « l'optimiseur ». « Le contrôleur » permet la génération d'algorithmes d'optimisation, de paramètres associés et de solutions aléatoires. « L'optimiseur » gère l'évaluation et la mise à jour des solutions proposées. Les échanges entre ces deux modèles permettent au comportement du modèle « optimiseur » de s'adapter de manière dynamique au problème étudié. Leur objectif est de permettre une convergence des solutions vers l'optimum, plus rapide et de meilleure qualité.

« L'évaluation externalisée » permet quant à elle, de réaliser l'évaluation d'une solution proposée sur un modèle DEVS indépendant, représentant le problème étudié. Pour cela, elle se base sur deux types de modèles : « les adaptateurs » et «les interpréteurs ». Les « adaptateurs » permettent la traduction de solutions représentées de manière standard vers les ports d'entrées typés du modèle représentant le problème étudié. Différents adaptateurs sont proposés : classiques, temporisés, spatialisés, temporisés-spatialisés. Les « interpréteurs » gèrent le flux d'information provenant des sorties de simulation du modèle représentant le problème étudié en direction du modèle d'optimisation. Ils peuvent être « unitaires » et interpréter une seule valeur de sortie ou être « multiples » et interpréter plusieurs valeurs de sorties à des temps distincts. Ces derniers permettent également de réduire la durée de certaines simulations jugées inutiles à la progression de l'optimisation et in fine, augmenter la rapidité d'obtention d'une ou plusieurs solutions optimales.

Une fois ces différents concepts de notre architecture présentés, nous détaillerons leurs transcriptions DEVS sous la forme de modèles atomiques. Pour cela nous représenterons le comportement de chacun des modèles sous la forme de diagrammes UML d'états-transitions et détaillerons leur comportement en respectant les spécifications du formalisme DEVS.

4.1 Présentation de l'architecture globale

L'architecture proposée se compose de deux groupes de modèles DEVS dédiés : « l'optimisation intelligente », « l'évaluation externalisée » du problème comme cela est schématisé sur la Figure 49.

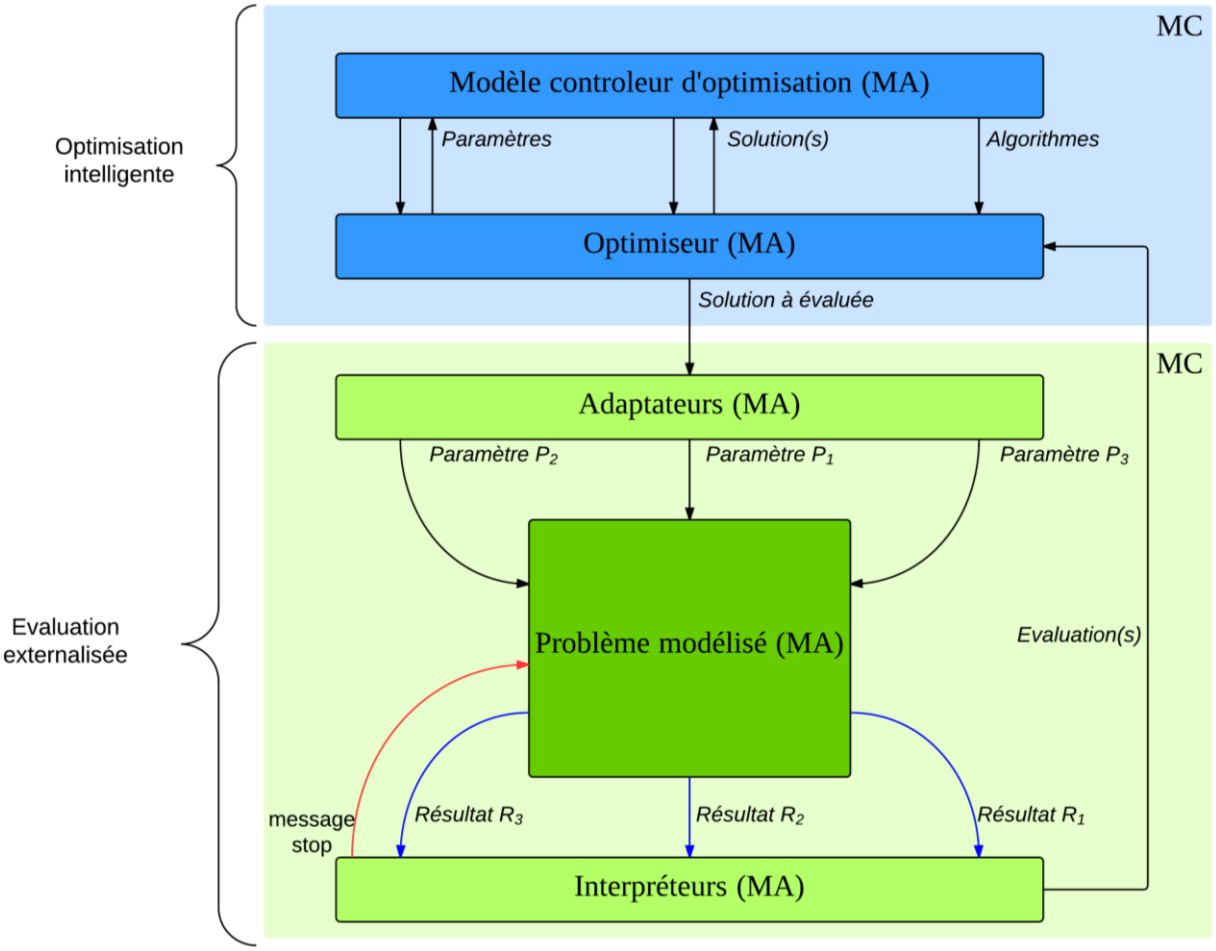


Figure 49 Architecture proposée

« L'optimisation intelligente » permet l'optimisation paramétrique du système étudié. Elle se compose de deux modèles : le « Contrôleur d'optimisation » et l'« Optimiseur ». Le contrôleur d'optimisation permet de piloter l'optimiseur lors la résolution simulée du problème. Pour cela, il peut réaliser trois types d'échanges contenant : des algorithmes d'optimisation ainsi que leur paramétrage et enfin des solutions générées de manière aléatoire pouvant être de différents types et structures. À partir de ces différentes informations, l'optimiseur va être instancié lors de l'initialisation de la simulation et démarrer sa première itération en externalisant une par une les différentes solutions qui le composent vers le modèle représentant le problème qui sera lui aussi simulé. D'après les résultats de l'ensemble des sorties produites par le modèle du problème, l'optimiseur attribuera à chacune des solutions potentielles une évaluation. Cette évaluation sera utilisée par l'algorithme d'optimisation afin de guider le processus d'intensification.

« L'évaluation externalisée » quant à elle permet le couplage rapide du modèle étudié dans le contexte d'optimisation. Les modèles qui la composent sont différents adaptateurs d'entrées et

différents interpréteurs de sorties. À partir d'une description standard (e. g. binaire) permettant la compatibilité avec les opérateurs des techniques d'optimisation, les adaptateurs vont générer différentes sorties représentant les paramètres d'entrées du problème étudié. Durant leurs simulations ceux-ci vont générer des résultats de simulation qui seront collectés par des interpréteurs en vue de leur attribuer une évaluation cohérente avec les attentes décisionnelles exprimées préalablement.

Ces deux groupes de modèles sont présentés en détail dans les deux parties suivantes.

4.1.1. Optimisation intelligente

Comme cela a été dit précédemment (cf. 2.2.4.) le choix d'une métaheuristique ainsi que son paramétrage sont des étapes indispensables pour assurer la qualité de l'optimisation. Cependant celui-ci est dans la grande majorité des applications réalisé de manière empirique. Cela génère d'importantes variations de performances sur différents problèmes mais également d'une instance à une autre d'un même problème.

Dans notre architecture, nous proposons donc l'intégration du processus d'optimisation sous la forme de modèles atomiques DEVS. Cette représentation permet de faire évoluer les méthodes d'optimisation à travers leurs paramétrages et leur comportement et permet ainsi à l'optimisation de s'adapter automatiquement au problème étudié ainsi qu'aux caractéristiques de décision exprimées par les décideurs.

Pour permettre le dynamisme du modèle atomique « Optimiseur », nous avons conçu un modèle atomique « Contrôleur » permettant de modifier les algorithmes d'optimisation. Ces modifications se basent sur des échanges de messages DEVS contenant les différents paramètres mais également une proposition d'algorithme d'optimisation décrivant le comportement du modèle d'optimisation. La génération de ces paramètres et de ces algorithmes d'optimisation sont présentés dans les parties suivantes.

4.1.1.1. Paramétrage automatique des algorithmes

4.1.1.1.1. Paramétrage pseudo-aléatoire

Le premier type de paramétrage automatique que nous proposons est un paramétrage basé sur la génération de nombres pseudo-aléatoires. Ces nombres pseudo-aléatoires sont générés d'après une fonction de distribution elle-même définie à partir d'une valeur ou d'un intervalle généralement proposés dans la description de la métaheuristique (e. g. taux de mutation des algorithmes génétiques compris entre 5% et 15%). Cette distribution peut suivre différentes lois :

- uniforme
- normale
- exponentielle
- personnalisée

4.1.1.1.2. Paramétrage déterministe

Le second type de paramétrage automatique offert par notre architecture DEVS se base sur une représentation des paramètres non pas par une valeur mais par une fonction dont le résultat dépend du nombre d'itérations réalisées. Cette fonction se présente sous la forme suivante :

$$f(iteration) \rightarrow paramètre$$

La nature de cette fonction pourra être de différents types d'après les connaissances disponibles du problème et de la sémantique du paramètre (e. g. un paramètre de sélection ou de perturbation n'évoluera pas de la même manière) :

- exponentielle
- logarithmique
- sinusoïdale
- polynomiale

4.1.1.1.3. Paramétrage cognitif

Ce troisième type de paramétrage se base sur une prise en compte indirecte des spécificités du problème. Pour cela durant l'exécution de l'optimisation, différentes méta-informations pourront être collectées par le contrôleur afin de permettre l'élaboration de statistiques. Comme nous pouvons le voir sur la Figure 50, ces informations pourront être la progression des évaluations au fil des itérations. Elles pourront également concerner l'ensemble des solutions (e. g. évaluation moyenne des solutions conservées en mémoire) ou une unique solution (e. g. meilleure évaluation, moins bonne évaluation, évaluation médiane). Le contrôleur pourra également analyser l'impact des différentes itérations (e. g. pourcentage d'itérations inutiles réalisées) sur le niveau de progression de l'optimisation. Le contrôleur pourra également tester différents paramètres sur des intervalles de temps rapprochés afin de mettre en place des stratégies d'apprentissage. De plus, si la progression observée est inférieure à un niveau souhaité, le processus d'optimisation sera interrompu le temps d'un échange de nouveaux paramètres entre l'optimiseur et le contrôleur.

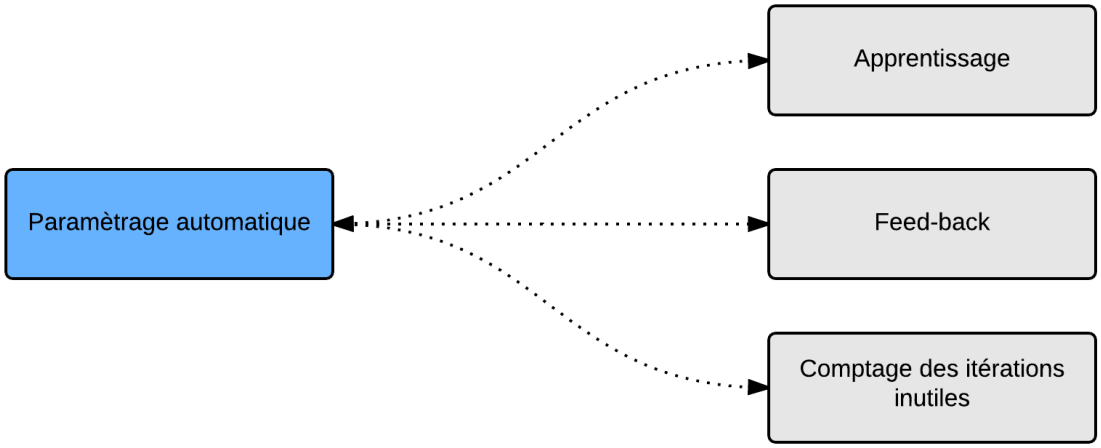


Figure 50 Sources d'informations pour le paramètrage automatique

La Figure 51 présente un exemple de choix de paramètres effectués par le contrôleur. En fonction des itérations précédentes, le contrôleur associe à chaque paramètre un indice de performance (feedback). Dans cet exemple, nous pouvons voir que deux paramètres doivent être choisis : « le taux de mutation » et la « méthode de croisement » (crossover). Concernant le taux de mutation, nous remarquons que les meilleurs résultats sont obtenus avec un taux de mutation de 15% et concernant la méthode de croisement, nous remarquons que le nombre d'itérations inutiles le plus faible est obtenu avec un croisement de type point unique. À ce moment précis du processus d'optimisation, le paramétrage qui semble être le plus pertinent est donc un taux de mutation de 15% et un croisement de type point unique.

Cette proposition de paramétrage n'est en aucun cas définitive, elle pourra fournir des résultats inférieurs à ceux espérés, dû notamment à des effets de combinaisons de paramètres inadéquates. (e.g un taux de mutation grand pour un croisement à points multiples) Dans ce cas-là, une génération de nouveaux paramètres sera rapidement déclenchée. Elle se matérialisera par un message émis par l'optimiseur en direction de son contrôleur. Le contrôleur répondra à cette requête par un message contenant un dictionnaire de paramètres appropriés pour l'algorithme présent dans l'optimiseur.

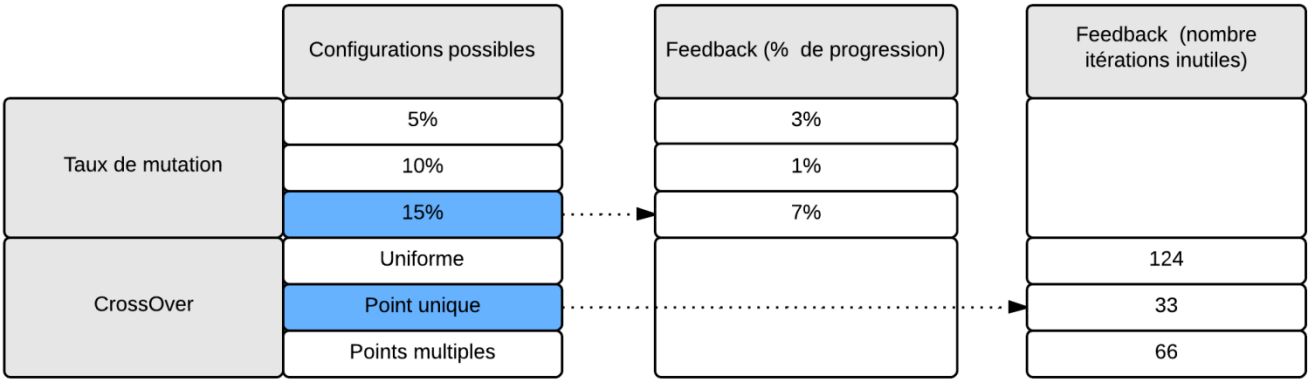


Figure 51 Exemple de choix automatique de paramètres

Si celle-ci s'avère insuffisante, l'algorithme d'optimisation pourra lui aussi être changé durant la simulation.

4.1.1.2. Choix et hybridation automatiques des algorithmes

Pour permettre le dynamisme de la méthode d'optimisation, nous avons isolé les différentes étapes communes à chaque métaheuristique : la génération aléatoire, l'évaluation, la sélection, les perturbations et l'arrêt du processus. Ces différentes étapes ont été classées en deux groupes : les étapes pouvant être délocalisées dans d'autres modèles que l'optimiseur et les étapes pouvant être segmentées en plusieurs sous-étapes dans l'optimiseur.

Dans notre architecture DEVS, la génération aléatoire des solutions initiales ainsi que le critère d'arrêt sont délocalisés. La génération aléatoire est gérée par le modèle « contrôleur » et est considérée comme un paramètre initial. L'évaluation quant à elle, est externalisée vers un modèle décisionnel représentant le problème étudié et traverse différents modèles présentés dans la partie suivante en vue d'assurer la compatibilité des échanges. Enfin, l'arrêt du processus d'optimisation est confié au modèle « interpréteur » qui pourra interrompre le processus d'optimisation dès que des sorties de simulation adéquates auront été collectées.

La segmentation concerne les étapes de perturbation, de mises à jour des différentes solutions potentielles. Chacune de ces étapes a été transcrite sous forme d'objets contenant tous une méthode nommée « action » prenant en paramètre d'entrée : l'ensemble des solutions potentielles, les résultats de leur évaluation externalisée et produisant comme sortie un ensemble de solutions potentielles modifiées, mises à jour qui devront être évaluées ou réévaluées. Précisions également que malgré d'importants points communs, des dissemblances terminologiques et conceptuelles existent entre les différents algorithmes. Cet aspect a été pris en compte dans notre architecture de modèles et a été traité par un typage et une structuration dynamique des variables sous la forme d'un dictionnaire de

valeurs associé à chaque solution. Cette segmentation alloue la possibilité de deux modes d'optimisation : l'optimisation dynamique et l'optimisation hybride.

4.1.1.2.1. Optimisation dynamique

Tout comme pour le choix des paramètres, les différents algorithmes mise en œuvre pour le processus d'optimisation pourront être choisis de manière aléatoire, déterministe ou cognitive. En effet le modèle « optimiseur » dispose d'une variable interne contenant les différentes « actions » qui seront exécutées séquentiellement formant ainsi l'étape de mise à jour des solutions de l'optimisation. Cette liste faisant partie de l'état interne du modèle, elle pourra être facilement mise à jour par l'émission d'une demande de nouvelles actions vers le « contrôleur » qui transmettra la réponse correspondante. Ainsi différentes méthodes d'optimisation pourront être utilisées durant la résolution du problème comme cela est décrit sur la Figure 52.

La méthode d'optimisation « Algorithmes génétiques » est utilisée dans la première phase d'optimisation du problème étudié. Elle est ensuite remplacée durant la phase intermédiaire par la méthode d'optimisation « Recherche harmonique » qui est elle-même remplacée lors de la phase finale du processus d'optimisation par la méthode d'optimisation dite « Algorithme de clonage ». Ce dynamisme peut également permettre l'utilisation de plusieurs méthodes de manière concurrente ouvrant alors la porte au mode d'optimisation hybride.

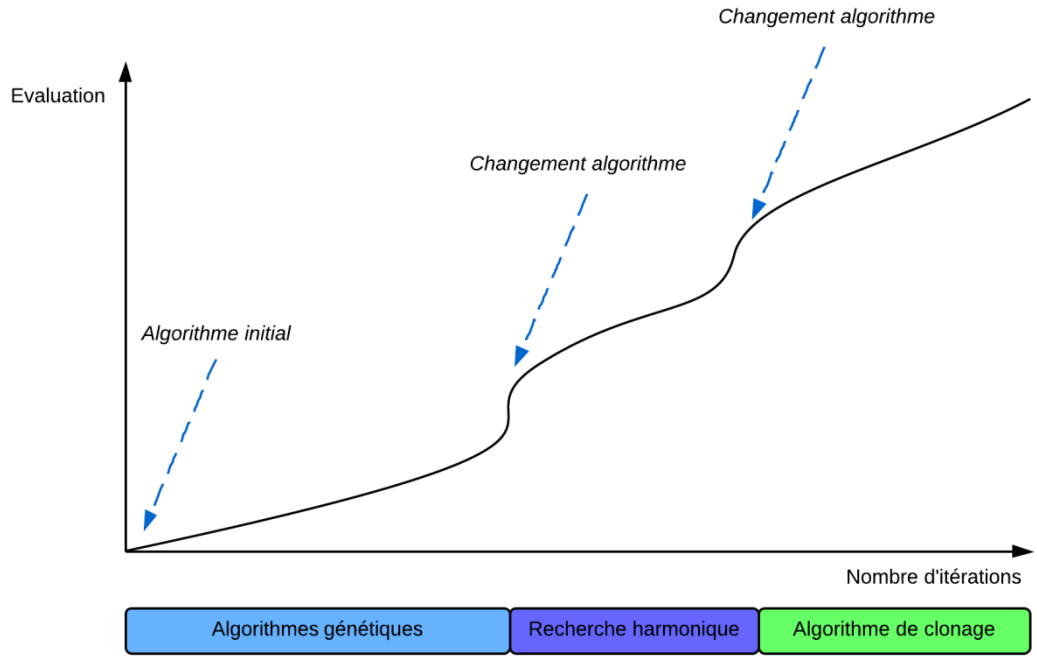


Figure 52 Exemple d'optimisation dynamique

4.1.1.2.2. Optimisation hybrides

Notre architecture se base sur une représentation unifiée de chacune des solutions potentielles. Ces solutions peuvent donc être mises à jour par n'importe quelle métaheuristique intégrée à l'outil. Comme nous pouvons le voir dans l'exemple sur la Figure 53, les modifications peuvent être effectuées en alternant un nombre illimité de techniques d'optimisation. Différents modes d'alternances peuvent également être définis : aléatoires, déterministes (e. g. la moitié des itérations utilisent les algorithmes génétiques pour mettre à jour les solutions potentielles et l'autre moitié des

itérations utilisent la recherche harmonique). Dans l'exemple de la Figure 53 nous remarquons aussi que chronologiquement ce sont les actions des algorithmes génétiques qui sont utilisées (cercles verts) lors de la première vague de mises à jour et que lors de la deuxième vague de mises à jour, ce sont les actions de la recherche harmonique (cercles bleus) qui sont utilisées. Il est également important de souligner l'aspect générique des méthodes de génération aléatoire et d'évaluation (cercles blancs) qui peuvent être utilisées avec l'ensemble des métaheuristiques proposées grâce une représentation binaire et standard des solutions.

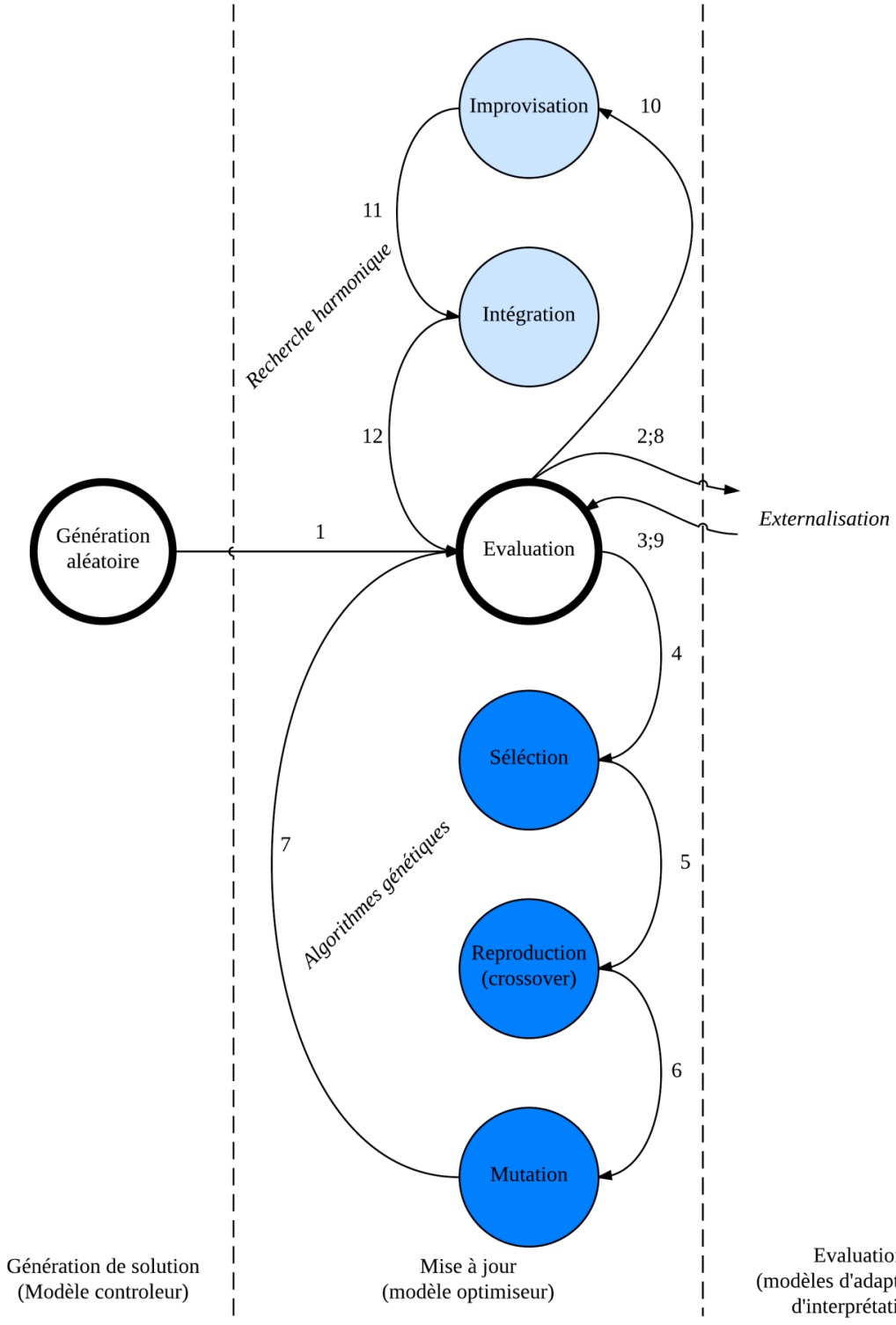


Figure 53 Exemple d'optimisation hybride

Les deux modes que nous avons présentés permettent la combinaison synergétique des avantages de différentes techniques d'optimisation tout en contournant leurs limites. Dans tous les cas, chaque solution doit être évaluée. Nous avons choisi d'externaliser cette évaluation comme cela est présenté dans la partie suivante.

4.1.2. Evaluation externalisée

L'évaluation externalisée permet de réaliser une optimisation paramétrique sur tout système modélisé par le formalisme DEVS. Pour cela, la compatibilité des modèles d'optimisation avec le modèle décrivant le système étudié est assurée par deux groupes de modèles : des « adaptateurs » et des « interpréteurs ».

4.1.2.1. Les adaptateurs

Les adaptateurs assurent la connexion du modèle d'optimisation vers les entrées du modèle étudié. Leur rôle est donc d'assurer la cohérence des données transmises vers le modèle représentant le problème étudié. Notre architecture DEVS propose quatre types d'adaptateurs : les adaptateurs classiques, les adaptateurs temporisés, les adaptateurs géo-localisés et les adaptateurs temporisés et géo-localisés.

4.1.2.1.1. Les adaptateurs classiques

Les adaptateurs classiques permettent de traduire un flux d'entrée binaire dans l'ensemble des types primitifs que sont : les entiers, les réels, les booléens, les caractères et les chaines de caractères. Ceux-ci se composent donc d'un port d'entrée et d'un port de sortie. La Figure 54 présente un exemple d'adaptateur classique de nombres entiers. Comme on peut le voir dans cet exemple, la traduction de la chaine binaire peut être non bornée (traduction d'une entrée en base binaire vers une sortie en base décimale). Elle peut également être bornée à partir d'une valeur minimale et d'une valeur maximale.

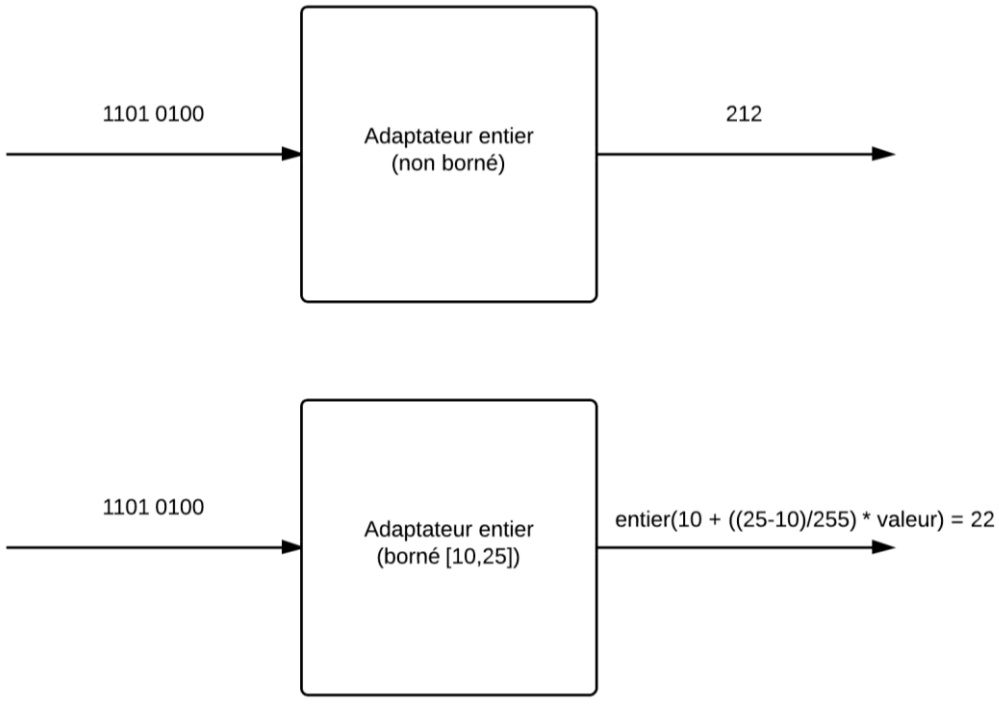


Figure 54 Exemple d'adaptateur classique

4.1.2.1.2. Les adaptateurs temporisés

Les adaptateurs temporisés permettent également la traduction d'un flux binaire vers l'ensemble des types primitifs. Cependant cette traduction ne se limite pas à la traduction d'une valeur. Elle permet d'associer à chaque valeur décrivant la solution potentielle un temps. Ensuite, nous obtenons alors une traduction de la forme « valeur-temps » pour la prise en charge de décision dans le temps lors des processus de simulation.

Comme nous pouvons aussi le remarquer sur la Figure 55, le flux binaire d'entrée est dans un premier temps séparé en deux parties : la première partie décrit la ou les valeurs à traduire alors que la deuxième partie décrit les temps qui doivent être associés à chacune des valeurs. Une fois la traduction du flux binaire terminée, une étape intermédiaire doit être réalisée. Elle consiste à réordonner les différents temps et leurs valeurs associées de manière chronologique. Enfin, le modèle calcule les différents temps auxquels le modèle devra s'activer et générer une sortie.

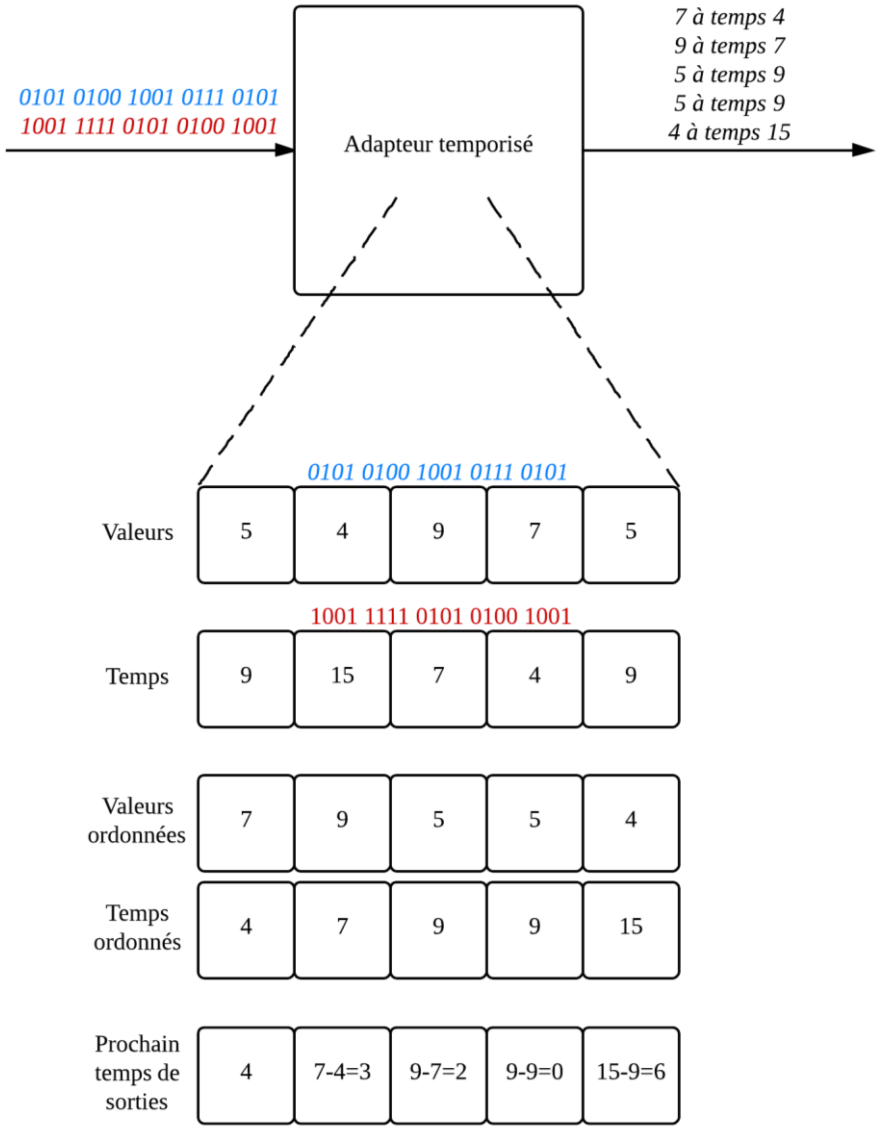


Figure 55 Exemple d'adaptateur temporisé

Certaines applications nécessitent également l'optimisation de variables spatialisées. Des adaptateurs ont donc été développés pour répondre à cette attente.

4.1.2.1.3. Les adaptateurs géo-localisés

Les adaptateurs géo-localisés permettent l'optimisation de variables dans l'espace. La représentation binaire de chaque solution potentielle contient une ou plusieurs valeurs auxquelles une position géographique est associée. Afin que nos travaux soient le plus génériques possibles, le système de coordonnées utilisé est « latitude-longitude ». La position géographique peut être interprétée de deux manières par les adaptateurs géo-localisés : la « spatialisation par ports» et la « spatialisation par coordonnées».

La « spatialisation par ports » affecte chaque variable traduite à un port de sortie spécifique connecté à un port d'entrée représentant une zone géographique bien définie du modèle du système étudié. En d'autres termes, ces modèles peuvent être considérés comme des routeurs spatiaux de messages DEVS. En effet, comme nous pouvons le voir sur l'exemple de la Figure 56, le flux binaire est séparé en plusieurs parties. Le nombre de parties est égal au nombre de ports de sorties présents sur le modèle qui lui-même est égal au nombre de ports d'entrées associés à des zones sur le modèle représentant le problème étudié.

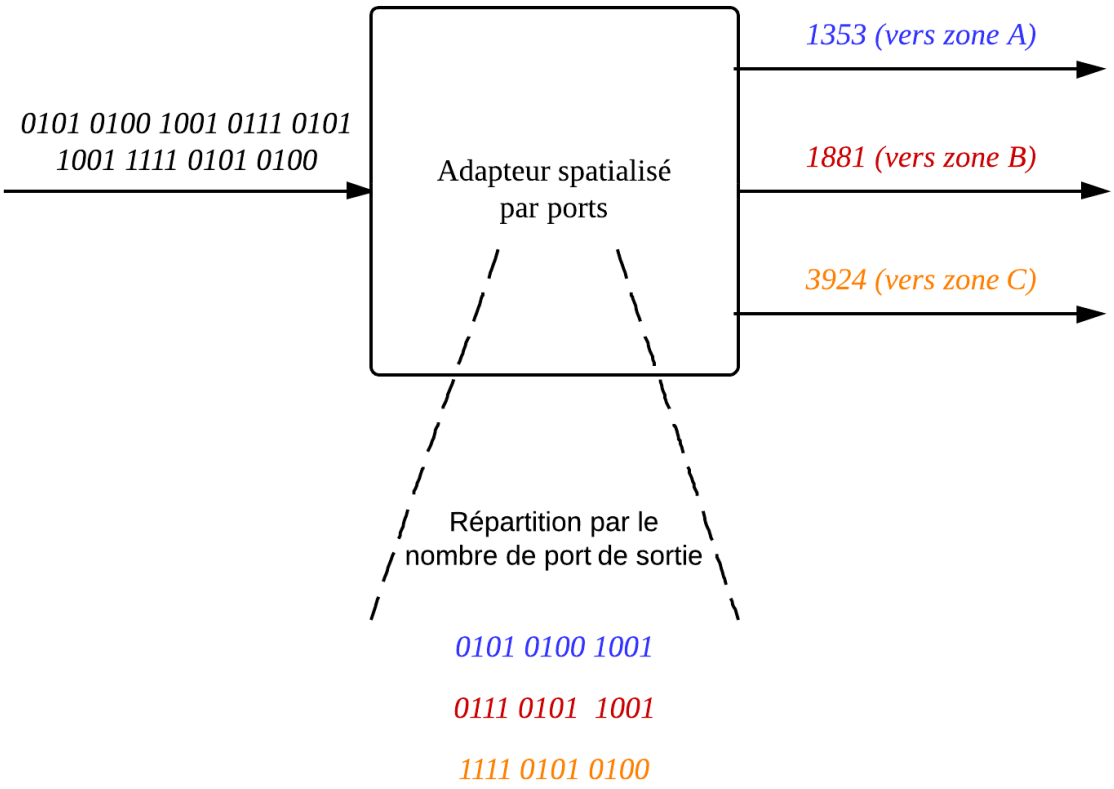


Figure 56 Exemple d'adaptateurs géo-localisés (spatialisation par port)

La « spatialisation par coordonnées » quant à elle fonctionne différemment. Elle affecte une coordonnée à chaque valeur indépendamment de son futur port d'émission. La coordonnée est donc transmise dans le contenu du message ce qui n'est pas le cas avec les adaptateurs de « spatialisation par port ». Un exemple de ce type d'adaptateur est donné dans la Figure 57. Comme nous le voyons, à chaque valeur traduite est associée une coordonnée géographique. Une liste contenant ces valeurs et

leurs coordonnées sont transmises sur l'unique port de sortie vers le modèle représentant le problème et impliquant un aspect spatial.

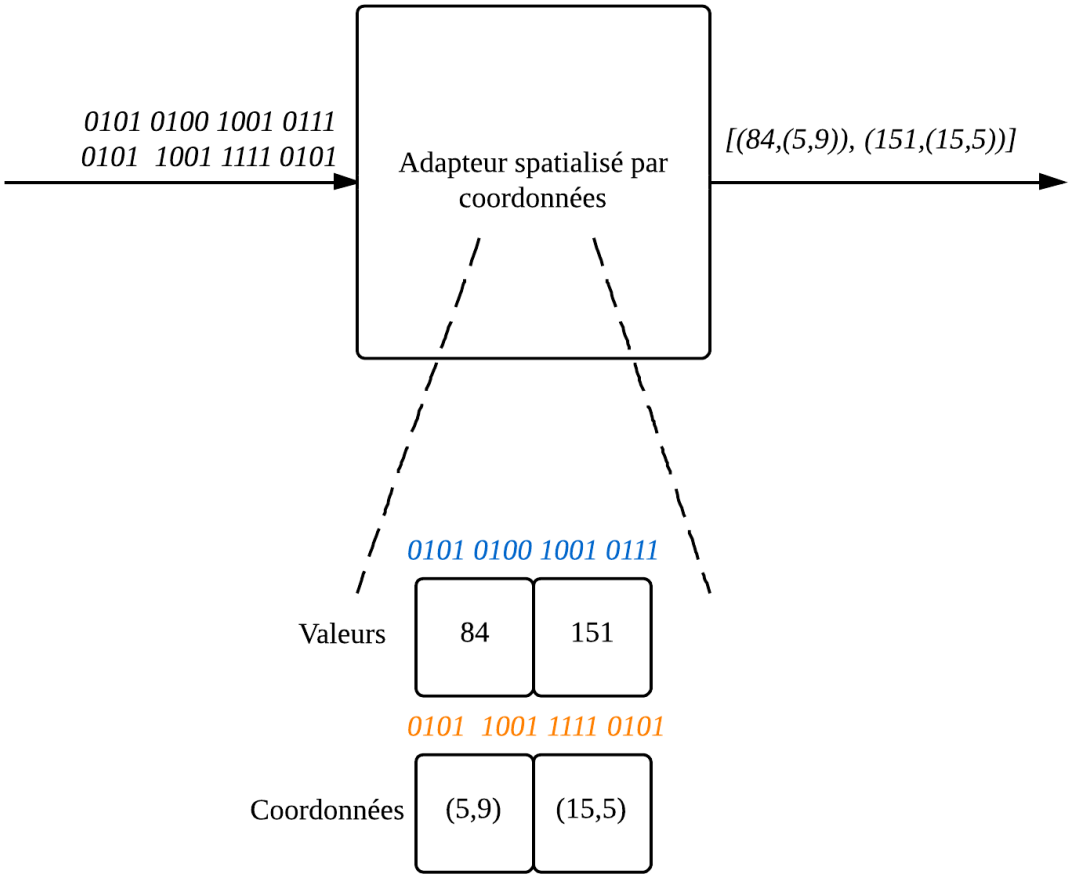


Figure 57 Exemple d'adaptateurs géo-localisés (spatialisation par coordonnées)

4.1.2.1.4. Les adaptateurs temporisés et géo-localisés

Le concept de temporisation ainsi que le concept de géolocalisation peuvent tous deux être utilisés pour une seule et même adaptation de solution. Deux adaptateurs sont alors disponibles : « les adaptateurs temporisés et spatialisés par port » et les « adaptateurs temporisés et spatialisés par coordonnées ». Pour chacun de ces modèles, un exemple est présenté sur la Figure 58 et la Figure 59.

Comme nous pouvons le voir sur la Figure 58, le flux binaire entrant est séparé en trois parties de tailles variables (choisies par l'utilisateur en fonction du niveau de précision requis). En effet, dans cet exemple seuls deux bits sont nécessaires pour décrire le numéro du port de sortie. Comme nous pouvons également l'observer, chacune de ces trois parties est traduite en base décimale. Nous obtenons alors trois listes décrivant les valeurs, leur temps, leur zone associée par un numéro de port. Ainsi la solution proposée dans cet exemple consiste à : transmettre la valeur 7 au temps 4 sur le port 3 ; transmettre la valeur 9 au temps 7 sur le port 1 ; transmettre la valeur 5 au temps 9 sur le port 2 ; transmettre la valeur 5 au temps 9 sur le port 0 ; transmettre la valeur 4 au temps 15 sur le port 2.

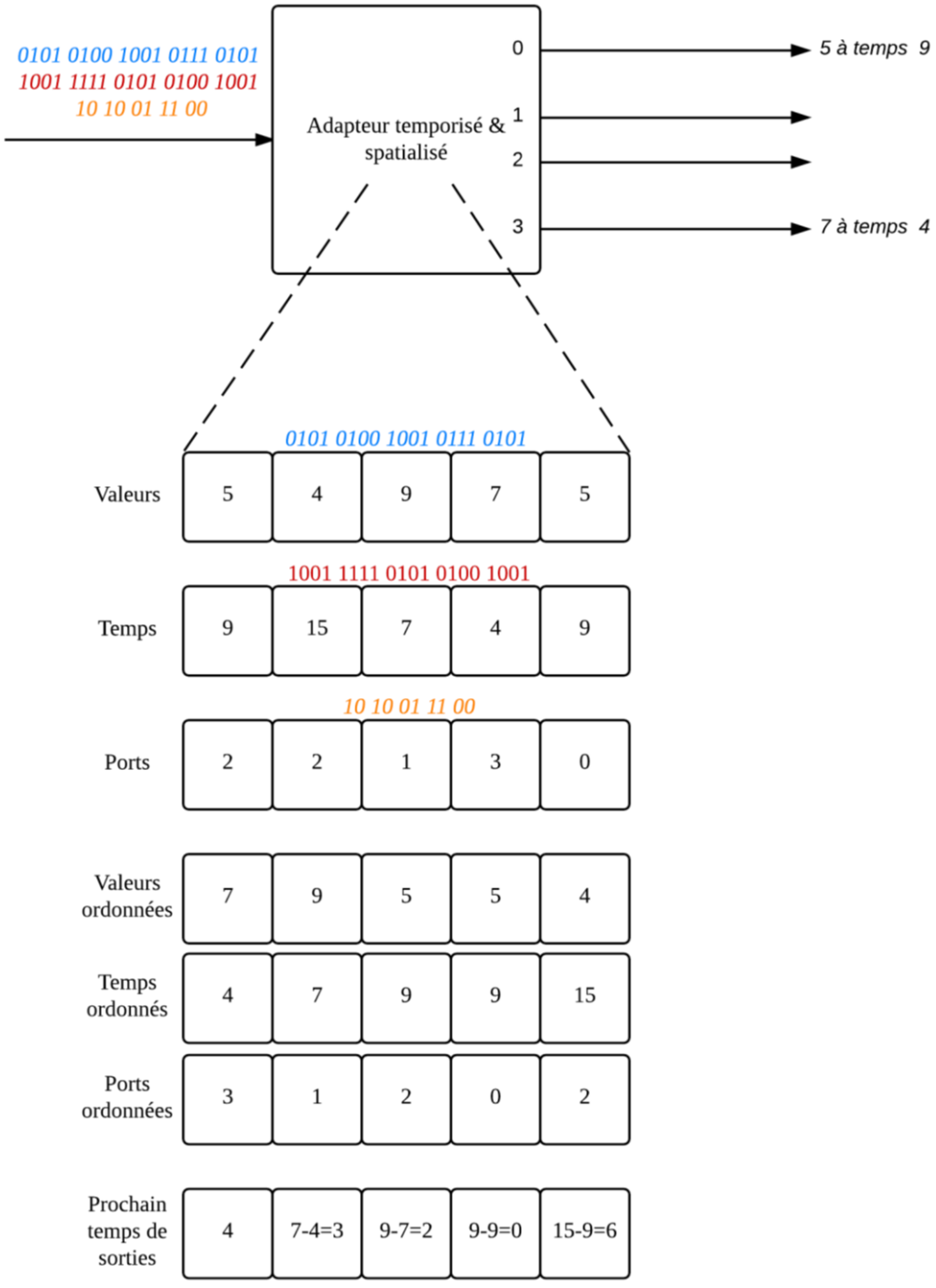


Figure 58 Exemple d'adaptateur temporisé et spatialisé (par port)

L'exemple de la Figure 59 est quelque peu différent. Le flux binaire est également séparé en trois parties. Cependant la description de l'aspect spatial n'est pas matérialisée sur les différents ports mais par différentes coordonnées géographiques. Pour cela, quatre valeurs doivent obligatoirement être définies pour permettre à l'adaptateur de délimiter la zone géographique du problème. Il s'agit de la « latitude minimale », la « latitude maximale », la « longitude minimale » et la « longitude maximale ». Les sorties correspondant à la solution potentielle adaptée seront les suivantes : transmettre la valeur 7 de coordonnée (23.33, 40) au temps 4 ; transmettre la valeur 9 de coordonnée (23.33, 30) au temps 7 ; transmettre la valeur 5 de coordonnée (23.33, 60) au temps 9 ; transmettre la valeur 5 de coordonnée (36.66, 40) au temps 9 ; transmettre la valeur 4 de coordonnée (36.66, 30) au temps 15.

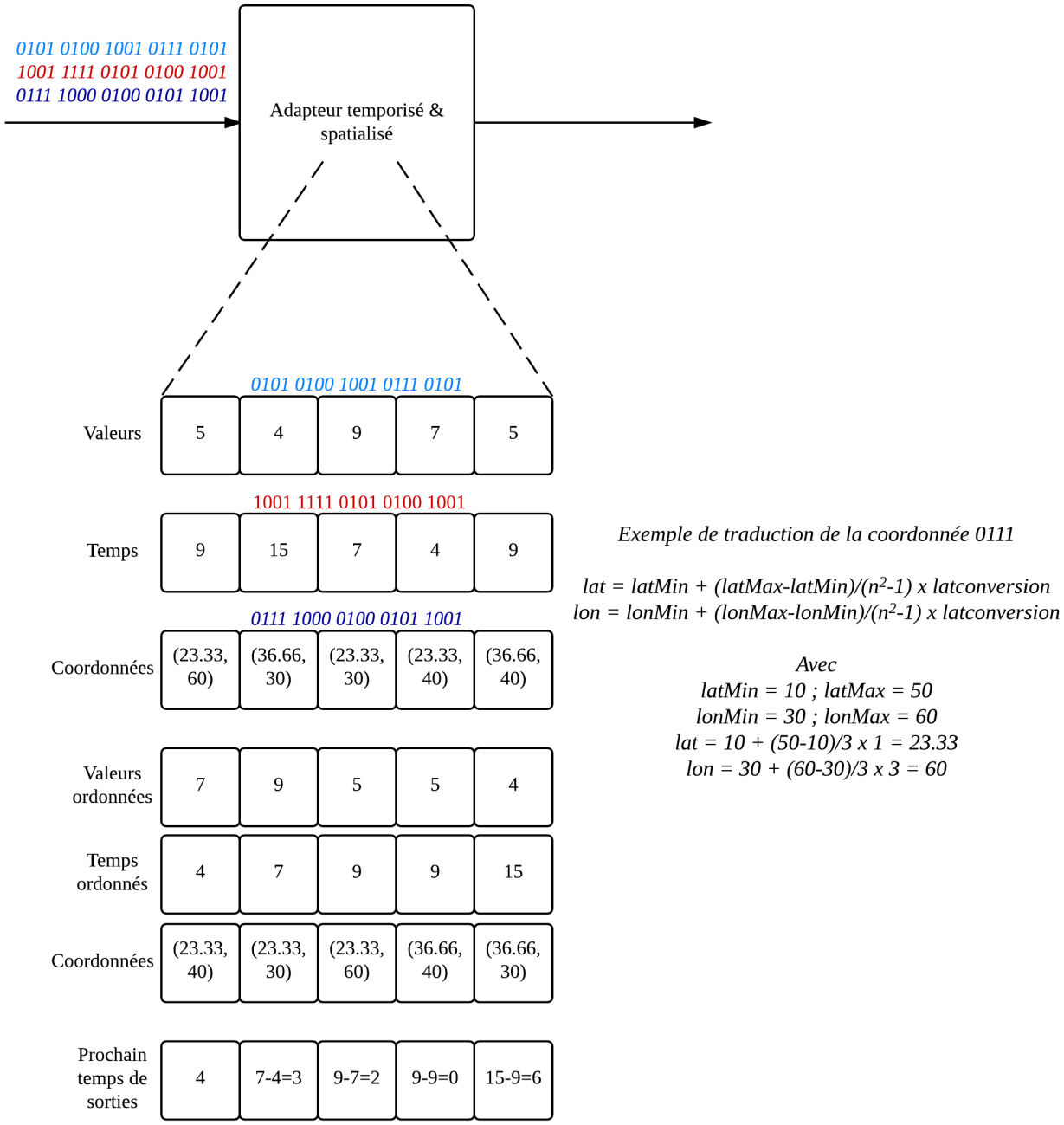


Figure 59 Exemple d'adaptateur temporisé et spatialisé (par coordonnées)

4.1.2.1.5. Le « segmenteur » de code

Notre architecture permet l'optimisation de solutions pouvant être décrites par un ensemble de variables de types pouvant être hétérogènes. Par conséquent plusieurs adaptateurs peuvent être nécessaires afin d'interfacer les modèles d'optimisation avec le ou les modèles décrivant le problème étudié.

Pour prendre en compte ce type de configuration, un modèle « segmenteur » a été conçu. Celui-ci vient s'intercaler entre le modèle « optimiseur » qui transmet une description de la solution

sous forme binaire et les différents « adaptateurs » qui eux sont en attente de ce flux binaire. Il permet le découpage du flux en plusieurs parties pouvant être de tailles identiques ou hétérogènes en fonction des spécificités du problème. Son fonctionnement est illustré dans l'exemple de la Figure 60.

Dans cet exemple un flux binaire provenant de l'optimiseur. Le « segmenteur » étant connecté à trois adaptateurs distincts, le flux est divisé en trois parties. Comme nous pouvons le voir la première partie du code est une traduction décimale de la portion binaire reçue. La seconde partie est une traduction vers un nombre réel et enfin la troisième partie quant à elle est une traduction vers un caractère codé en ASCI.

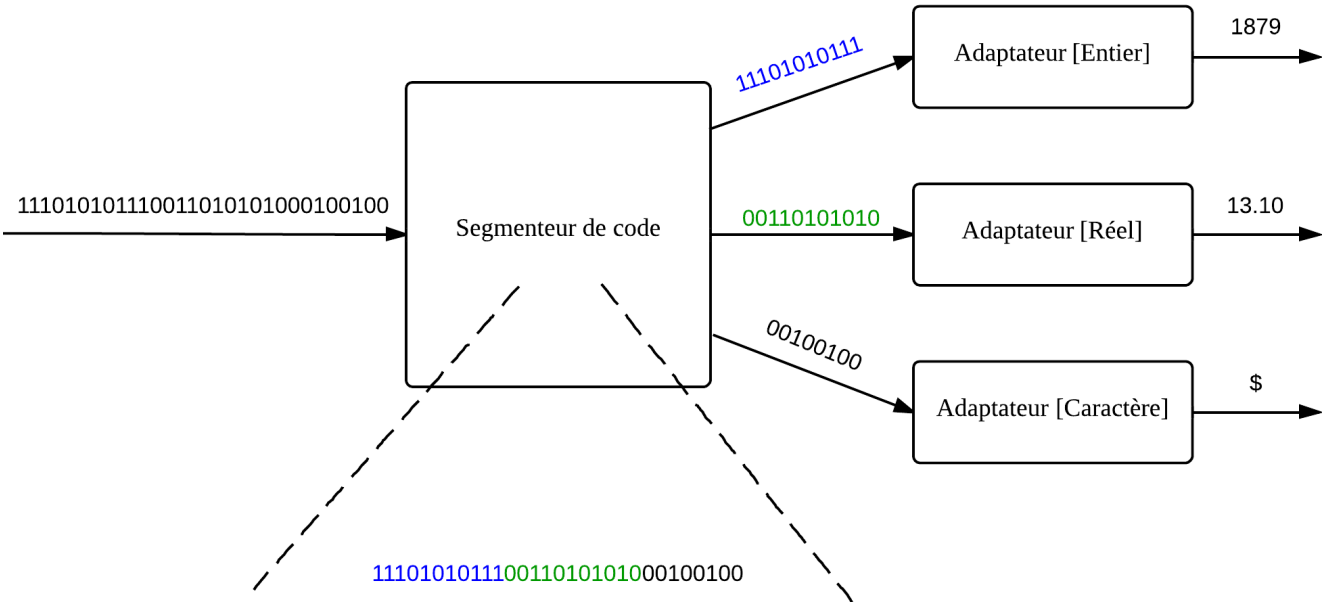


Figure 60 Exemple de segmentation de représentation

Toutes ces valeurs sont alors transmisses au modèle représentant le problème étudié et permettent ainsi la génération de sorties simulées qui seront analysées par les modèles « interpréteurs ».

4.1.2.2. Les interpréteurs

Les interpréteurs permettent de moduler les sorties du modèle décrivant le problème étudié et de les retransmettre vers modèle « optimiseur » dans lequel elles orienteront le processus d'intensification. Cette modulation se base sur les attentes exprimées par les décideurs.

Notre architecture propose deux groupes d'interpréteurs : les « interpréteurs unitaires » et les « interpréteurs multiples ». Elle propose également un modèle de centralisation de différentes interprétations permettant à l'analyse de devenir multicritère.

4.1.2.2.1. Les interpréteurs unitaires

Les interpréteurs unitaires collectent un unique message provenant du modèle étudié. À partir de ce message une évaluation est instantanément effectuée. Elle se base sur une fonction « valeur-satisfaction » qui associe à chaque sortie une valeur comprise entre 0 et 1 représentant la qualité de la solution. L'interprétation de ces valeurs dépend des finalités du processus d'optimisation qui peuvent consister à trouver les minimums ou les maximums globaux d'une fonction. Différentes représentations de la fonction « valeur-satisfaction » sont présentes dans notre architecture et peuvent

être facilement étendues. Parmi elles, nous pouvons citer les représentations suivantes : courbe cloche, courbe pyramidale et courbe personnalisée.

Un exemple d'interpréteur unitaire est donné sur la Figure 61. Dans cet exemple, nous pouvons voir une courbe « valeur-satisfaction » utilisée dans le cadre d'une optimisation de maximisation. Comme nous pouvons le constater la valeur souhaitée par l'utilisateur est 30. Si le modèle interpréteur reçoit comme entrée 30, l'évaluation qui sera transmise à l'optimiseur sera égale à 1 (la meilleure évaluation possible). Cependant si le modèle reçoit comme entrée 42, l'évaluation correspondante sera 0.575. Enfin pour toutes les valeurs inférieures à 0 ou supérieures à 60, une évaluation égale à 0 sera transmise (la plus mauvaise évaluation possible).

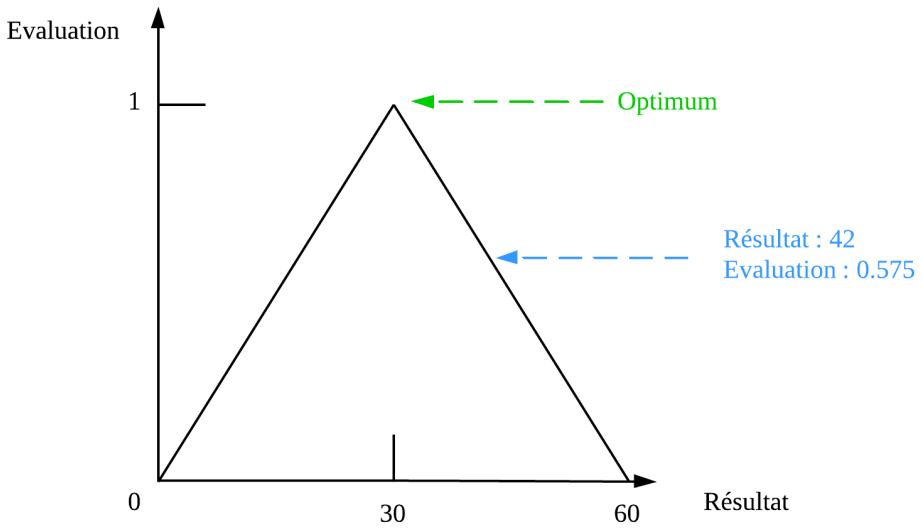


Figure 61 Exemple d'interpréteur unitaire

Les interpréteurs unitaires sont utiles pour la phase d'analyse de modèle produisant des sorties uniquement à la fin de leur simulation ou n'étant pas simulés dans le temps. Pour les modèles produisant différentes sorties, des « interpréteurs multiples » doivent être utilisés.

4.1.2.2.2. Les interpréteurs multiples

Les interpréteurs multiples ont également comme objectif de moduler les sorties du modèle représentant le problème en fonction des attentes exprimées par les décideurs. Cependant, contrairement aux interpréteurs unitaires qui collectent une seule valeur, ceux-ci raisonnent sur un ensemble de valeurs collectées et stockées durant la simulation du modèle représentant le problème.

La taille de la mémoire de stockage de ces valeurs est limitée. Elle peut être ajustée et ainsi permet aux « interpréteurs multiples » d'effectuer des analyses à différentes échelles de temps pouvant ainsi produire des propositions d'optimisation très différentes. En effet, une solution pourra présenter un optimum à court terme mais ne pas l'être à moyen et à long terme.

Deux types d'analyse peuvent être choisies par le décideur : « l'analyse différentielle » et « l'analyse par comptage ». La première consiste à mesurer la somme des distances entre une courbe de valeurs souhaitées et la courbe de valeurs obtenues lors de la collecte.

Un exemple « d'analyse différentielle » est présenté dans la Figure 62. La courbe bleue définit les différentes valeurs que le décideur souhaite obtenir par le modèle du problème durant sa

simulation. La courbe rouge pointillée quant à elle définit les différentes sorties que « l'analyseur multiple » a collecté durant la simulation du modèle. Le niveau de satisfaction sera inversement proportionnel à la somme des écarts entre ces deux courbes.

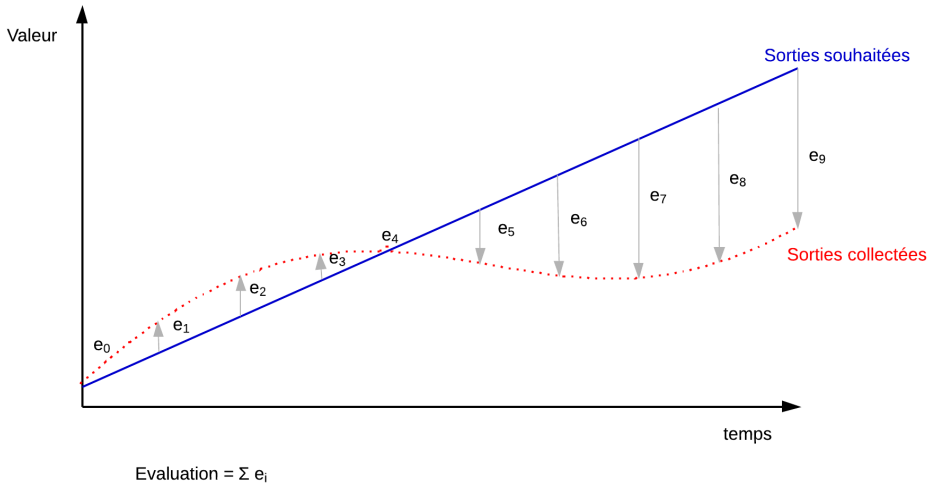


Figure 62 Exemple d'analyse différentielle

Un exemple « d'analyse par comptage » est présenté dans la Figure 63. Comme nous pouvons le voir dans cet exemple, différents intervalles sont définis permettant la classification des différents résultats obtenus. Cette classification permet le comptage des différentes valeurs à partir desquelles un niveau de satisfaction pourra être produit. Dans cet exemple, le niveau de satisfaction pourra être donné par la somme des points associés à chacune des valeurs en fonction de leur classification.

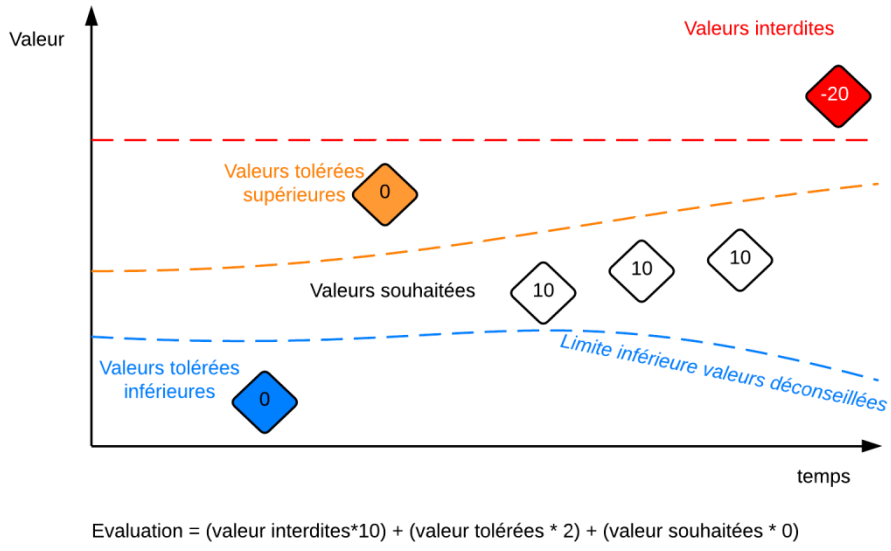


Figure 63 Exemple d'analyse par comptage

Ces deux analyses peuvent être déclenchées de différentes manières. Elles peuvent être déclenchées lorsque que le nombre de valeurs collectées est égal au nombre de valeurs pouvant être stockées en mémoire. Un message « stop » est alors transmis de l'analyseur vers le modèle étudié permettant de stopper sa simulation et de le réinitialiser à son état initial. Cependant, l'inconvénient de cette méthode est que chaque solution sera évaluée à partir d'un nombre constant de résultats de simulation. Nous pensons que dans certains cas ce nombre peut être réduit grâce à une analyse

économe permettant d'interrompre certaines simulations et ainsi de réaliser d'importants gains de temps.

4.1.2.2.3. Simulation économe

Les modèles « interpréteurs » peuvent donc limiter la durée de certaines simulations en les interrompant avant d'avoir collecté le nombre de valeurs maximales. Ces interruptions pourront se produire lorsque les résultats analysés assurent de ne pas obtenir une évaluation de meilleure qualité que la plus mauvaise évaluation présente parmi les solutions potentielles. Cette interruption est transparente pour le modèle représentant le problème étudié. Tout comme pour une interruption classique, elle se matérialise par l'émission d'un message « stop » du modèle « interpréteur » vers le modèle représentant le problème étudié. L'évaluation transmise vers le modèle « optimiseur » est quant à elle égale à l'infini dans le cas d'une minimisation et à moins l'infini dans le cas d'une maximisation.

La Figure 64 présente un exemple de collecte de résultats. Sur celle-ci nous pouvons observer quatre courbes distinctes. La courbe « Résultats souhaités » présente l'évolution idéale des résultats que le décideur a définie préalablement à l'optimisation. La courbe « Résultats mauvais » présente l'évolution des résultats de la plus mauvaise solution potentielle conservée en mémoire. Les deux courbes : « Solution simulée A » et « Solution simulée B » présentent quant à elles un exemple de résultats pouvant être collectés par les modèles interpréteurs.

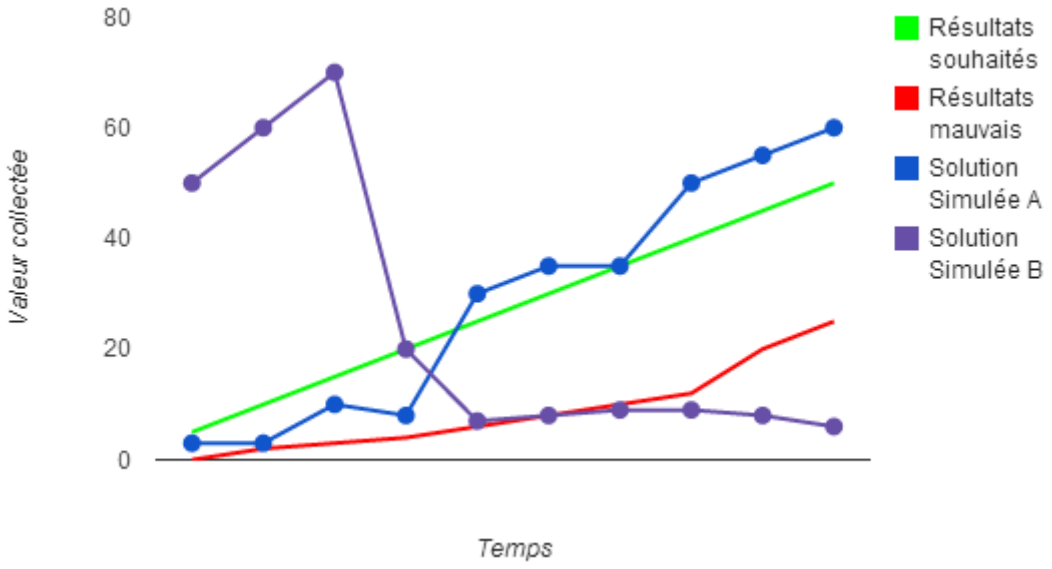


Figure 64 Exemple de résultats : souhaités, collectés, mauvais

Durant la collecte des informations de ces deux courbes, le modèle « interpréteur » va cumuler les différents écarts par rapport aux résultats souhaités, comme cela est présenté dans la Figure 65. Deux raisonnements sont alors possibles : un raisonnement déterministe et un raisonnement probabiliste.

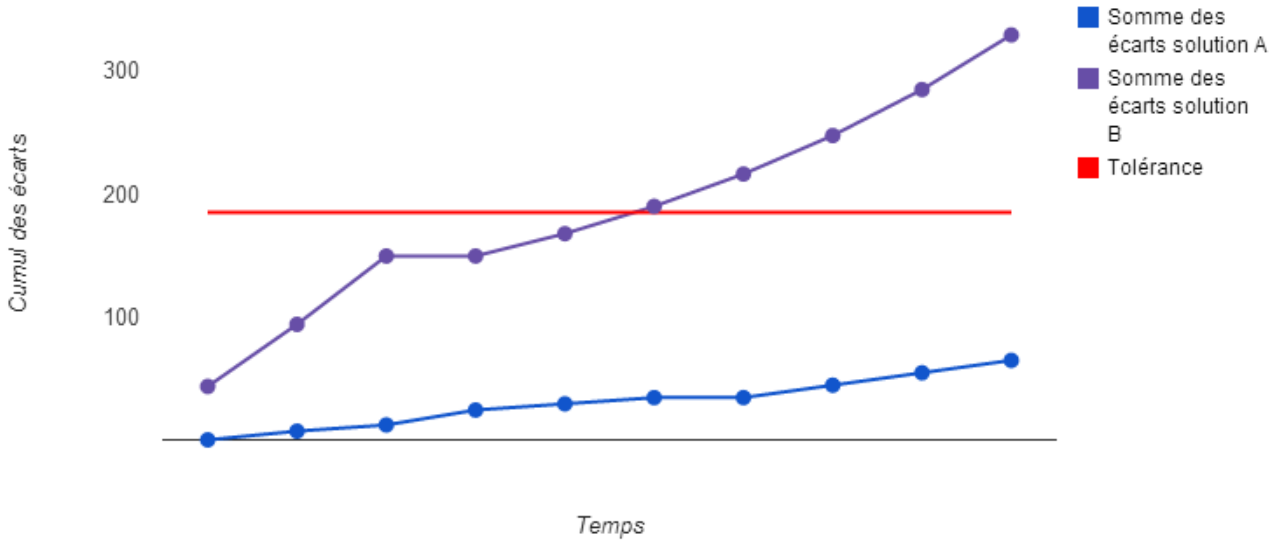


Figure 65 Cumul des écarts

Le raisonnement déterministe interrompra la simulation en cours dans l'unique cas où la somme des écarts mesurés dépasse le « seuil de tolérance ». Le déclenchement de ces interruptions peut être caractérisé par la fonction suivante :

$$\begin{cases} Vrai\ Si \sum_{i=0}^{n} ec_i > \sum_{j=0}^{m} em_i \\ Faux\ Sinon \end{cases}$$

Dans cette fonction, « ec » désigne le vecteur des écarts rencontrés pour la solution en cours d'évaluation et « n » est le nombre de valeurs collectées, « em » représente le vecteur des écarts rencontrés pour la plus mauvaise solution potentielle présente en mémoire et « m » désigne le nombre de valeurs maximales pouvant être collectées par le modèle d'interprétation.

Le raisonnement probabiliste, quant à lui, pourra interrompre des simulations avant que la somme des écarts mesurés pour la solution actuelle soit supérieure à la somme des écarts de la plus mauvaise solution. Pour cela, il se basera sur une probabilité dont l'importance sera proportionnelle à la somme des écarts mesurés. En d'autres termes, plus la somme des écarts entre les valeurs espérées et les valeurs collectées est grande plus la probabilité d'interruption sera forte. Le déclenchement d'une interruption probabiliste se basera donc sur une fonction du type suivant :

$$\begin{cases} Vrai\ si\ \xi < \left(\frac{\sum_{i=0}^{n} ec_i}{\sum_{j=0}^{m} em_i} \right) / m \\ Faux\ sinon \end{cases}$$

Dans cette fonction « ξ » désigne un nombre réel aléatoire compris entre 0 et 1, « ec » désigne le vecteur des écarts rencontrés pour la solution en cours d'évaluation et « n » représente le nombre de valeurs collectées, « em » désigne le vecteur des écarts rencontrés pour la plus mauvaise solution potentielle présente en mémoire et « m » renferme le nombre de valeurs maximales pouvant être collectées par le modèle d'interprétation.

Ces deux raisonnements présentent chacun des avantages et des inconvénients. L'utilisation de raisonnements probabilistes augmente le risque d'interruption de simulations utiles à la progression de

l'optimisation mais permet, en contrepartie, l'interruption plus rapide de simulations présentant des résultats de mauvaise qualité. Le raisonnement déterministe quant à lui impose d'atteindre « le seuil de tolérance », ce qui augmente la durée moyenne des simulations interrompues mais assure de ne pas stopper des simulations utiles au processus d'optimisation.

Les résultats d'un modèle représentant le problème pouvant être de types et de sémantiques divers, notre architecture permet l'utilisation de plusieurs interpréteurs dans le cadre d'une même optimisation.

4.1.2.2.4. Le « regroupeur» de satisfactions

Les regroupeurs centralisent les différentes interprétations des résultats de simulation du problème étudié. Ils se présentent sous la forme d'un modèle ayant un nombre de ports d'entrées variable et un unique port de sortie en direction de l'optimiseur.

Différents types de regroupeurs peuvent être intégrés à notre architecture. Ces modèles se basent sur une analyse statistique des différents niveaux de satisfaction obtenus. Ils permettent aux décideurs de modéliser facilement le niveau de considération des différentes analyses réalisées par les interpréteurs.

Un exemple de « regroupeur » est présenté dans la Figure 66 ; les trois modèles interpréteurs qui chacun ont collecté et analysé différentes valeurs de sortie produites par le modèle représentant le problème étudié. Les résultats obtenus par ces interpréteurs : 0.88, 0.12 et 0.47 sont transmis vers le regroupeur de satisfactions. En fonction d'une pondération des ports d'entrées définie préalablement par le décideur, une moyenne pondérée est effectuée.

Ce résultat est alors transmis vers l'optimiseur où il sera affecté à la solution qui avait été transmise vers le ou les adaptateurs. Si toutes les solutions de l'optimiseur sont évaluées, la phase de perturbation des solutions sera alors lancée. Dans le cas contraire, la prochaine solution sera transmise par l'optimiseur en utilisant les différents composants d'externalisation de l'évaluation.

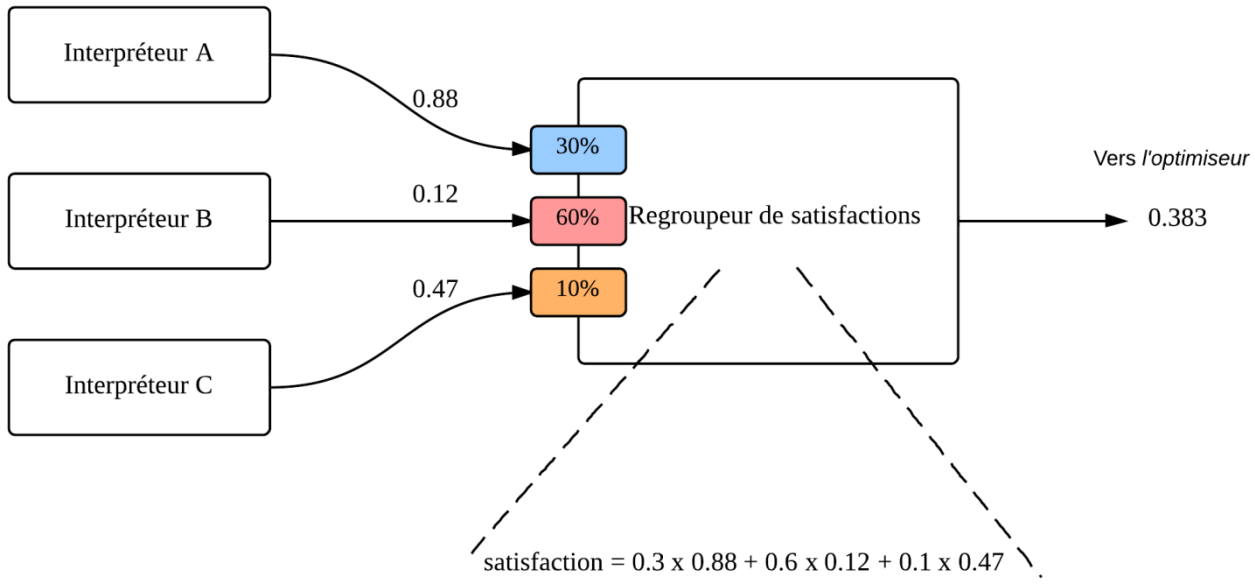


Figure 66 Exemple de « regroupeur » de type moyenne pondérée

4.2 Modélisation DEVS

4.2.1. Modélisation de « l'optimisation intelligente »

4.2.1.1. Le modèle atomique DEVS « Controler » (Contrôleur)

Le modèle atomique « Controler », dont la représentation graphique est visible dans la Figure 67, assure le dynamisme de l'optimisation et son adaptation automatique au problème étudié. Pour cela, il dispose de trois ports d'entrées, de trois ports de sorties et fonctionne sur le modèle « client-serveur » en produisant une ou plusieurs réponses en fonction d'une ou plusieurs requêtes.

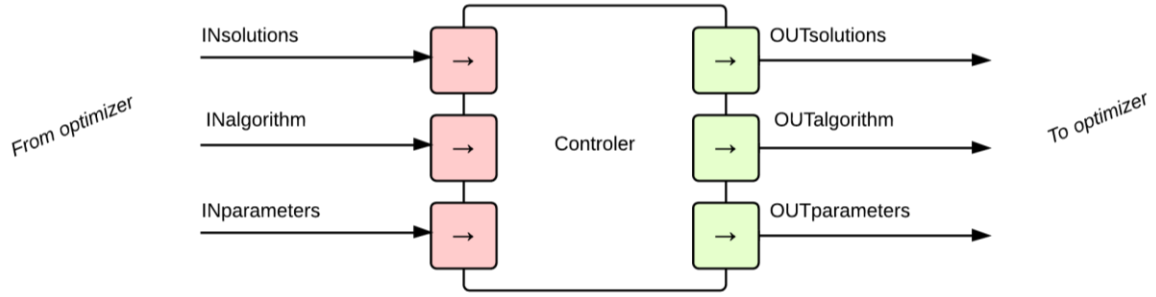


Figure 67 Modèle atomique « controler » (contrôleur)

Deux modes de fonctionnement du modèle sont possibles. Le premier est utilisé lors de processus d'initialisation de l'optimisation. Comme cela est présenté sur la Figure 68 et le Tableau 18, lors du démarrage de la simulation (temps de simulation = 0) le modèle va dans un premier temps générer un ensemble de solutions aléatoires (GENERATE_SOLUTIONS). Le nombre ainsi que le type de représentation de solutions est défini dans un fichier de configuration lu par le modèle. Une fois cette étape de génération aléatoire terminée, le modèle va ensuite produire le comportement (GENERATE_ALGORITHM) de l'algorithme d'optimisation en sélectionnant des composants présents dans une libraire contenant les différentes métaheuristiques (e. g. algorithmes génétiques, recherche harmonique). Ce choix pourra être imposé par l'utilisateur lors de la configuration du modèle grâce à l'interface graphique ou être aléatoire. Une fois l'algorithme généré, le modèle « Controler » va alors proposer un ensemble de paramètres nécessaires à l'algorithme (GENERATE_PARAMETERS) produit précédemment (e. g. taux de mutation pour les algorithmes génétiques). Ce triplet composé : des solutions potentielles, de l'algorithme d'optimisation et des paramètres associés, sera transmis vers le modèle « Optimizer ». Le modèle « Controler » sera alors désactivé jusqu'à la prochaine demande de solutions, d'algorithmes ou de paramètres provenant du modèle « Optimizer » passant alors dans le second mode de fonctionnement.

Le second mode de fonctionnement permet de répondre à des demandes unitaires : nouvelles solutions aléatoires, nouveaux paramètres, nouveaux algorithmes. La demande d'une nouvelle solution ainsi que la demande de nouveaux paramètres vont générer un seul message sur les ports de sorties correspondants. Ce n'est pas le cas de la demande d'un nouvel algorithme qui elle va générer deux messages distincts. Le premier contenant le nouvel algorithme proposé et le second contenant les paramètres associés proposés.

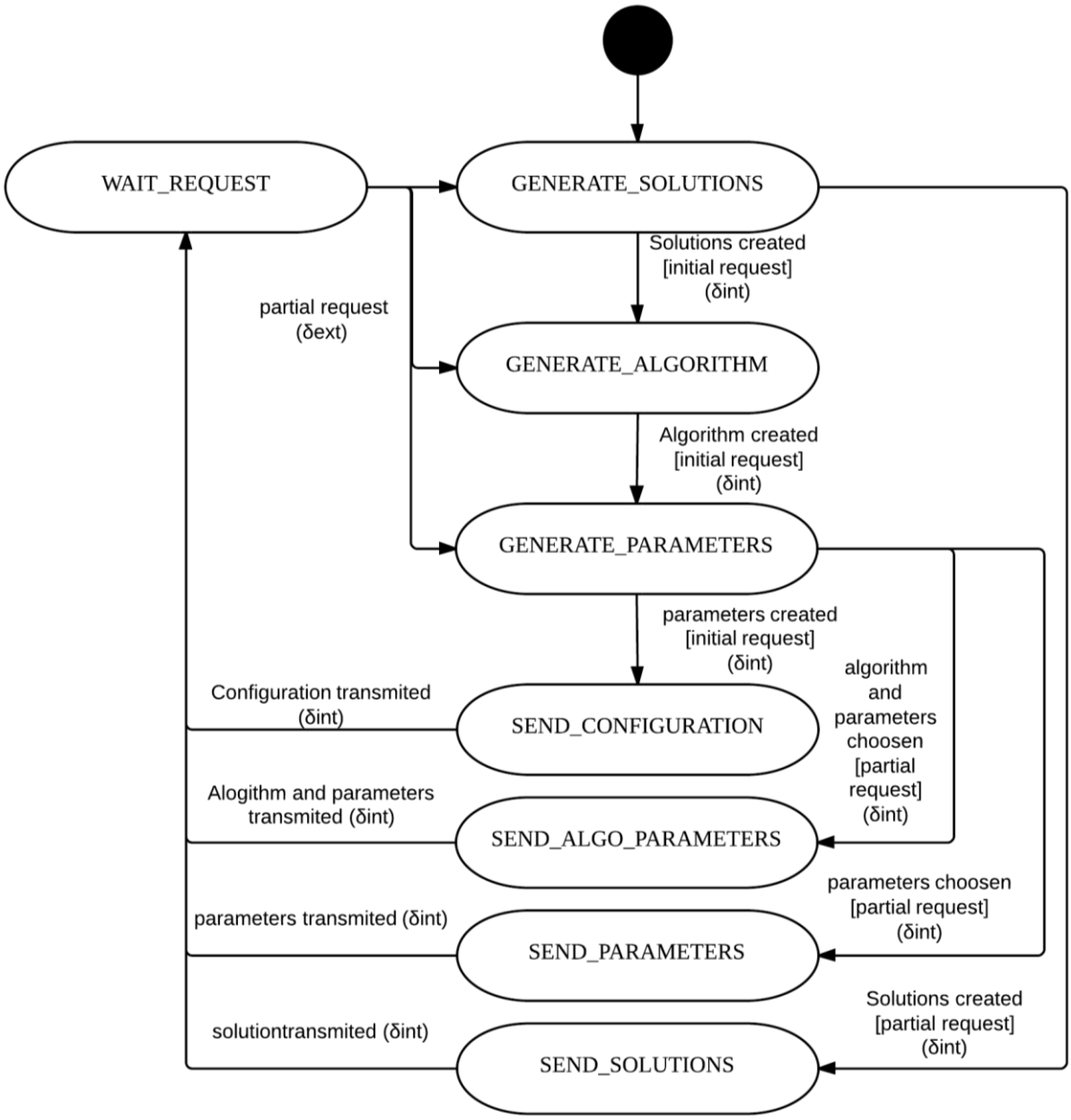


Figure 68 Diagramme UML Etats Transitions : « Controler » (contrôleur)

X	{INsolutions, INalgorithm, INparameter}	
Y	{ OUTsolutions, OUTalgorithm, OUTparameter }	
S	s ϵ { GENERATE_SOLUTIONS, GENERATE_ALGORITHM, GENERATE_PARAMETERS, SEND_CONFIGURATION, SEND_ALGO_PARAMETERS, SEND_PARAMETER, SEND_SOLUTION WAIT_REQUEST }	
δext (S, X)	parameters ← newParameters() s ← SEND_PARAMETER	If messagePort = INParameters

	solutions← newSolution() s ← SEND_SOLUTION	If messagePort = INSolutions
	algorithm ← newAlgorithm() s ← SEND_ALGO_PARAMETERS	If messagePort = INAlgorithm
δint(S)	s = WAIT_REQUEST	If s = SEND_PARAMETER
	s = WAIT_REQUEST	If s = SEND_SOLUTION
	s = WAIT_REQUEST	If s = SEND_ALGO_PARAMETERS
	s = WAIT_REQUEST	If s = SEND_CONFIGURATION
λ(S,Y)	((solutions,OUTsolutions),(parameters,OUTparameters),(algoritm,OUTAlgorithm))	If s = SEND_CONFIGURATION
	((solutions,OUTsolutions))	If s = SEND_SOLUTIONS
	((parameters,OUTparameters))	If s = SEND_PARAMETER
	((algoritm,OUTAlgorithm), (parameters,OUTparameters))	If s = SEND_ALGO_PARAMETERS
ta(S)	0	If s = SEND_CONFIGURATION
	0	If s = SEND_SOLUTIONS
	0	If s = SEND_PARAMETER
	0	If s = SEND_ALGO_PARAMETERS
	∞	If s = WAIT_REQUEST

Tableau 18 Formalisation DEVS du modèle « Controler » (Contrôleur)

4.2.1.2. Le modèle atomique DEVS « Optimizer » (Optimiseur)

Le modèle « Optimiser » dont la représentation graphique est décrite dans la Figure 69, coordonne l'évaluation externalisée des solutions proposées ainsi que leur modification. Il se compose de deux groupes de ports d'entrées et de deux groupes de ports de sorties. Le premier groupe de ports d'entrées permet la réception des différents éléments (solutions, algorithmes, paramètres) nécessaires à l'optimisation. Le second groupe de ports d'entrées permet la réception de l'interprétation finale des résultats collectés lors de la simulation du problème en fonction des paramètres proposés par la solution potentielle évaluée. Pour les ports de sorties, le premier groupe permet l'émission de demandes de nouveaux éléments (solutions, algorithmes, paramètres) vers le « controler ». Le second groupe des ports de sorties permet quant à lui l'externalisation de l'évaluation vers le ou les « adapters».

Comme nous pouvons le voir sur le diagramme de la Figure 70, lors de la phase d'initialisation, le modèle « optimizer » est inactif et attend l'envoi de son algorithme d'optimisation et des paramètres associés, par le modèle « controler ». Une fois cette configuration reçue et intégrée dans son comportement, le modèle change d'état et passe alors en phase d'évaluation. Durant cette phase, le modèle va transmettre la première solution non-évaluée (ou l'ensemble des solutions dans le cas de simulations stochastiques) qu'il possède vers le modèle « adapter » en vue de rendre celle-ci compatible avec les entrées du modèle représentant le problème étudié. Le modèle passera alors en attente, tant que celui-ci ne collectera pas sur chacun de ses ports de résultats une valeur émise par les interpréteurs correspondants. Une fois que toutes les solutions présentes en mémoire sont évaluées, deux possibilités s'offrent au modèle :

- La première consiste à passer en mode « OPTIMISE ». Les différentes solutions vont alors être perturbées, croisées, mixées en fonction des évaluations qui leurs ont été attribuées lors des états précédents. Une fois cette étape terminée, le modèle repassera en mode « EVALUATION » et itérera.
- La seconde consiste à demander une nouvelle configuration partielle ou totale au modèle « Controler ». Cette requête pourra contenir une demande de nouvelles solutions aléatoires, de nouveaux paramètres ou encore de nouvel algorithme. Son émission est déclenchée lorsque la vitesse d'amélioration des solutions observées par le modèle « optimizer » est insuffisante. Ce comportement est présenté dans le Tableau 19.

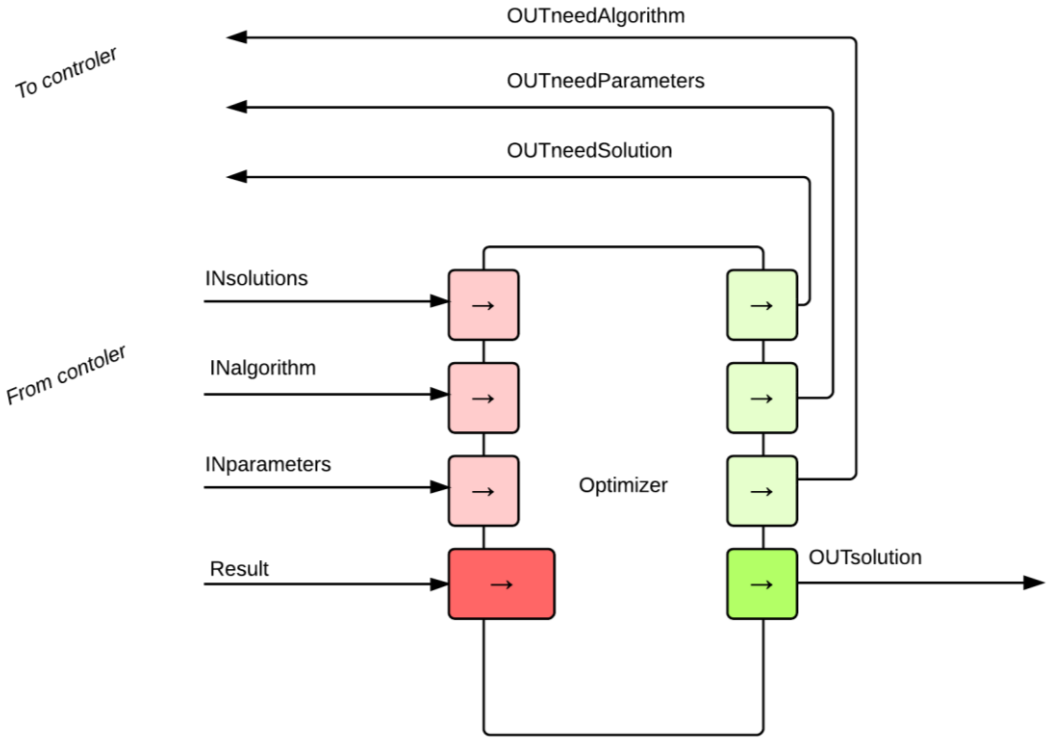


Figure 69 Modèle atomique « Optimizer » (optimiseur)

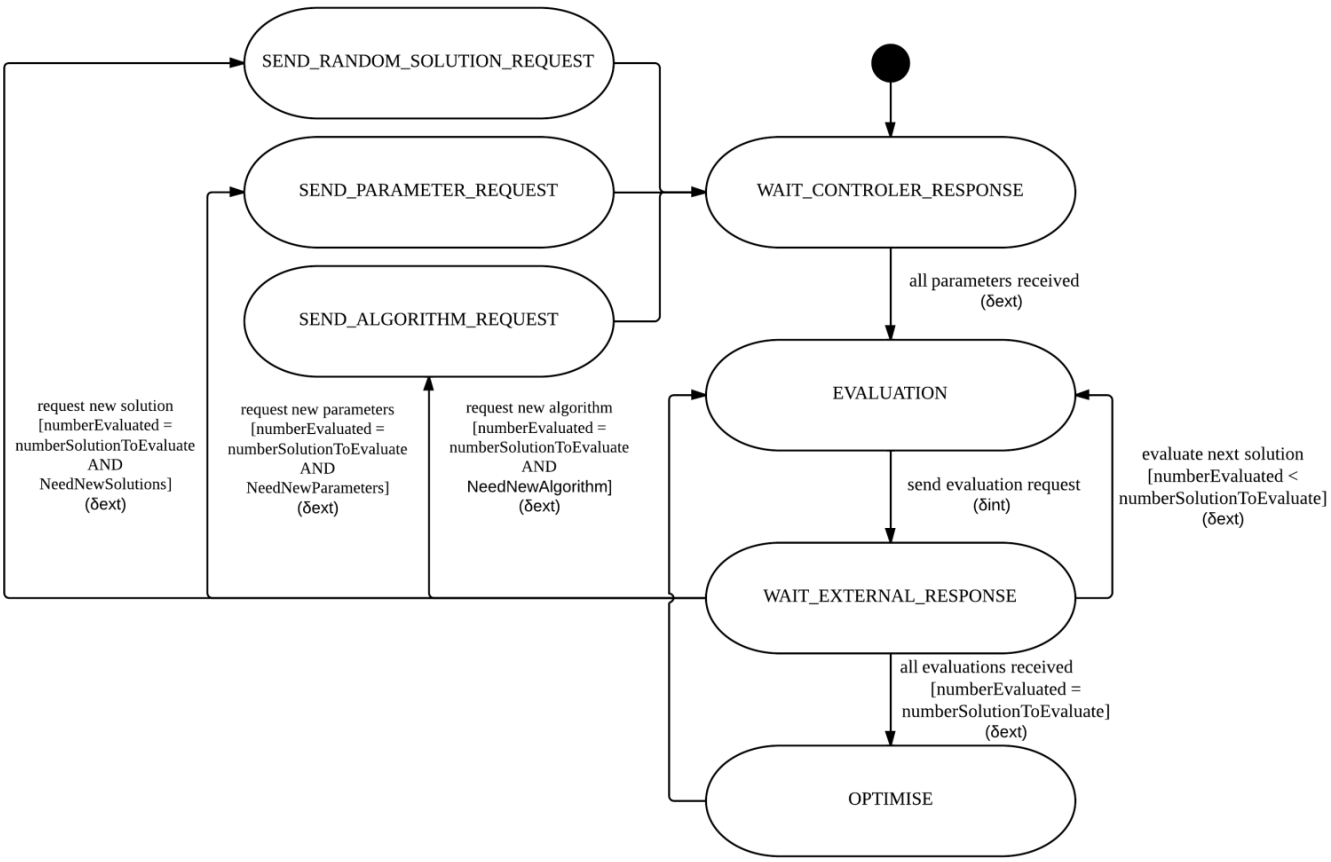


Figure 70 Diagramme UML Etats Transitions : « Optimiser » (optimiseur)

X	{INsolutions, INalgorithm, INparameters, INresult1, INresult2, InresultN}	
Y	{ OUTneedAlgorithm, OUTneedParameters, OUTneedSolution, OUTSolution }	
S	s ϵ { WAIT_CONTROLER_RESPONSE, EVALUATION, WAIT_EXTERNAL_RESPONSE, OPTIMIZE, SEND_ALGORITHM_REQUEST, SEND_PARAMETER_REQUEST, SEND_RANDOM_SOLUTION_REQUEST}	
δext (S, X)	solution ← integrateSolution(message)	If messagePort = INsolutions
	algorithm ← message	If messagePort = INalgorithm
	parameters ← message	If messagePort = INparameters
	result[currentSolution] ← result ← 0.2 × message	If messagePort = INresult1
	result[currentSolution] ← result + 0.5 × message	If messagePort = INresult2
	result[currentSolution] ← result + 0.3 × message	If messagePort = INresult3
	s = OPTIMIZE	If allResultsCollected() If noNeeds()
	s = SEND_RANDOM_SOLUTION_REQUEST	If needsNewSolutions()
	s = SEND_PARAMETER_REQUEST	If needsNewParameters()
	s = SEND_ALGORITHM_REQUEST	If needsNewAlgorithm()
δint(S)	s = WAIT_EXTERNAL_RESPONSE	If s = EVALUATION
	s = WAIT_CONTROLER_RESPONSE	If s = SEND_RANDOM_SOLUTION
	s = WAIT_CONTROLER_RESPONSE	If s = SEND_PARAMETER_REQUEST
	s = WAIT_CONTROLER_RESPONSE	If s = SEND_ALGORITHM_REQUEST

λ(S,Y)	((currentSolution, OUTSolution))	If s = SEND_CONFIGURATION
	((numberNeeded, OUTneedSolution))	If s = SEND_SOLUTIONS
	((algorithm, OUTneedAlgorithm))	If s = SEND_PARAMETER
	((parameters, OUTneddParameters))	If s = SEND_ALGO_PARAMETERS
ta(S)	∞	If s = WAIT_EXTERNAL_RESPONSE
	∞	If s = WAIT_CONTROLER_RESPONSE
	0	If s = SEND_RANDOM_SOLUTION
	0	If s = SEND_PARAMETER_REQUEST
	0	If s = SEND_ALGORITHM_REQUEST
	0	If s = EVALUATION

Tableau 19 Formalisation DEVS du modèle « Optimizer » (optimiseur)

4.2.2. Modélisation de « l'évaluation externalisée »

4.2.2.1. Les modèles atomiques DEVS « adapters » (Adaptateurs)

Les « adapters» permettent de décoder une solution encodée afin de la rendre compatible avec les ports d'entrées du modèle étudié. Quatre catégories « d'adapters » sont disponibles : les « canonical adapter » (adaptateurs classiques), les «temporized adapters » (adaptateurs temporisés), les « spacialized adapters » (adaptateurs spatialisés) et enfin les adaptateurs « temporized and spacialized » (adaptateurs temporisés et spatialisés). Chacune de ces catégories possède ses spécificités.

Au niveau de la configuration et du nombre de ports deux alternatives sont possibles. La première possède un port de sortie et un port d'entrée comme cela est visible sur la Figure 71. Elle est utilisée dans la grande majorité des cas. La seconde possède un port d'entrée et plusieurs ports de sortie, comme nous pouvons le voir sur la Figure 72. Cette configuration permet la répartition de

messages grâce à l'utilisation de différents ports. L'utilisation de ces deux types de configurations est résumée dans le Tableau 20. Il est important de préciser que l'unicité du port d'entrée commune à tous ces modèles permet leur réutilisation pour différentes applications.

	Nombre de port d'entrées	Nombre de port de sortie	
Canonical adapter	1	1	
Temporized adapter	1	1	
Spacialized adapter	1	1	N (répartition géographique)
Temporised and Spacialized adapter	1	1	N (répartition géographique)

Tableau 20 Configuration et nombre de ports des différents adaptateurs

Au niveau de l'éxecution des modèles, différentes transitions sont définies pour chacune des quatre catégories « d'adapters ». Lors des simulations, les « canonical adapters » sont initalisés en état d'attente (WAIT_ADAPT_REQUEST). Lorsque ceux-ci recoivent un message, le modèle s'active et traduit la solution encodée (TRANSLATE_VALUES). Une fois que cette traduction est terminée, le modèle transmet la solution (SEND_ADAPT_RESPONSE) vers le modèle representant le problème étudié. Une fois cet envoi terminé, le modèle se repositionne en attente de nouvelles traductions (WAIT_ADAPT_REQUEST) jusqu'à la réception d'une nouvelle demande. Ces transitions d'états sont resumées dans la Figure 73 et ont été impléméntées en respectant les spécifications du formalisme DEVS, comme cela est précisé dans le Tableau 21.

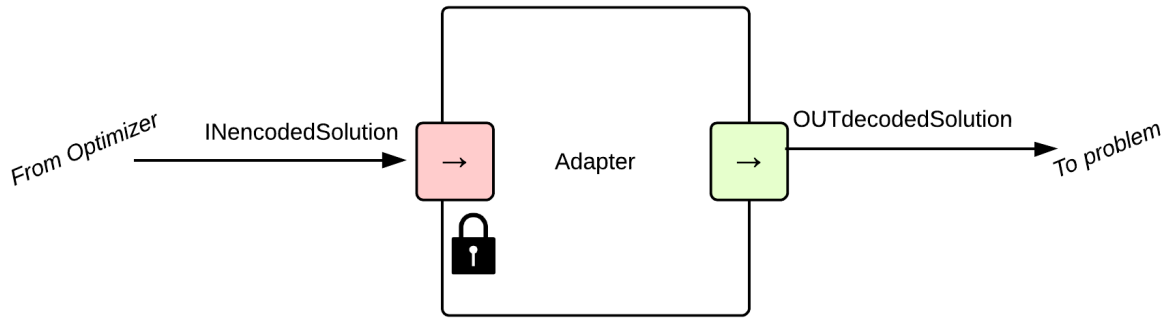


Figure 71 Modèle atomique « Adapter » (adaptateur)

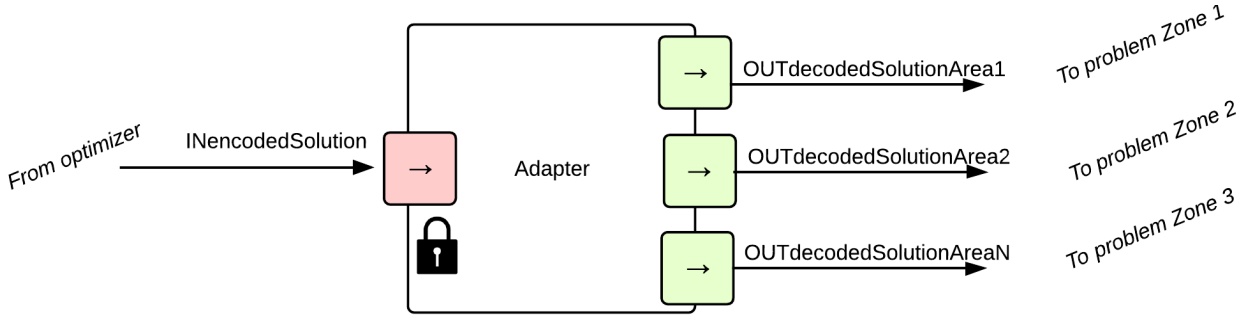


Figure 72 Modèle atomique « Adapter » (adaptateur) avec répartition géographique des sorties

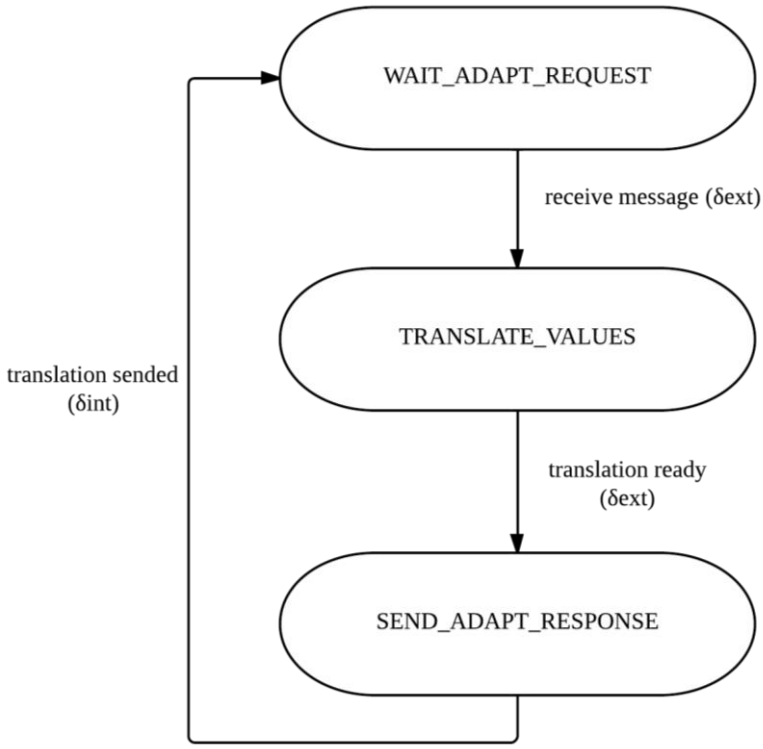


Figure 73 Diagramme UML Etats Transitions « Canonical adapter » (adaptateur classique)

X	{INencodedSolution}	
Y	{OUTdecodedSolution}	
S	s ϵ {WAIT_ADAPT_REQUEST, TRANSLATE_VALUES, SEND_ADAPT_RESPONSE}	
δext (S, X)	encodedSolution ← (message, INencodedSolution) decodedValues ← translateValue(encodedSolution)	If s = WAIT_ADAPT_REQUEST
δint(S)	s ← WAIT_ADAPT_REQUEST	
λ(S,Y)	((decodedValues, OUTdecodedValue))	
ta(S)	0	If s = TRANSLATE_VALUES
	∞	If s = SEND_ADAPT_RESPONSE

Tableau 21 Formalisation DEVS du modèle « Canonical adapter » (adaptateur classique)

La seconde catégorie d'adaptateurs dits : « spacialized adapters » permet la traduction de solutions encodées vers un ensemble de valeurs temporisées sous la forme de couples « Valeur-Temps ». Leurs différentes transitions d'états sont résumées sur la Figure 74. À l'initialisation de la simulation, ces modèles attendent la réception d'un message (WAIT_ADAPT_REQUEST). Quand une demande d'adaptation est reçue, ceux-ci vont dans un premier temps traduire les différentes valeurs (TRANSLATE_VALUES) puis traduire les différents temps (TRANSLATE_TIMES) à partir de deux intervalles définis à l'intérieur du modèle. Les différentes valeurs vont être transmises les unes après les autres (SEND_ADAPTATION_RESPONSE et WAIT_NEXT_SEND) jusqu'à ce qu'il n'y

ait plus aucune valeur à transmettre. Le modèle rebascule alors en attente d'une nouvelle demande d'adaptation (WAIT_ADAPT_REQUEST). Ce comportement est résumé dans le Tableau 22.

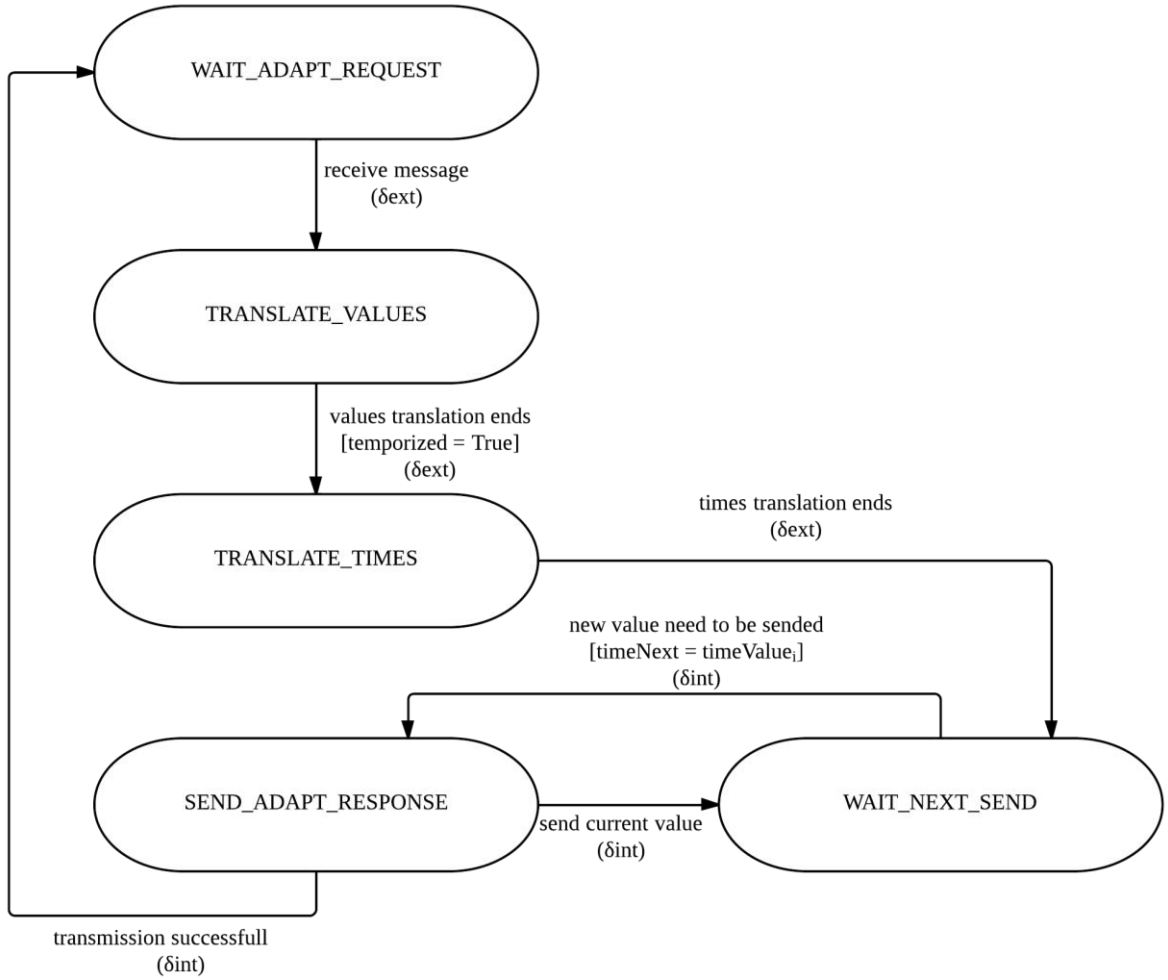


Figure 74 Diagramme UML Etats Transitions « Temporized adapter » (adaptateur temporisé)

X	{INencodedSolution}	
Y	{OUTdecodedSolution}	
S	s ϵ {WAIT_ADAPT_REQUEST, TRANSLATE_VALUES, TRANSLATE_TIMES, SEND_ADAPT_RESPONSE, WAIT_NEXT_SEND}	
δext (S, X)	s ←TRANSLATE_VALUES encodedValues ← valuesPart((message, INencodedSolution)) decodedValues ← translateV (encodedSolution) s ← TRANSLATE_TIMES encodedTimes ← timesPart((message, INencodedSolution)) decodedTimes ← translateT(encodedTimes)	If s = WAIT_ADAPT_REQUEST

	i ← 0 s ←WAIT_NEXT_SEND	
δint(S)	s ← SEND_ADAPT_RESPONSE i ← i + 1	If s = WAIT_NEXT_SEND Et i < taille(decodedValues)
	s ← WAIT_NEXT_SEND	If s = SEND_ADAPT_RESPONSE Et i < taille(decodedValues)
	s ← WAIT_ADAPT_RESPONSE	If s = SEND_ADAPT_RESPONSE Et i = size(decodedValues)
λ(S,Y)	((decodedValues[i], OUTdecodedvalue))	
ta(S)	0	If s = SEND_ADAPT_RESPONSE
	decodedTimes[i]	If s = WAIT_NEXT_SEND
	∞	If s = WAIT_ADAPT_RESPONSE

Tableau 22 Formalisation DEVS du modèle « Temporized adapter» (Adaptateur temporisé)

La troisième catégorie « d'adapters » regroupe les « adapters » dits : « Spacialized adapters ». Ces modèles permettent le décodage des solutions intégrant un aspect spatial formant ainsi un couple « Valeur-Position ». Cela est illustré sur le diagramme de la Figure 75. Lors de la simulation, le modèle attend une demande d'adaptation (WAIT_ADAPT_REQUEST). Une fois cette demande reçue, la ou les valeurs sont traduites (TRANSLATE_VALUES) ainsi que les coordonnées géographiques (TRANSLATE_LOCATIONS). Elles sont ensuite transmises vers le modèle représentant le problème étudié (SEND_ADAPT_RESPONSE). « L'adapter » passe alors de nouveau en état d'attente (WAIT_ADAPT_REQUEST). Ces différentes transitions sont synthétisées dans la Figure 75 et leurs structurations dans le formalisme DEVS sont présentées dans le Tableau 23. Dans ce tableau, deux définitions différentes des ports et des fonctions de sorties sont présentes. Le comportement de la fonction de sortie λ dépend en effet du nombre de ports qu'elle doit gérer.

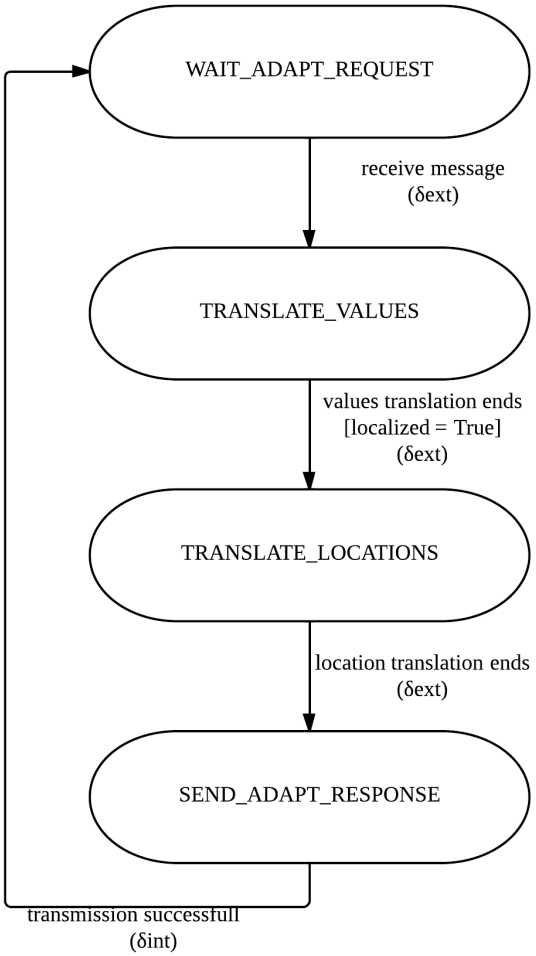


Figure 75 Diagramme UML Etats Transitions « Spacialized adapter » (Adaptateur spatialisé)

X	{INencodedSolution}	
Y [1 port de sortie]	{OUTdecodedSolution}	
Y [n ports de sortie]	{OUTdecodedSolutionArea1, OUTdecodedSolutionArea2, OUTdecodedSolutionAreaN}	
S	s ϵ {WAIT_ADAPT_REQUEST, TRANSLATE_VALUES, TRANSLATE_LOCATIONS, SEND_ADAPT_RESPONSE}	
δext (S, X)	s = TRANSLATE_VALUES encodedValues ← valuesPart((message, INencodedSolution)) decodedValues ← translateV (encodedSolution)	If s = WAIT_ADAPT_REQUEST

	s ← TRANSLATE_LOCATIONS encodedLocations ← locationPart((message, INencodedSolution)) decodedLocations ← translateL(encodedLocations) s ← SEND_ADAPT_RESPONSE	
δint(S)	s ← WAIT_ADAPT_REQUEST	If s = SEND_ADAPT_RESPONSE
λ(S,Y) [1 port de sortie]	message.values ← decodedValues message.locations ← decodedLocations ((message, OUTdecodedValue))	
λ(S,Y) [n ports de sortie]	For i in size(decodedValues) ((decodedValues[i], correspondingZonePort(decodedLocations[i])))	
ta(S)	0	If s = SEND_ADAPT_RESPONSE
	∞	If s = "WAIT_ADAPT_REQUEST"

Tableau 23 Formalisation DEVS du modèle « Spacialized adapter» (adaptateur spatialisé)

Enfin la quatrième catégorie concerne les « Temporized and spacialiez adapters ». Ils peuvent être vus un regroupement, une unification des « temporized adapters » et des « spacialized adapters » dans un unique modèle. Les différents états et transitions sont résumés dans la Figure 76 et leur description dans le formalisme DEVS est définie dans le Tableau 24 ainsi que les nuances inhérentes à la « géolocalisation par coordonnées » et la « géolocalisation par ports ».

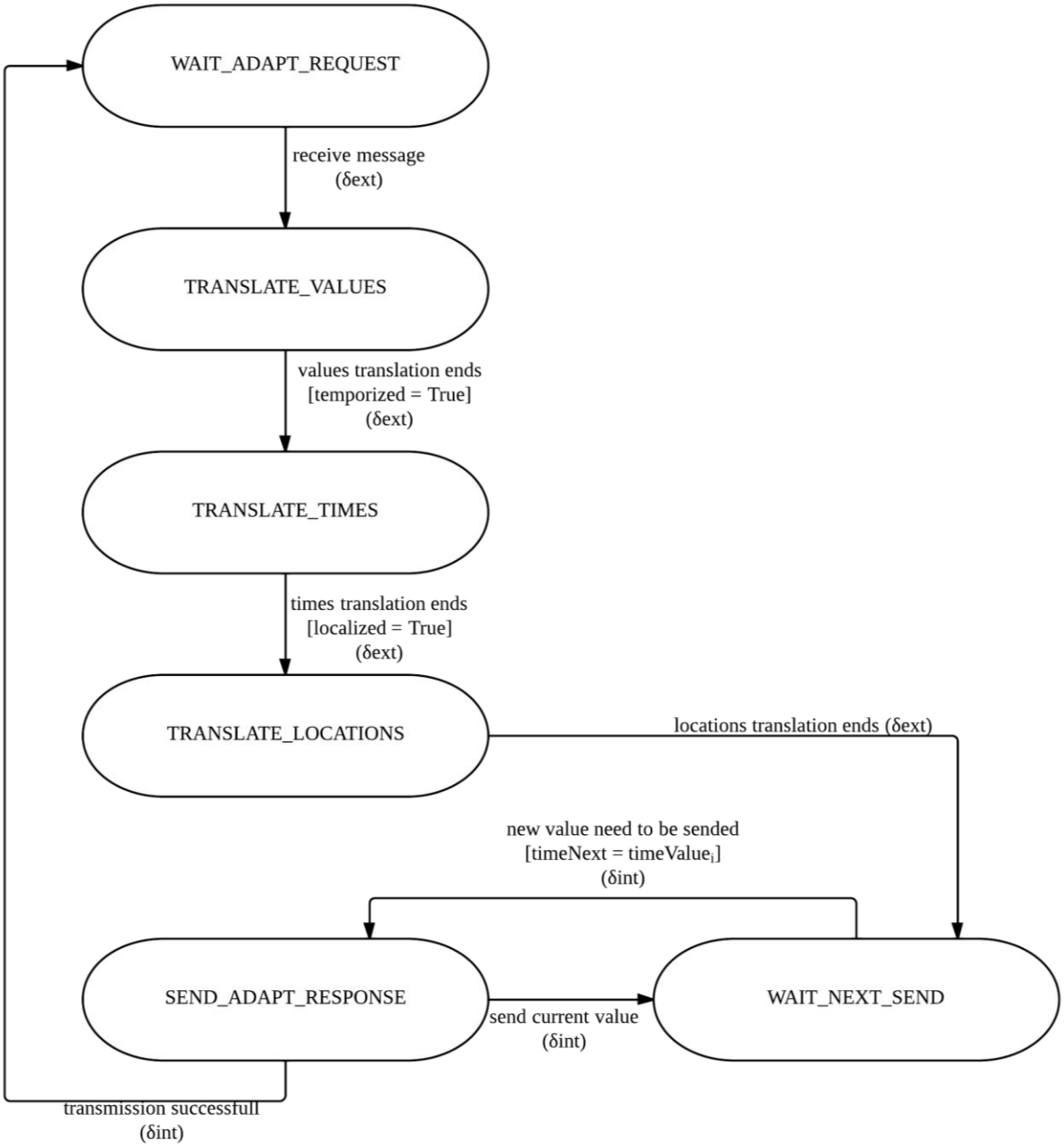


Figure 76 Diagramme UML Etats Transitions « Temporized & spacialized adapter» (adaptateur temporisé et spatialisé)

X	{INencodedSolution}
Y [1 port de sortie]	{OUTdecodedSolution}
Y [n ports de sortie]	{OUTdecodedSolutionArea1, OUTdecodedSolutionArea2, OUTdecodedSolutionAreaN}
S	s ϵ {WAIT_ADAPT_REQUEST, TRANSLATE_VALUES, TRANSLATE_TIMES,

	TRANSLATE_LOCATIONS, SEND_ADAPT_RESPONSE, WAIT_NEXT_SEND}	
δext (S, X)	s ← TRANSLATE_VALUES encodedValues ← valuesPart((message, INencodedSolution)) decodedValues ← translateV (encodedValues) s ← TRANSLATE_LOCATIONS encodedLocations ← locationsPart((message, INencodedSolution)) decodedLocations ← trasnlateL(encodedLocations) s ← TRANSLATE_TIMES encodedTimes ← timesPart((message, INencodedSolution)) decodedTimes ← trasnlateT(encodedTimes)	If s = WAIT_ADAPT_REQUEST
δint(S)	s ← SEND_ADAPT_RESPONSE i ← i + 1	If s = WAIT_NEXT_SEND Et i < taille(decodedValues)
	s ← WAIT_NEXT_SEND	If s = SEND_ADAPT_RESPONSE Et i < taille(decodedValues)
	s ← WAIT_ADAPT_REQUEST	If s = SEND_ADAPT_RESPONSE Et i = taille(decodedValues)
λ(S,Y) [1 port de sortie]	message.values ← decodedValues[i] message.locations ← decodedLocations[i] ((message, OUTdecodedValue))	
λ(S,Y) [n ports de sortie]	For i in size(decodedValues) ((decodedValues[i], portPourZone(decodedLocations[i])))	
ta(S)	0	If s = SEND_ADAPT_RESPONSE
	decodedTime[i]	If s = WAIT_NEXT_SEND
	∞	If s = WAIT_ADAPT_RESPONSE

Tableau 24 Formalisation DEVS du modèle « Temporized & spacialized adapter» (Adaptateur temporisé et spatialisé)

4.2.2.2. Les modèles atomiques DEVS « interpreters » (Interpréteurs)

Les modèles atomiques « interpreters » permettent l'analyse des résultats de simulation du modèle étudié en fonction des décisions qui lui ont été soumises sous la forme de paramètres d'entrées. Comme cela est présenté sur la Figure 78 et la Figure 77, le nombre de ports d'entrées de ces modèles est libre. Cette multitude de ports d'entrée se justifie par le besoin, pour certaines

applications, d'analyser de manière corrélée des résultats provenant de différentes sorties. Le nombre de ports de sorties quant à lui est compris entre un et deux. Un seul port de sortie est utilisé dans le cas des «unit interpreters » recevant des valeurs à un seul instant. Deux ports de sorties sont utilisés pour les « multiple interpreters » recevant plusieurs valeurs à des temps distincts.

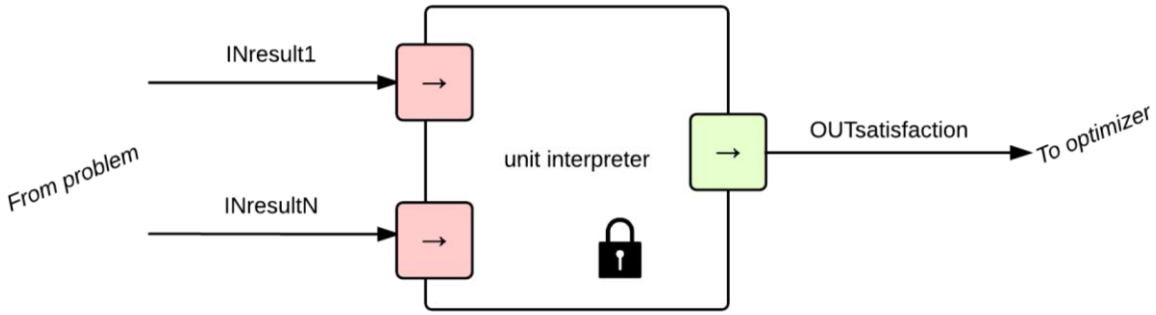


Figure 77 Modèle atomique «unit interpreter » (interpréteur unitaire)

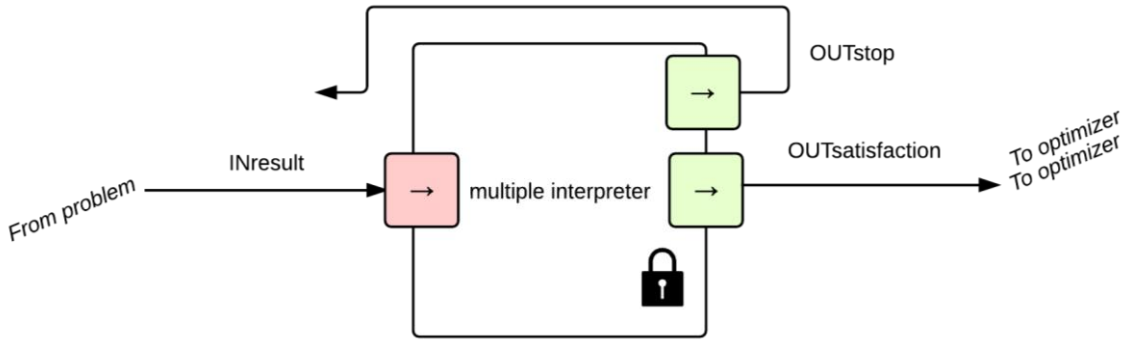


Figure 78 Modèle atomique « multiple interpreter » (interpréteur multiple) avec corrélation des résultats

Comme cela a été dit précédemment, deux types « d'interpreters » sont disponibles : les « unit interpreters » et les « multiple interpreters ». Les transitions d'états des « unit interpreters » sont présentées dans la Figure 79. Comme on peut le voir, lors de l'initialisation de la simulation « l'interpreter » est en attente d'une demande d'interprétation (WAIT_INTERPRET_REQUEST). Une fois qu'une demande est reçue, celle-ci est directement analysée (ANALYSE). Une fois la phase d'analyse terminée, le modèle transmet le résultat vers le modèle « Optimizer » (SEND_INTERPRET_RESPONSE) et attend de nouvelles demandes d'interprétation (WAIT_INTERPRET_REQUEST). Ce comportement a été implémenté sous la forme d'un modèle atomique dont le comportement est résumé dans le Tableau 25.

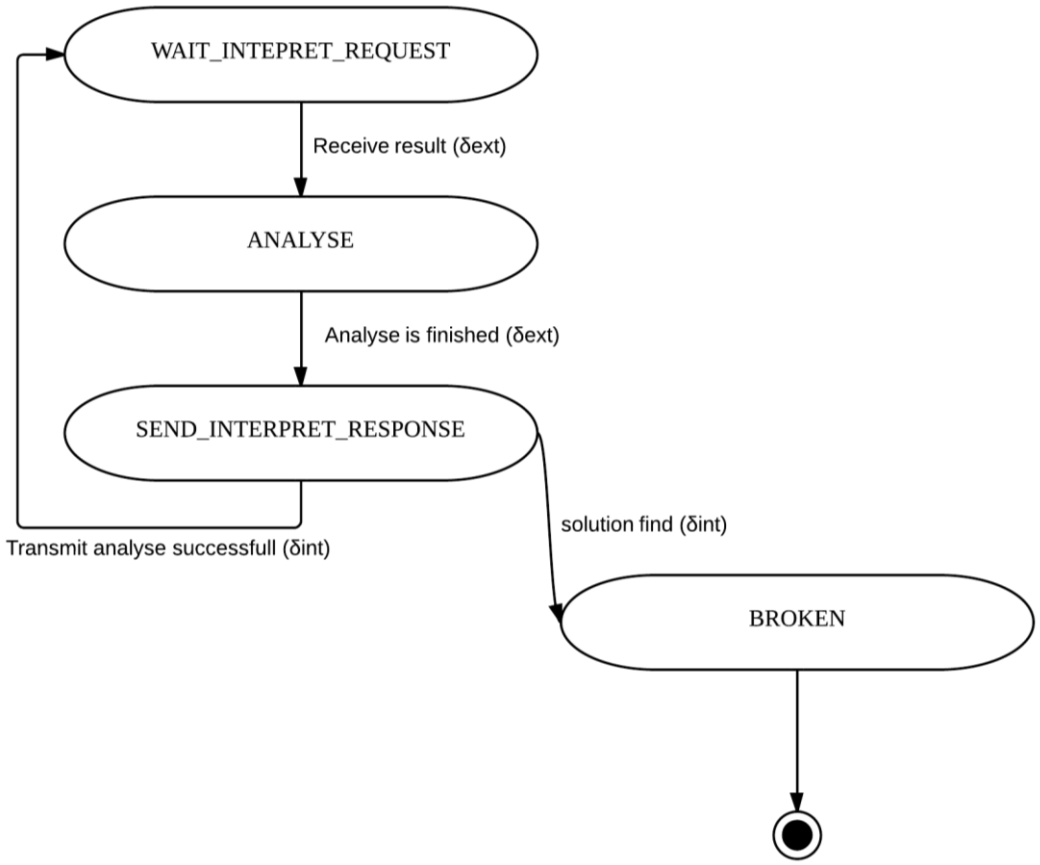


Figure 79 Diagramme UML Etats Transitions « unit interpreter» (interpréteur uniataire)

X [1 port entrée]	{INresult}	
X [N ports entrées]	{INreusult1, INresult2}	
Y	{OUTsatisfaction}	
S	s ϵ {WAIT_INTERPRET_REQUEST, ANALYSE, SEND_INTERPRET_RESPONSE, BROKEN}	
δext (S, X) [1 port entrée]	s ← ANALYSE result = ((message, INencodedSolution)) satisfaction ← analyse(result) s ← SEND_INTERPREAT_RESPONSE	If s = WAIT_INTERPRET_REQUEST
δext (S, X) [n ports entrées]	s ← ANALYSE listResults ← readAllMessages() satisfaction ← analyse(listResults) s ← SEND_INTERPREAT_RESPONSE	If s = WAIT_INTERPRET_REQUEST

δint(S)	s ← BROKEN	If isSolutionFind() = True
	s ← WAIT_INTERPRETN_REQUEST	If isSolutionFind() = False
λ(S,Y)	((satisfaction, OUTsatisfaction))	
ta(S)	0	If s = SEND_INTERPRETATION_RESPONSE
	∞	If s = WAIT_INTERPRETATION_REQUEST

Tableau 25 Formalisation DEVS du modèle « unit interpreter»

Le second type « d'interpreter » regroupe les « multiple interpreters » multiples. Les différents états et transitions de ce type de modèle sont décrits dans la Figure 80. Comme nous pouvons le voir, ces modèles sont initialement en attente d'une demande d'interprétation des résultats (WAIT_INTERPRET_REQUEST). Lorsque qu'un premier résultat de simulation parvient sur le port d'entrée de « l'interpreter », celui-ci va collecter les différentes sorties produites par le modèle représentant le problème (COLLECT_IN_PROCESS). Cette collecte continuera jusqu'à un certain seuil ou un nombre limité de valeurs (voir page 93). Lorsque ce seuil sera franchi ou lorsque le nombre de valeurs sera dépassé, la simulation du modèle étudié sera interrompue par l'émission d'un « message stop » (SEND_STOP_RESPONSE) en direction du modèle représentant le problème étudié. Dans le même temps, les résultats collectés seront analysés (ANALYSE). Une fois cette phase d'analyse terminée, le modèle transmettra le résultat de son analyse vers le modèle « Optimizer » (SEND_INTERPRET_RESPONSE), si un seul « interpreter » est utilisé ou vers le « regroupeur d'interprétations » si plusieurs « interpreters » sont utilisés (pour une optimisation multicritère). La transcription de ces différentes transitions dans le formalisme DEVS a été réalisée, comme présenté dans le Tableau 26.

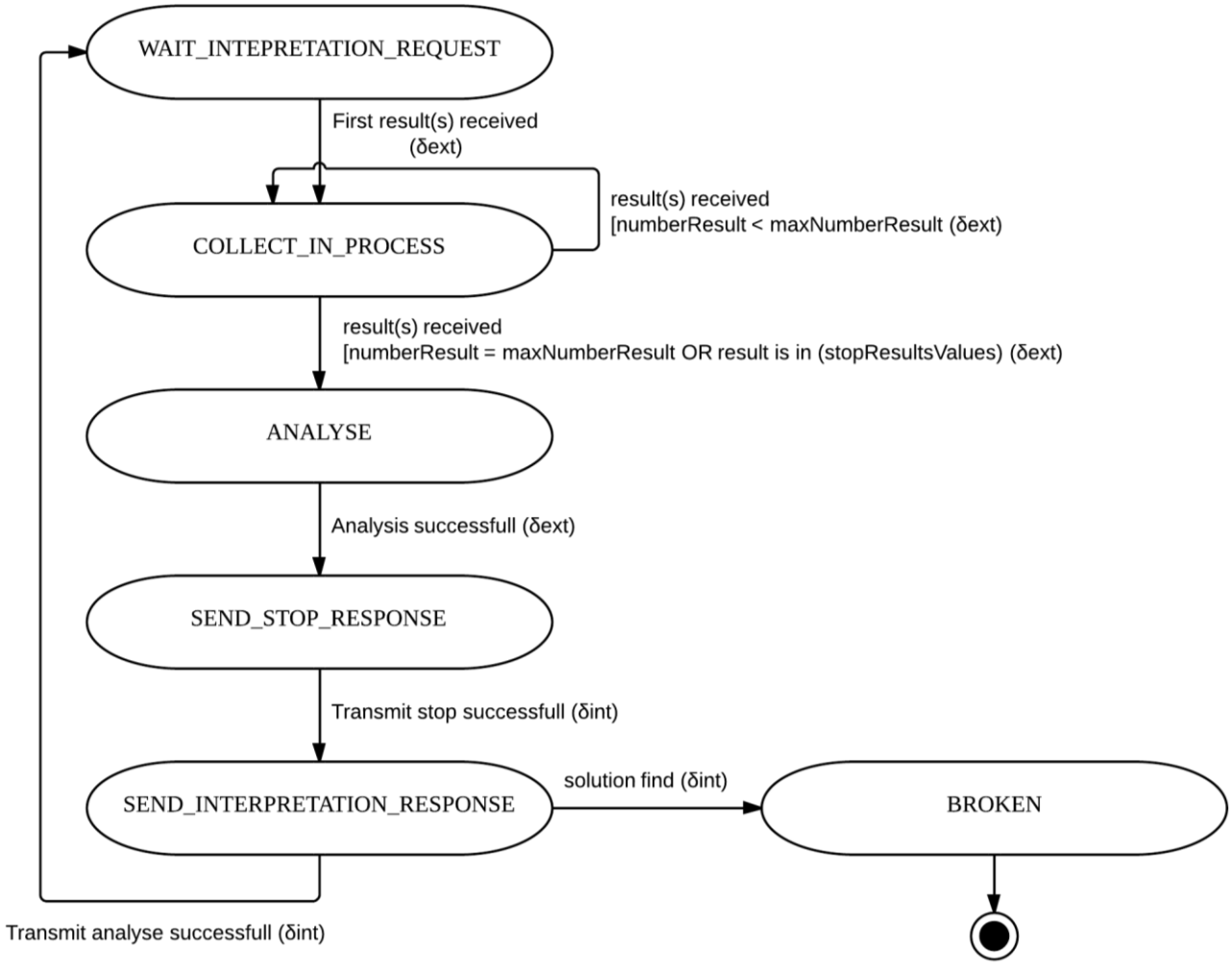


Figure 80 Diagramme UML Etats Transitions « multiple interpreter» (interpréteur multiple)

X [1 port entrée]	{INresult}	
X [N ports entrées]	{INreusult1, INresult2}	
Y	{OUTsatisfaction, OUTstop}	
S	s ϵ {WAIT_INTERPRET_REQUEST, COLLECT_IN_PROCESS, SEND_STOP_RESPONSE, ANALYSE, SEND_INTERPRET_RESPONSE, BROKEN}	
δext (S, X) [1 port entrée]	results ← saveResults((message,INresult)) s ← COLLECT_IN_PROCESS	If s = WAIT_INTERPRET_REQUEST
	results ← saveResults((message,INresult))	If s = COLLECT_IN_PROCESS
	s ← ANALYSE	If size(result) = sizeMemory

	Satisfaction ← analyse(results) S ← SEND_STOP_RESPONSE	
δext (S, X) [N port entrée]	For port in INPorts result ← saveResults((message, port)) S ← COLLECT_IN_PROCESS	If s = WAIT_INTERPRET_REQUEST
	For port in INPorts result ← saveResults((message, port))	If s = COLLECT_IN_PROCESS
	S ← ANALYSE satisfaction ← analyse(results) S ← SEND_STOP_RESPONSE	If size(result) = sizeMemory
δint(S)	S ← BROKEN	If isSolutionFind() = True
	S ← SEND_INTERPRET_RESPONSE	If s = SEND_STOP_RESPONSE
	S ← WAIT_INTERPRET_REQUEST	If s = SEND_INTERPRET_RESPONSE
λ(S,Y)	((stopFlag, OUTStop))	If s = END_STOP_RESPONSE
	((satisfaction, OUTsatisfaction))	If s = SEND_INTERPRET_RESPONSE
ta(S)	0	If s = SEND_STOP_RESPONSE
	0	If s = SEND_INTERPRET_RESPONSE
	∞	If s = WAIT_INTERPRET_REQUEST
	∞	If s = COLLECT_IN_PROCESS
	∞	If s = BROKEN

Tableau 26 Formalisation DEVS du modèle « mutilple interpreter»

4.3 Conclusion

Dans ce chapitre, nous avons présenté notre architecture d'optimisation via simulation orientée modèles DEVS. Son objectif est de permettre l'optimisation paramétrique de systèmes via des simulations de modèles DEVS de manière générique et performante. La description de cette architecture s'est déroulée en deux étapes.

Dans un premier temps, nous avons isolé les différents modèles qui composent notre architecture et mis en lumière leurs apports en comparaison des approches analogues présentes dans la littérature. Contrairement à celles-ci notre architecture intègre l'optimisation à l'intérieur de deux modèles simulés : « l'optimiseur » et le « contrôleur ». Cela permet de dynamiser l'optimisation par des changements d'algorithmes, de paramètres et de solutions. Ces changements sont possibles par une analyse de la progression de l'optimisation par « l'optimiseur » qui, lorsqu'elle est insuffisante d'éclanche des demandes de nouvelles configurations en direction du « contrôleur ». De plus, notre architecture contient de nombreux modèles « adaptateurs » permettant d'assurer la cohérence des échanges entre le modèle d'optimisation et le modèle représentant le problème. Pour cela, les descriptions binaires des solutions gérées par « l'optimiseur » peuvent être transmises vers le modèle du problème étudié en étant converties dans différents types et structures. De plus, des adaptateurs spécifiques permettant la temporisation et/ou la spatialisation de ces différentes solutions sont disponibles pour des problèmes posés dans le temps et l'espace.

Dans un second temps, nous nous sommes concentrés sur la formalisation DEVS des modèles proposés. Pour cela, nous avons choisi de baser notre description à la fois sur une représentation graphique et une description formelle. La représentation graphique des différents états possibles sous la forme de diagrammes UML d'états-transitions nous a semblé être la plus intuitive. A partir des différents schémas obtenus nous avons alors procédé à la formalisation du comportement de nos modèles sous la forme de modèles atomiques en respectant la définition DEVS.

Une fois ces deux étapes terminées, nous sommes passés à la phase d'implémentation de ces différents modèles DEVS et à leur validation sur différentes applications. Celle-ci est décrite dans le chapitre suivant.

Chapitre 5
Validation de l'architecture de modèles DEVS : implémentation et validation

« Longue est la route par le précepte, courte et facile par l'exemple »

Sénèque

L'implémentation de notre architecture de modèles a été réalisée sur la plateforme de développement DEVSimPy (Capocchi et al., 2011). Les différents modèles DEVS réalisés ont été regroupés à l'intérieur d'une librairie de modèles que nous avons nommée « Optimization ». Dans ce chapitre, nous justifierons le choix de ce logiciel et présenterons ses principales caractéristiques, avantages et inconvénients.

Afin de mettre en avant l'aspect générique et général de notre approche, la validation de nos modèles s'effectuera sur trois applications distinctes : l'optimisation paramétrique de fonctions multidimensionnelles et non linéaires, l'optimisation paramétrique d'un déploiement de réseau de capteurs sans fils, et l'optimisation paramétrique d'un traitement médical. Les finalités attendues de ces trois applications sont résumées dans le Tableau 27.

Application	Optimisation externalisée				Optimisation intelligente	
	Adaptateurs classiques + Interpréteurs unitaires	Adaptateurs spatialisés + Interpréteurs unitaires	Adaptateurs temporisés + Interpréteurs multiples	Simulation économe	Choix dynamique d'algorithme et de paramètres	Hybridation des algorithmes
Optimisation paramétrique de fonctions multidimensionnelles	X					
Optimisation paramétrique d'un déploiement de réseau de capteurs sans fils		X			X	X
Optimisation paramétrique d'un traitement médicale			X	X	X	X

Tableau 27 Concepts validés par chacune des applications

La première, l'optimisation paramétrique destinée à trouver les minimums globaux de fonctions mathématiques, permettra de vérifier le bon fonctionnement de notre architecture et de son concept « d'évaluation externalisée » sur des « Benchmark » traditionnellement utilisés dans la littérature pour la comparaison des méthodes d'optimisation.

La seconde concernera l'optimisation d'un déploiement de réseaux de capteurs sans fil. Elle permettra de valider le fonctionnement de notre outil sur une application spatialisée et multicritère. En effet, deux modèles représentant le problème du déploiement, seront utilisés lors de l'optimisation par simulation : la couverture du réseau de capteurs (zone de détection du phénomène observé en fonction des spécificités du terrain) et la connectivité du réseau (maillage, tolérance de pannes, goulot d'étranglement). Cette application nous permettra d'une part de tester le concept « d'optimisation intelligente » et d'autre part d'observer les différents changements de comportement de l'optimiseur en termes d'algorithmes, de paramètres et de solutions). Une comparaison entre les performances générées par un comportement dynamique et un comportement statique de l'optimiseur sera alors réalisée.

Enfin la troisième se basera sur l'optimisation paramétrique de trois traitements médicaux de patients diabétiques, en vue de choisir la molécule ayant le plus fort potentielle de réponse. Elle permettra de valider notre concept de temporisation des variables à optimiser sur un modèle de

glycémie, simulé sur vingt-quatre heures et dans le même temps permettra de tester notre concept de « simulation économe » grâce à l'utilisation d'un « interpréteur multiple ». Au niveau du traitement, nous observerons le niveau d'optimisation de traitement proposée par l'utilisation de notre outil. Au niveau du temps de calcul, nous mesurerons les gains de temps qu'offre le concept de « simulation économe ».

5.1 DEVSimPy

5.1.1. Présentation

DEVSimPy (Capocchi et al., 2011) est une plateforme de modélisation et de simulation Open Source, développée au sein du laboratoire CNRS 6134 SPE (Sciences pour l'environnement). DEVSimPy peut être vu comme une surcouche graphique de la librairie PyDEVS elle-même développée au sein de l'Université de McGill (Bolduc and Vangheluwe, 2001). Elle permet la modélisation et la simulation modèles DEVS avec plusieurs extensions supportées.

DEVSimPy est développé en langage Python dans sa version 2.7 (Drake and Rossum, 2011). Ce langage supporte en effet de nombreuses librairies scientifiques telles que « numpy », « scipy » pour le calcul scientifique et « Matplotlib » pour la visualisation de données. De plus, Python est un langage de programmation orienté objet (POO) basé sur un typage dynamique des variables permettant l'implémentation rapide et structurée de modèles tout en conservant un niveau de performance acceptable.

DEVSimPy supporte plusieurs plugins mais permet aussi d'en intégrer facilement de nouveaux. Ils permettent de faciliter le travail du développeur mais également de visualiser les résultats de simulation. Parmi eux, nous pouvons citer l'exemple du plugin « Blink ». Celui-ci offre la possibilité de visualiser pas à pas les différents messages et transitions se déroulant lors d'une simulation, facilitant ainsi le débogage des modèles. La simulation, quant à elle est automatisée par une génération automatique de l'arbre de simulation inhérente au formalisme.

Le logiciel offre la possibilité d'éditer les modèles en exécutant un éditeur de code intégré. Il permet également, la visualisation de ceux-ci sous forme de graphes, comme cela est montré dans l'exemple de la Figure 81. En cohérence avec le concept de hiérarchisation des modèles, l'utilisateur de l'application dispose de différents niveaux de vue, définissant une imbrication visuelle des différents modèles atomiques DEVS regroupés dans des modèles couplés DEVS. Ces différents modèles se présentent sous la forme d'objets graphiques de type *drag and drop*. L'utilisateur choisit les différents modèles dont il a besoin pour modéliser son système. Ceux-ci sont ensuite interconnectés et enfin simulés. Afin d'éviter des codages répétitifs de modèles identiques, les modèles réalisés sont stockés dans différentes « libraires de modèles » pouvant être mises à jour directement par l'utilisation de web-service. Ce concept permet une organisation cohérente et une mutualisation des différents modèles réalisés. En effet, DEVSimPy permet de profiter du concept de « modularité des modèles » et permet la réutilisation d'un même modèle pour différentes applications. À l'intérieur des librairies de modèles, deux formats de fichiers sont disponibles : les fichiers AMD décrivant les modèles atomiques et les fichiers CMD décrivant les modèles couplés. Cependant, il est également possible d'utiliser directement des fichiers python comme modèles atomiques lors des phases de validations.

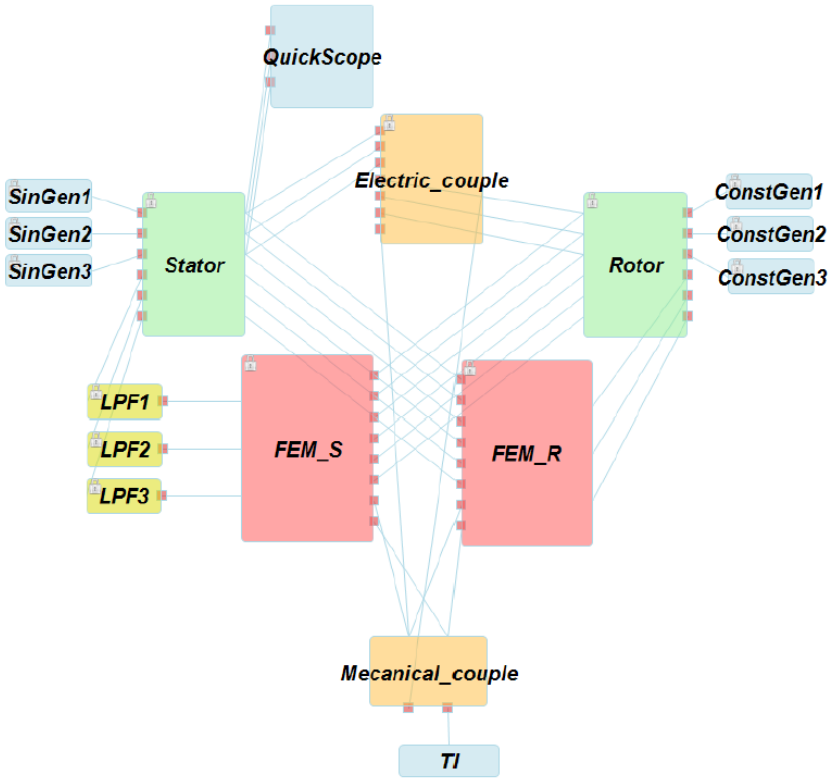


Figure 81 Exemple de système de machine éléctrique modélisé avec DEVSimPy

Afin de valider notre architecture, nous avons implémenté nos modèles sous la forme d'une librairie de modèles DEVS que nous avons appelée librairie « Optimization »

5.1.2. Intégration de l'architecture de modèles proposée

La librairie « Optimization » contient l'ensemble des modèles « d'optimisation intelligente » et « d'évaluation externalisée ». La Figure 82 listes les différents concepts supportés (contrôleur d'optimisation, optimiseur, adaptateur, interpréteurs).

La librairie se décompose en différents paquets basées sur l'architecture précédemment présentée. Le paquet « Problems » contient les différents modèles à optimiser qui n'ont pas forcement été conçus à l'origine pour réaliser des optimisations paramétriques. Deux sous-catégories de modèles sont actuellement présentes : les applications théoriques (e. g. fonction Rastringin) et les applications pratiques (e. g. modèle de couverture). Le paquet « SmartOptimization » (Optimisation intelligente) contient le « controler » (contrôleur) ainsi que « l'optimiser » (optimiseur) associé. Le paquet « ExternalEvaluation » (Evaluation externalisée) contient les différents « adapters » (adaptateurs) nécessaires pour traduire la représentation générique de la solution vers les ports d'entrées du modèle étudié. Dans ce sous-paquet, trois sous catégories sont présentes : « Temporized » (adaptateurs temporisés), « Spacialized » (adaptateurs spatialisés), « Canonical » (adaptateurs classiques) et le modèle « slicer » (découpeur de code). Le paquet « ExternalEvaluation » (Evaluation externalisée) contient également un sous-paquet « Interpreters » (interpréteurs). Il contient les différents interpréteurs permettant la collecte et l'analyse des sorties du modèle étudié. Deux sous-catégories ont été isolées : « unit » pour les interpréteurs unitaires et « multi » pour les interpréteurs multiples ainsi qu'un modèle « Grouper » pour le « regroupeur multicritères.

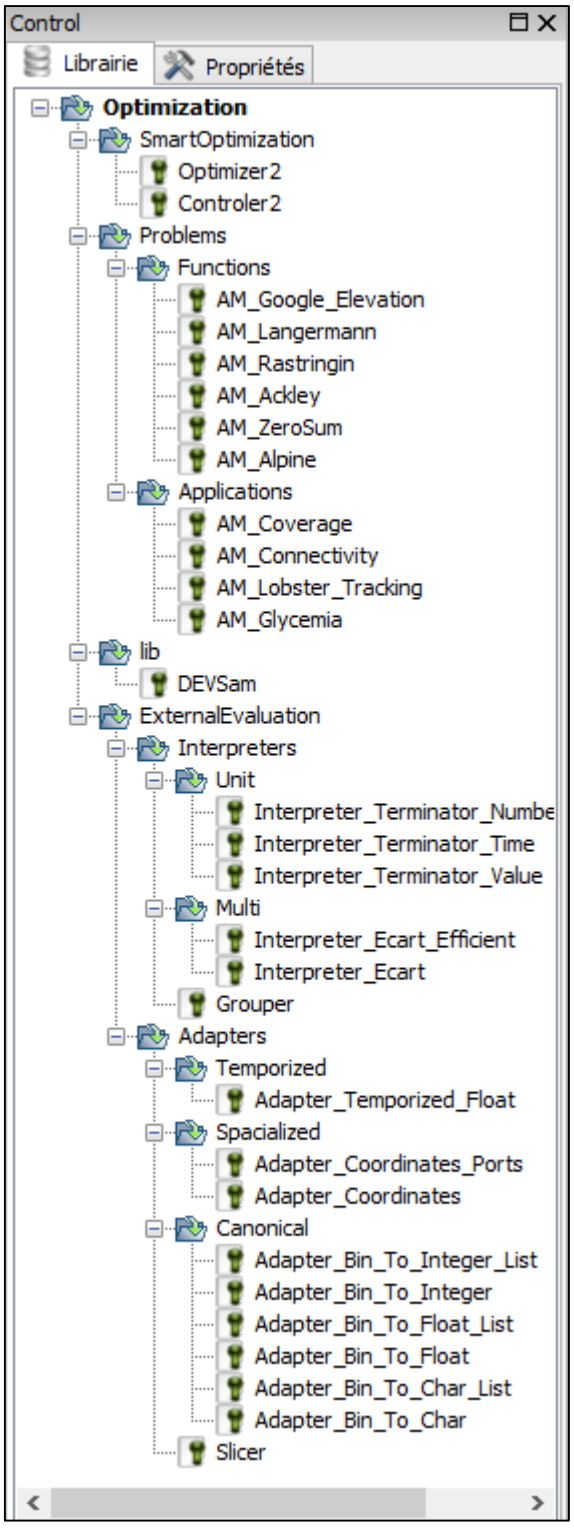


Figure 82 Arborescence de la librairie « Optimization »

A partir de notre librairie de modèles DEVSimPy d'OvS, nous avons validé nos différents concepts sur trois problèmes distincts : l'optimisation paramétrique de fonctions, le déploiement d'un réseau de capteurs et enfin l'optimisation d'un traitement médical.

5.2 Optimisation de fonctions

5.2.1. Présentation du problème

Nous allons commencer par chercher à valider notre architecture sur différentes fonctions mathématiques traditionnellement utilisées comme « Benchmark ». De très nombreuses fonctions sont présentes dans la littérature. Ces fonctions possèdent différentes caractéristiques. Elles peuvent être convexes/non convexes, unidimensionnelles/multidimensionnelles, modales/multimodales. Pour couvrir ces différentes caractéristiques, nous avons choisi un nombre restreint de fonctions. Les fonctions implémentées sous la forme de modèles atomiques DEVS sont les suivantes :

- La fonction d'écarts de sommes
- La fonction d'Ackley
- La fonction de Rastringin
- La fonction de Langermann

5.2.2. Modélisation du problème

Notre outil exploite l'héritage de la Programmation Par Objet (POO), ce qui permet l'intégration rapide de l'ensemble de fonctions traditionnellement utilisé pour la comparaison de métaheuristiques(Molga and Smutnicki, 2005). Les fonctions ci-dessus ont donc été représentées sous la forme de modèles AMD DEVSimPy à partir des descriptions suivantes.

La fonction d'écart de sommes est une fonction convexe, multidimensionnelle et multimodale de nature régulière. Les différentes sorties de la fonction en dimension deux sont illustrées sur la Figure 83. Elle est définie par l'équation suivante :

$$fSommeZero(x) = \begin{cases} 0 \; si \sum_{i=1}^{n} xi = 0 \\ 1 + (10000 \left| \sum_{i=1}^{n} xi \right|)^{0.5} \end{cases}$$

Nous avons recherché par notre architecture de modèle à trouver la combinaison de variables nous permettant d'atteindre le minimum global cette fonction.

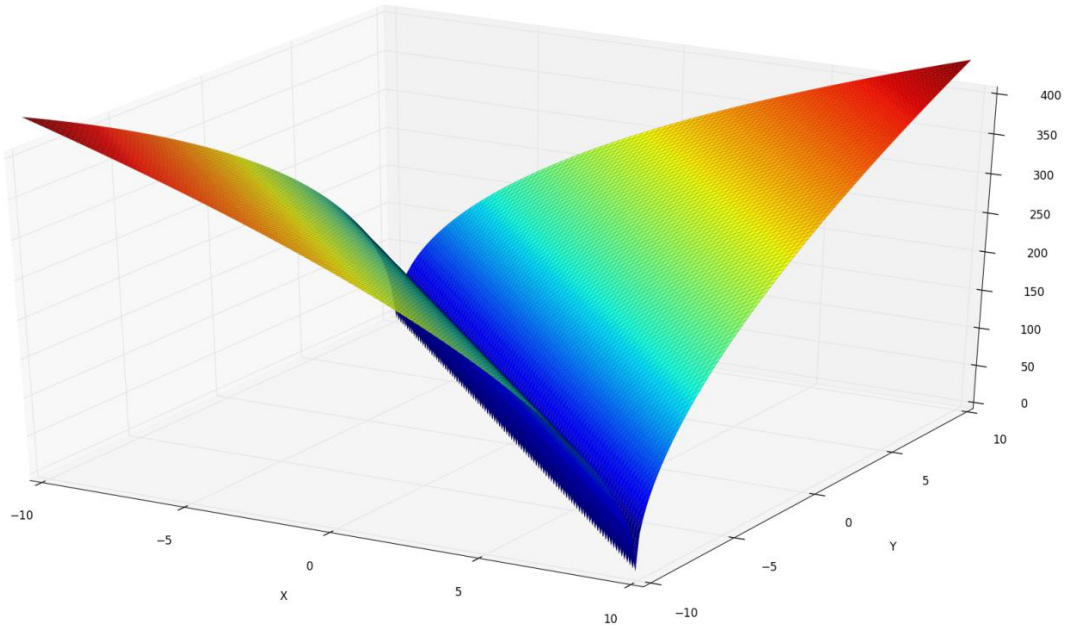


Figure 83 Fonction de sommes égales à 0 (2D)

Comme nous pouvons le voir sur la Figure 84, la fonction d'Ackley est une fonction non convexe, modale et non linéaire. C'est pourquoi, le processus de minimisation des variables du vecteur de celle-ci peut d'avérer extrêmement fastidieux. La fonction d'Ackley a été intégrée sous la forme d'un modèle AMD DEVSimPy décisionnel à partir de l'équation suivante :

$$f(x) = -a\,.\exp(-b\,.\sqrt{\frac{1}{n}\sum_{i=1}^{n} xi^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(c\,.xi)) + a + \exp(1)$$

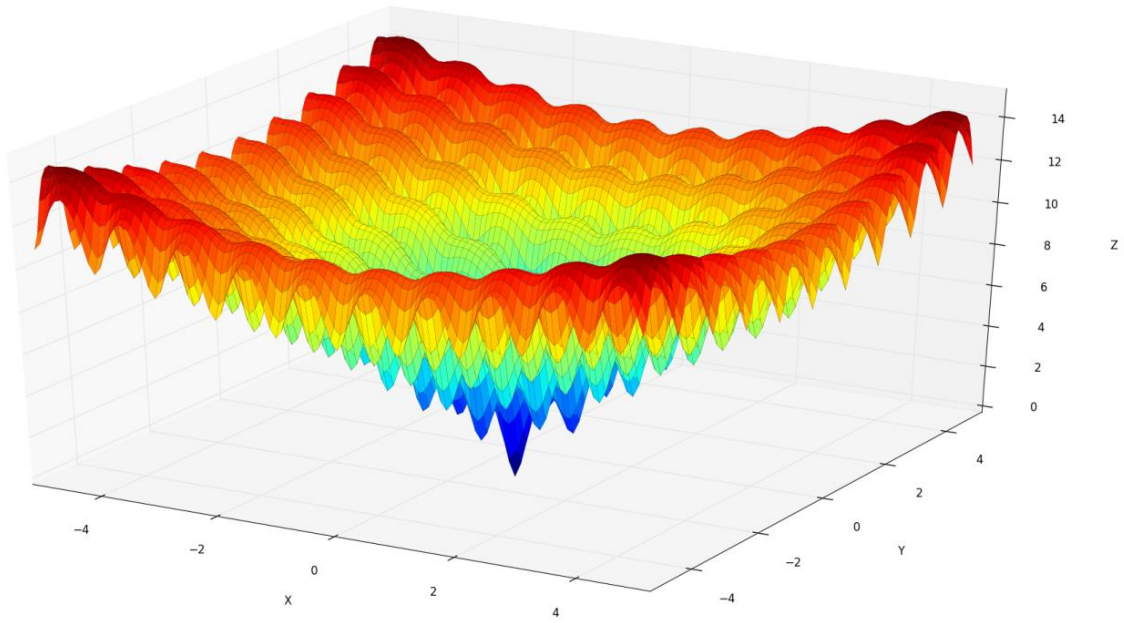


Figure 84 Fonction d'Ackley (2D)

Nous avons également modélisé la fonction Rastringin sous forme de modèle AMD DEVSimPy. La Figure 85 décrit cette fonction non convexe, multimodale, non linéaire et comportant nombreux minimums et maximums locaux. Le comportement du modèle qui lui est associé a été réalisé à partir de l'équation suivante.

$$f(x) = 10n + \sum_{i=1}^{n} [xi^2 - 10\cos(2\pi xi)]$$

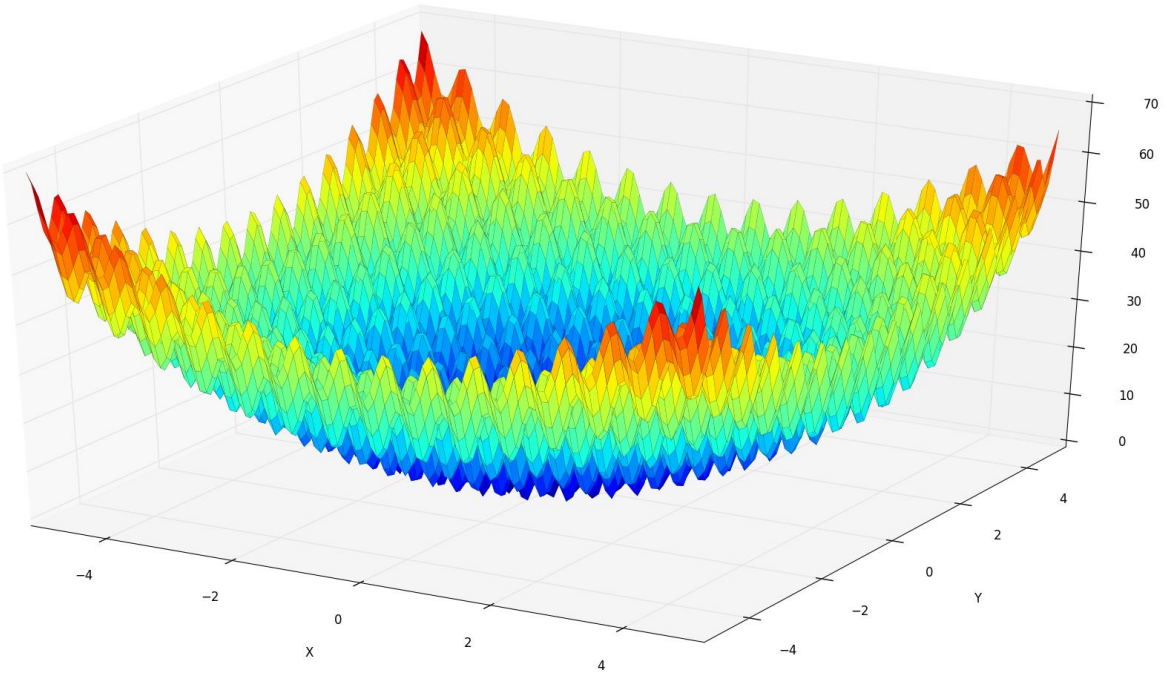


Figure 85 Fonction de Rastringin (2D)

Enfin, la fonction de Langemann, dont la Figure 86 donne un aperçu en dimension 2 a également été intégrée sous la forme d'un modèle AMD DEVSimPy. Comme nous le voyons, cette fonction est de nature non convexe, multimodale, et non linéaire. Elle se différencie des autres fonctions présentées précédemment par des minimums locaux très inégalement répartis. En ce sens, c'est une des fonctions pour lesquelles le risque de blocage du processus d'optimisation dans un minimum local est très important. Elle est définie par l'équation et la matrice suivante :

$$f(x) = \sum_{i=1}^{m} ci \exp\left[-\frac{1}{\pi}\sum_{j=1}^{n}(xj - Aij)^2\right] cos\left[\pi \sum_{j=1}^{n}(xj - ij)^2\right]$$

$$A = \begin{pmatrix} 3 & 5 \\ 5 & 2 \\ 2 & 1 \\ 1 & 4 \\ 7 & 9 \end{pmatrix}$$

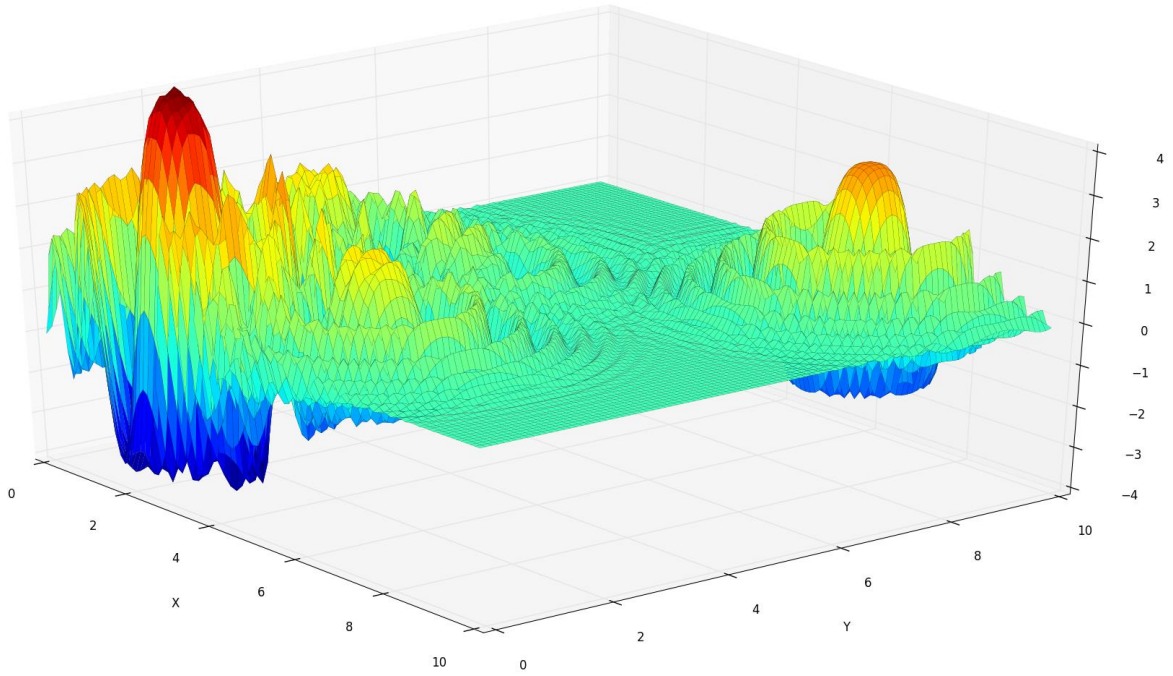


Figure 86 Fonction de Langermann (2D)

5.2.3. Résultats obtenus

Une fois les différents modèles de chaque fonction créés, le système d'optimisation via simulation a été graphiquement réalisé avec l'interface de DEVSimPY. Comme nous pouvons le voir sur la Figure 87, les modèles « d'optimisation intelligente » (couleur rouge) ont été interconnectés avec les modèles « d'évaluation externalisée » (couleur bleue) afin d'optimiser les paramètres de la fonction étudiée (couleur verte).

Grâce aux modèles d'adaptation, les solutions décrites sous forme de chaines binaires ont facilement pu être adaptées aux entrées des fonctions nécessitant des paramètres de type nombres réels. Aucun codage de l'utilisateur n'a été nécessaire pour l'obtention des résultats. Sur la Figure 87 nous pouvons observer que l'utilisateur a juste du choisir l'adaptateur adéquat et lancer la simulation avec l'interface graphique de DEVSimPy.

Pour cette application classique, trois métaheuristiques ont été utilisées : les algorithmes génétiques, la recherche harmonique et une métaheuristique utilisant ces deux algorithmes de manière alternative. Le paramétrage des algorithmes est basé sur les valeurs généralement présentes dans la littérature, à savoir : un taux de mutation de 5% pour les algorithmes génétiques avec un opérateur de croisement aléatoire et un taux de considération de la mémoire interne de 95% avec un ajustement de 5% et une improvisation de 5% pour la recherche harmonique.

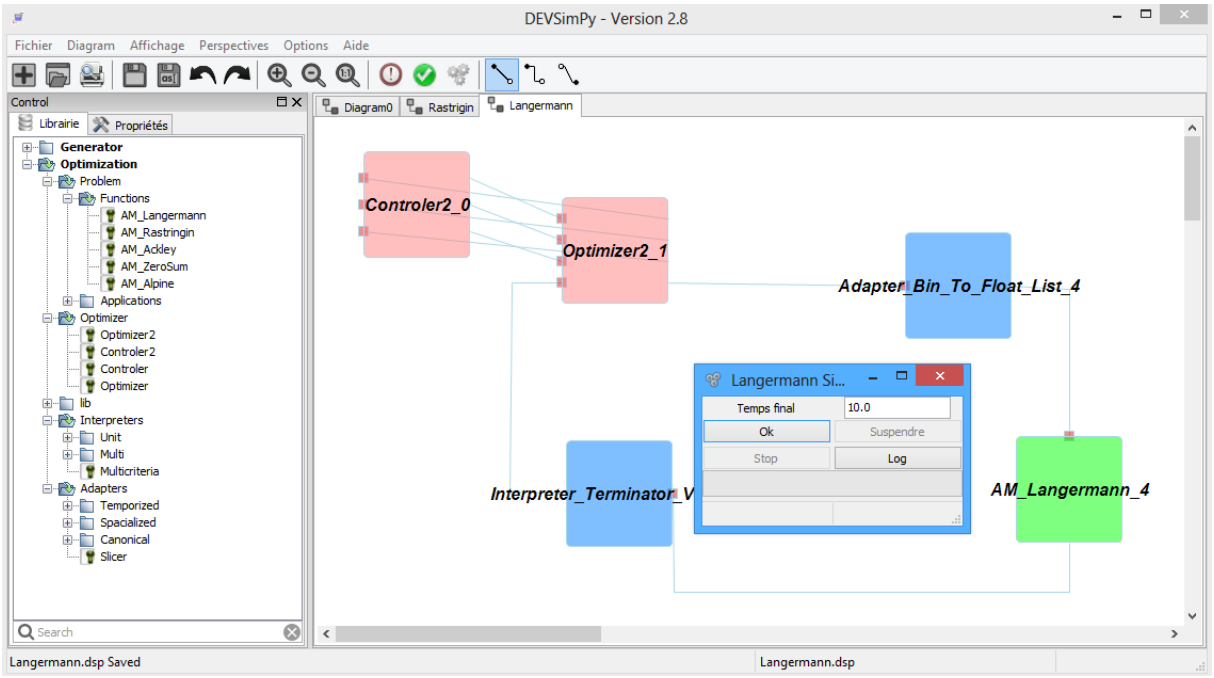


Figure 87 Optimisation de fonctions via simulation dans DEVSimPy

Comme nous pouvons le voir sur le Tableau 28, le Tableau 29, le Tableau 30, le Tableau 31, et la Figure 88 qui synthétisent les temps de calcul présents sur ces tableaux : la métaheuristique qui semble être la plus rapide dans le temps est l'algorithme génétique. Cependant nous pouvons observer que ce sont les techniques hybrides qui offrent une meilleure qualité de solution finale. Même si les temps de calcul sont légèrement supérieurs à ceux des algorithmes génétiques cette approche semble offrir une optimisation plus puissante. De plus nous remarquerons que les écarts de performance entre les algorithmes génétiques et la recherche harmonique sont très importants. L'utilisation de métaheuristiques hybrides permet alors de limiter considérablement le risque d'un choix non adapté de métaheuristiques.

Concernant ce même temps de calcul, celui-ci est majoré par les émissions de messages DEVS entres nos différents modèles. Néanmoins le concept de « simulation économe » permet de compenser largement cet inconvénient.

Dimension	Solution	Evaluation	Algorithme	Critère arrêt	Intervalle	Temps de calcul (secondes)
2	-0.9765625 0.9765625	0	AG	< 0.1	[-5,5]	0.172498285714
2	-0.1171875, 0.1171875	0	HS	< 0.1	[-5,5]	0.817800661654
2	-2.65625 2.65625	0	AG-HS	< 0.1	[-5,5]	0.146297744361
4	-4.6484375 3.4765625 4.0234375 -2.8515625	0	AG		[-5,5]	0.878479398496
4	-2.6171875 2.578125 4.4921875 -4.453125	0	HS		[-5,5]	0.0853625263158
4	-4.4140625 0.5859375	0	AG-HS		[-5,5]	1.18649215038

	2.3046875 1.5234375					
8	-1.6796875 0.546875 -2.734375 3.515625 -2.5390625 -0.234375 2.3828125 0.7421875	0	AG		[-5,5]	1.10938610526
8	-2.6171875 -1.9921875 3.125 4.296875 -4.765625 4.1015625 -0.0390625 -2.109375	0	HS		[-5,5]	2.28115584962
8	-2.8515625 4.4921875 4.609375 -0.4296875 -4.296875 -3.90625 1.171875 1.2109375	0	AG-HS		[-5,5]	1.28585864662

Tableau 28 Résultats proposés (fonction d'ecart de somme)

Dimension	Solution	Evaluation	Algorithme	Critère arrêt	Intervalle	Temps de calcul (secondes)
2	-.0390625 0.0	0.1506089185	AG	< 0.25	[-5,5]	0.138142796992
2	-0.0390625 -0.0390625	0.2358961584	HS	< 0.25	[-5,5]	0.645820150376
2	-0.0390625 -0.0390625	0.2358961584	AG-HS	< 0.25	[-5,5]	0.0881097142858
4	-0.0390625 0.1171875 -0.0390625 -0.0390625	0.4959001667	AG	< 0.5	[-5,5]	0.311453834586
4	-0.0390625 -0.078125 0.0390625 0.0	0.3087759060	HS	< 0.5	[-5,5]	3.35763825564
4	-0.078125 -0.0390625 -0.0390625 -0.078125	0.4394836967	AG-HS	< 0.5	[-5,5]	0.256819969925
8	-0.0390625 -0.0390625 -0.1171875 0.8984375 0.8984375 -0.8203125 -0.7421875 -0.0390625	2.9483267327	AG	< 3	[-5,5]	0.397594947369
8	-0.078125,	2.7627840600	HS	< 4	[-5,5]	10.2931094135

	0.15625 1.0546875 -0.0390625 0.0 0.4296875 0.8984375 0.039062					
8	-0.0390625 -0.0390625 -0.0390625 -0.0390625 -0.1171875 1.8359375 -0.0390625 -0.0390625	2.7394022805	AG-HS	< 4	[-5,5]	0.346854496241

Tableau 29 Résultats proposés (fonction d'Ackley)

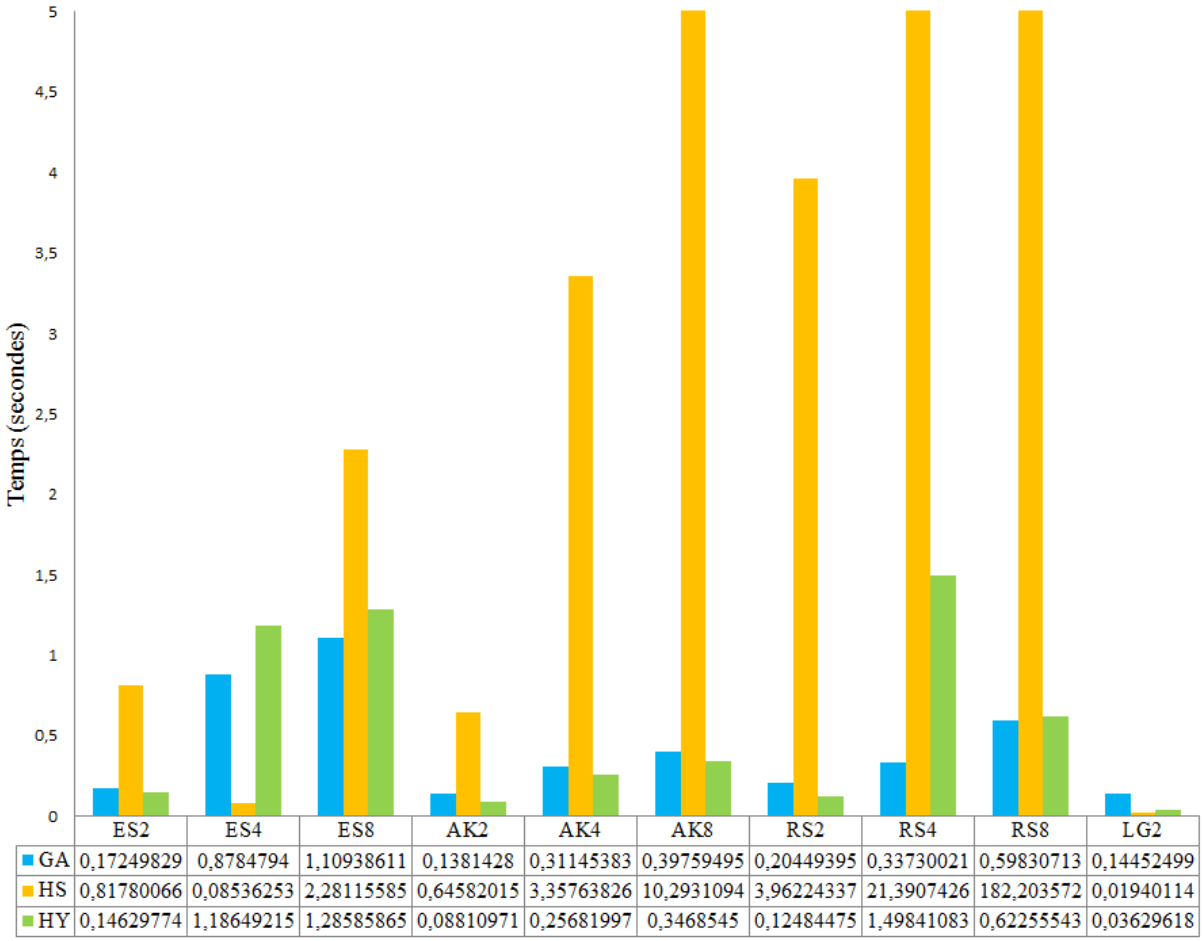
Dimension	Solution	Evaluation	Algorithme	Critère arrêt	Intervalle	Temps de calcul (secondes)
2	-0.02944946 -0.00015258	0.171576406653	AG	< 0.5	[-5,5]	0.204493954887
2	-0.0202941 0.0218200	0.1759089157	HS	< 0.5	[-5,5]	3.96224336842
2	-0.00381469 -0.00686645	0.0122392347	AG-HS	< 0.5	[-5,5]	0.12484475188
4	-0.00320434 -0.00015258 -0.0172424 -0.0392150	0.364525778431	AG	<2	[-5,5]	0.337300210526
4	-0.00366210 -0.01739501 0.04241943 -1.04904174	1.9890129966	HS	<2	[-5,5]	21.3907426165
4	-0.0003051 -0.0105285 -0.00015258 0.93734741	1.6655050359	AG-HS	<2	[-5,5]	1.49841082707
8	-0.93765258 0.93551635 -0.0050354 -1.0157775 -1.01608276 0.93643188 -0.0050354 -0.94284057	8.68690365461	AG	<10	[-5,5]	0.59830712782
8	-1.0485839 1.04904174 -1.0545349 -0.95108032 1.01715087 0.00823974 -1.0014343 0.00030517	8.30931170363	HS	<10	[-5,5]	182.20357197
8	-0.00015258 -0.00137329 -0.00030517 2.10815429	8.335994537956	AG-HS	<10	[-5,5]	0.622555428571

	-0.00015258 -0.00015258 -0.00991821 0.93704223					

Tableau 30 Résultats proposés (fonction de Rastringin)

Dimension	Solution	Evaluation	Algorithme	Critère arrêt	Intervalle	Temps de calcul (secondes)
2	1.09344482 0.62484741	-3.679505831	AG	< -3.5	[0,10]	0.144524992481
2	1.51229858 0.12390136	-3.632022889	HS	< -3.5	[0,10]	0.0194011428571
2	2.03536987 1.96212768	3.9749678795	AG-HS	< -3.5	[0,10]	0.036296180451

Tableau 31 Résultats proposés (fonction de Langermann)



	ES2	ES4	ES8	AK2	AK4	AK8	RS2	RS4	RS8	LG2
GA	0,17249829	0,8784794	1,10938611	0,1381428	0,31145383	0,39759495	0,20449395	0,33730021	0,59830713	0,14452499
HS	0,81780066	0,08536253	2,28115585	0,64582015	3,35763826	10,2931094	3,96224337	21,3907426	182,203572	0,01940114
HY	0,14629774	1,18649215	1,28585865	0,08810971	0,25681997	0,3468545	0,12484475	1,49841083	0,62255543	0,03629618

Figure 88 Résumé des résultats obtenus

Ces résultats ont permis de mettre en évidence la difficulté du choix d'une métaheuristique ainsi que son paramétrage pour une même application, justifiant l'intérêt d'une optimisation dynamique et auto-adaptable en fonction de la nature du problème. Pour valider cette proposition, nous avons choisi d'orienter nos tests vers une application destinée à l'optimisation de la couverture et de la connectivité de réseaux de capteurs sans-fils sur différentes zones.

5.3 Réseaux de capteurs sans fils

5.3.1. Présentation du problème

Les RCSF (réseaux de capteurs sans-fils RCSF) (Akyildiz et al., 2002; Yick et al., 2008) sont issus de deux domaines de recherche dont l'activité ne cesse de croitre : les réseaux informatiques, et la microélectronique. Ce succès s'explique en partie par les récentes avancées au niveau des MEMS (micro-electro-mechanical systems) et de l'IoT (Internet of Things).

Ces réseaux permettent la collecte décentralisée de données. Pour cela, ils se composent d'un ensemble de nœuds sur lesquels différentes sondes sont présentes. Ces sondes peuvent saisir diverses données telles que : la présence ou l'absence d'entités (personnes, animaux, substances), la température, le taux d'humidité, l'intensité lumineuse, la position GPS, le niveau de vibrations, etc.

Une fois ces données collectées, celles-ci convergent vers un « nœud puit » également appelé « Sink » et sont ensuite mises à disposition des utilisateurs depuis une passerelle généralement accessible depuis le web. Les données collectées par le réseau sont alors disponibles à partir de différents supports connectés (tablettes, téléphones portables ou ordinateurs personnels). Les différents concepts que nous venons d'énoncer sont résumés dans la Figure 89.

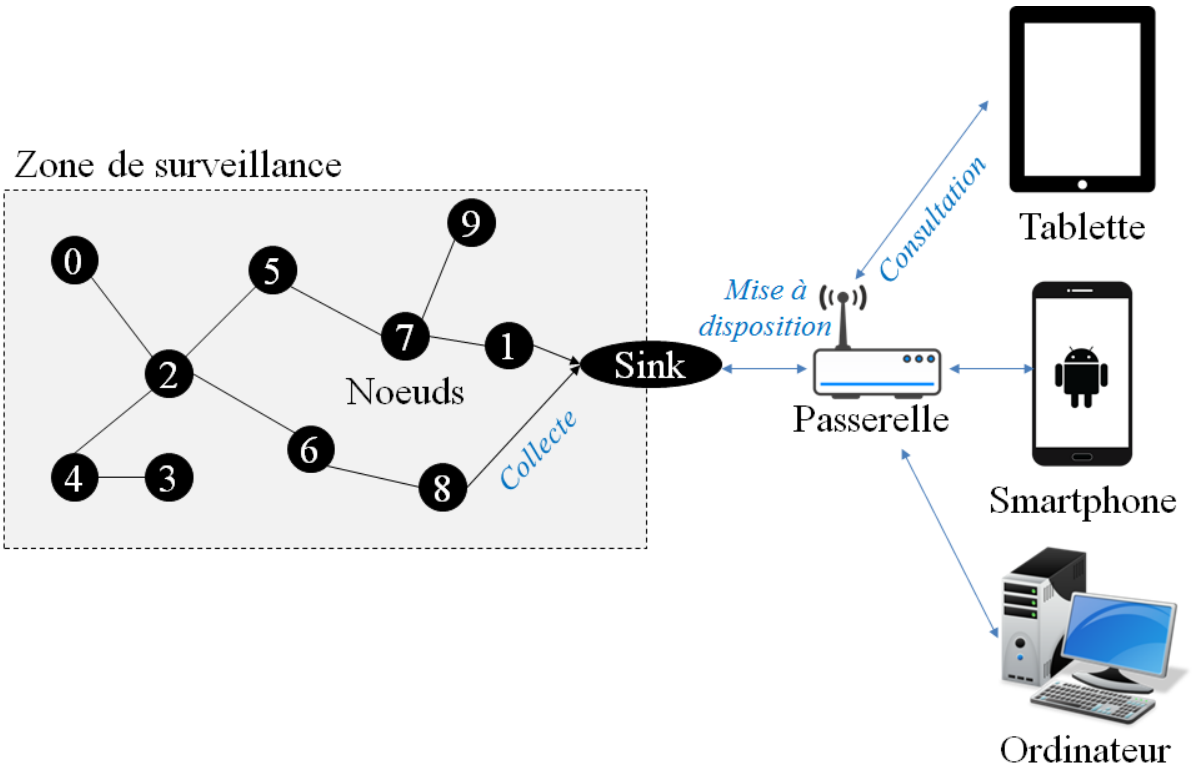


Figure 89 Exemple de réseau de capteurs sans fils (RCSF)

Les réseaux de capteurs sans-fils sont utilisés dans de nombreuses applications : le domaine militaire, notamment pour la détection et l'espionnage, le domaine médical pour le suivi des patients par les BAN (Body Area Networks). Enfin, ils suscitent un fort engouement dans les domaines de « l'agriculture intelligente » et de la prévention des risques notamment pour la détection de phénomènes naturels tels que : les éruptions volcaniques, les séismes, les tsunamis, les inondations, les avalanches, les tornades, les incendies ainsi que la sécheresse.

Cette technologie souffre cependant d'un manque de maturité, source de nombreuses problématiques (Puccinelli and Haenggi, 2005; Dargie, 2010, p. 9). En effet, les nœuds composant un réseau de capteurs sans-fils sont généralement alimentés par des batteries autonomes. Même si certaines technologies intègrent des systèmes d'auto-alimentation tels que des panneaux photovoltaïques, des micro-éoliennes ou encore des alimentations au fuel, la durée de vie d'un nœud reste limitée (Antoine-Santoni and Santucci, 2009). Le déploiement de ces réseaux est dans ce contexte essentiel pour l'économie d'énergie. De plus d'autres facteurs que l'alimentation peuvent conditionner un déploiement : les spécificités de l'application, les limites matérielles (portée de communication, zone de mesure, résistance) ainsi que l'aspect logiciel (protocoles, systèmes d'exploitation, sécurité).

Fort de ce constat et dans l'optique de fournir un outil d'aide à la décision pour le déploiement de réseaux de capteurs sans-fils, accessible depuis une interface graphique, nous avons modélisé sous forme de modèles DEVSimPy deux éléments centraux du réseau : la « couverture » et la « connectivité » entre les différents nœuds en fonction des spécificités de la zone de déploiement.

5.3.2. Modélisation du problème

5.3.2.1. Couverture du réseau

Comme énoncé précédemment, le premier objectif d'un RCSF est la collecte de données sur un support défini (zone, bâtiment, corps humain, etc.). De manière générale, la problématique qui en découle est de maximiser la surface couverte en utilisant un nombre minimum de nœuds.

Dans cette optique, nous avons modélisé la couverture d'un réseau de capteurs sans-fils à l'intérieur d'un modèle atomique DEVSimPy dont la représentation graphique est visible sur la Figure 90. Comme nous pouvons le voir sur cette figure, ce modèle dispose d'un seul port d'entrée sur lequel est transmise une liste de coordonnées. Ce modèle dispose également de cinq sorties pouvant être classées en deux groupes : «couverture » et « répartition ». Le groupe « couverture » produit quatre types de sorties : la surface couverte en m^2, la surface non couverte en m^2, le taux de couverture, le taux de non couverture. Le groupe « Répartition » produit une seule sortie représentant le niveau de dispersion des différents nœuds sur la zone surveillée.

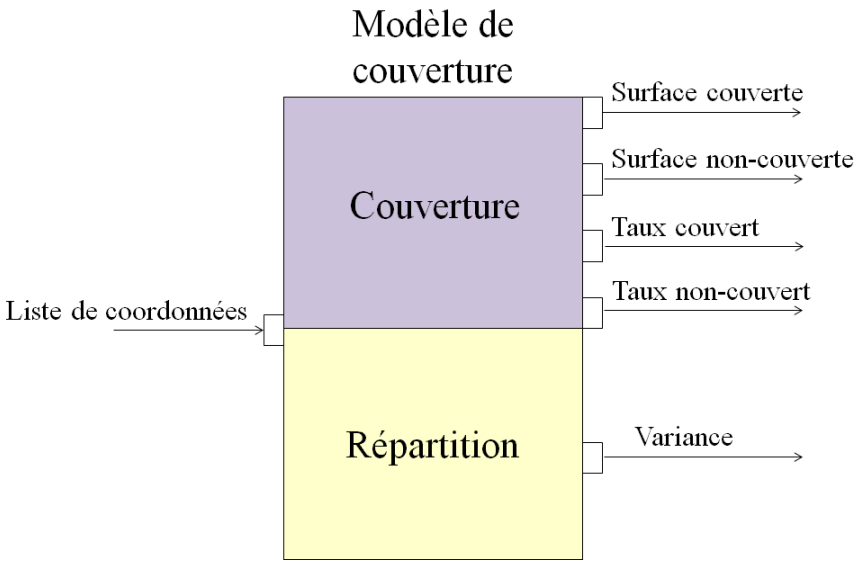


Figure 90 Modèle de couverture d'un RCSF

La configuration interne du modèle comporte deux paramètres essentiels : la description de la portée des sondes présentes sur les capteurs déployés et la définition des différentes sous-zones formant la zone de déploiement. Une sous-zone est définie par un polygone ainsi qu'un taux d'atténuation de la portée maximale de mesure du phénomène observé. Comme nous pouvons le voir sur la Figure 91, cinq types de zones peuvent par exemple être définis par l'utilisateur : « capacité maximale » (aucune atténuation), « capacité bonne » (atténuation de 20%), « capacité moyenne » (atténuation de 50%), « capacité faible » (atténuation de 80%) et enfin « capacité nulle » (atténuation totale). Les nœuds présents à l'intérieur de chaque zone verront leur zone de couverture diminuée du pourcentage correspondant.

Pour réaliser ces opérations géométriques, nous avons utilisé la librairie python « Shapley ». Elle permet la manipulation de points, de polygones et de lignes par différents prédicats et opérateurs. Les résultats générés peuvent être facilement convertis dans les principaux systèmes de projection et par conséquent être exportés sur différents systèmes d'information géographique (SIG).

La Figure 92 montre les différentes coordonnées représentant le positionnement des différents nœuds qui sont transmises au modèle sur son unique port d'entrée. Chaque coordonnée permet la création d'un cercle dont le diamètre est calculé en fonction du niveau de couverture de la zone dans laquelle il se trouve et de sa valeur maximale. Une fois les différents cercles créés, ceux-ci sont fusionnés par l'opérateur d'union et les débordements supprimés par l'opérateur de différence. L'objet obtenu est alors un polygone ou un ensemble de polygones formant la « zone couverte ». La différence entre cette « zone couverte » et la « zone de surveillance » est la « zone non couverte ».Le comportant permettant la génération de ces différentes sorties se base sur les équations présentées dans le Tableau 32.

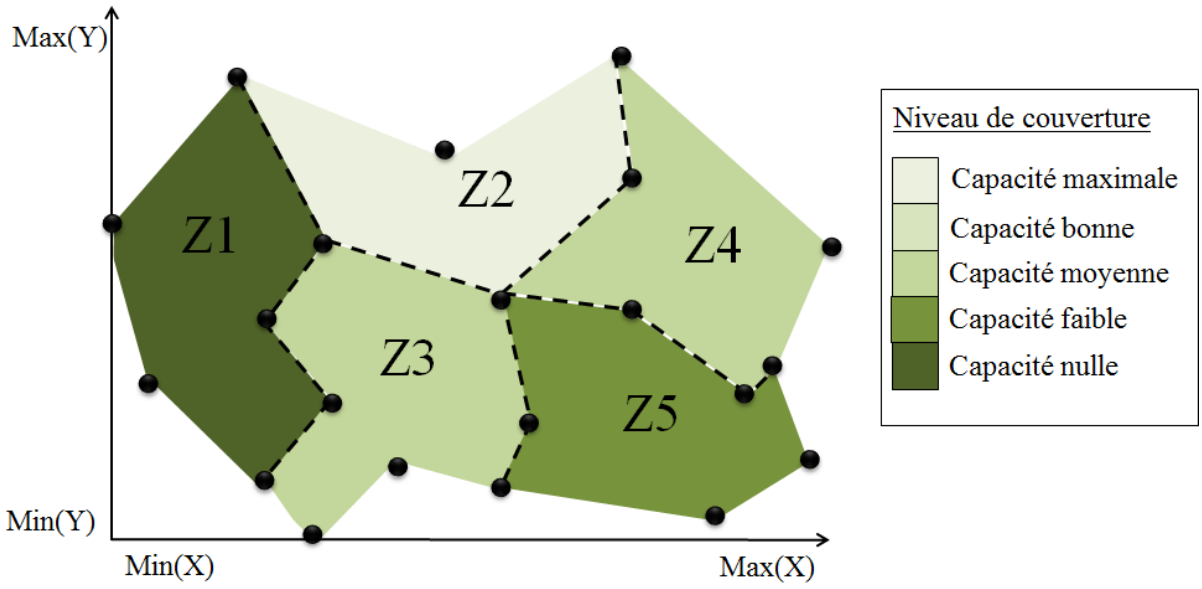


Figure 91 Définition des sous-zones et de leur atténuations

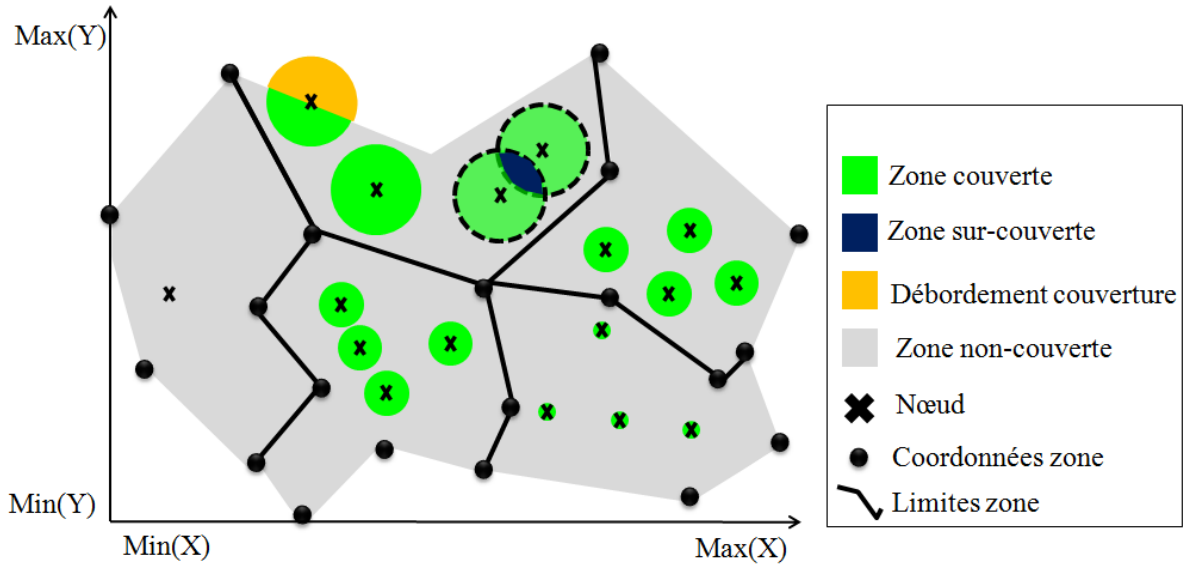


Figure 92 Atténuation du signal

Surface couverte	$\mathbf{SC} \leftarrow \bigcup_{i=1}^{n} \mathbf{f(Si)} \times \mathbf{g(Si)} \cap \mathbf{ZD}$ SC : Surface couverte f(Si) : Portée du capteur i g(si) : Taux atténuation de la portée du capteur i ZD : Zone de déploiement n : Nombre de capteurs
Taux de couverture	$TC \leftarrow SC \times 100 \div ST$ TC : Taux de couverture ST : Surface totale SC : Surface couverte
Surface non couverte	$SNC \leftarrow ST - SC$ SNC : Surface Non Couverte ST : Surface totale SC : Surface Couverte
Taux de non-couverture	$TNC \leftarrow SNC \times 100 \div ST$ TNC : Taux de non-couverture ST : Surface totale SNC : Surface non-couverte
Répartition	

	$\sum_{i=1}^{n}(\alpha - \beta)^2$ n : nombre de sous zones α : le nombre de nœuds sans la sous-zone i β : le nombre moyen de nœud par sous-zone i

Tableau 32 Sorties du modèle atomique DEVS « Couverture »

Couverture et connectivité du réseau ne doivent pas être confondues. En effet, un déploiement peut permettre la couverture d'une zone surveillée dans son intégralité par des sondes de mesure à large portée et cependant souffrir d'une connectivité médiocre entre les différents nœuds due à une faible portée de transmission ou à la présence de nombreuses interférences.

Partant de ce constat, nous avons alors développé un modèle distinct consacré uniquement à l'estimation de la connectivité d'un déploiement de réseaux de capteurs sans-fils.

5.3.2.2. Connectivité du réseau

Une fois les données collectées, le second objectif d'un réseau de capteurs sans-fils est d'acheminer ces données en direction du nœud puit. Les données seront alors sauvegardées et mises à disposition des utilisateurs à partir d'une passerelle. L'évaluation de la qualité et de la fiabilité d'un réseau se structure autour du concept de « connectivité ».

Pour évaluer la connectivité d'un déploiement de réseau, nous avons implémenté dans DEVSimPy le modèle décrit sur la Figure 93. Comme nous pouvons le voir, ce modèle dispose également d'une entrée sur laquelle la liste de coordonnées décrivant la topologie du réseau est reçue durant l'optimisation par simulation.

Son fonctionnement se base sur une distance de transmission spécifiée par l'utilisateur. Six valeurs décrivant le niveau de connectivité peuvent être générées en sortie lors de l'exécution du modèle. Ces six sorties peuvent être classées en deux groupes : « QoC » (Quality of Connectivity) et « QoN » (Quality of neighborhood). Les trois valeurs de « qualité de la connectivité » décrivent :

- le signal le plus faible entre deux nœuds du réseau
- le signal le plus fort entre deux nœuds du réseau
- le signal moyen de l'ensemble de nœuds du réseau

Les trois valeurs de « quantité de voisinage » décrivent :

- le nombre de nœuds voisins du nœud le plus isolé du réseau
- le nombre de nœuds voisins sur le nœud le plus connecté du réseau
- le nombre moyen de voisins sur l'ensemble des nœuds du réseau.

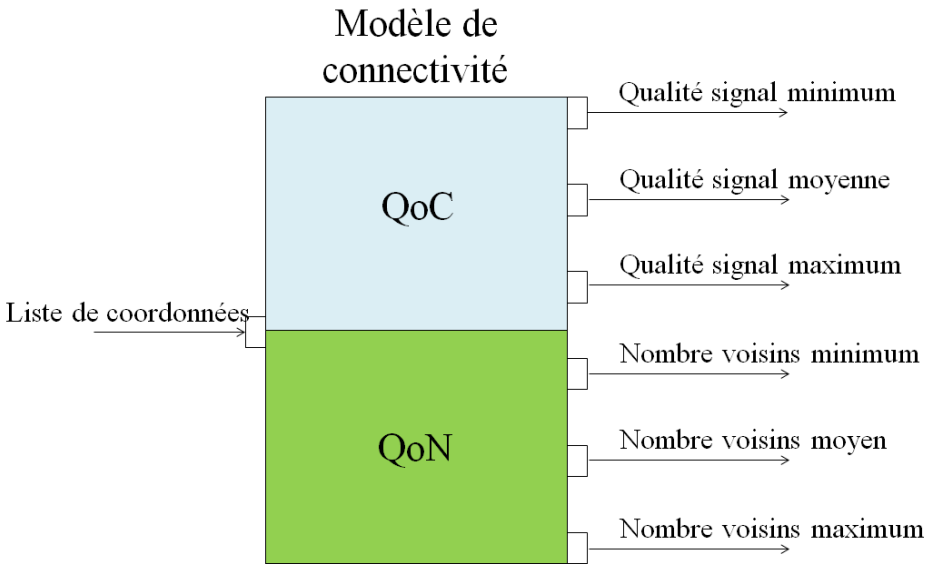


Figure 93 Modèle de connectivité d'un RCSF

Le calcul de ces différentes valeurs repose sur une analyse spatiale réalisée avec la librairie « Pysal » (Python Spatial Analysis Library) et son module « Spatial Weights ». Cette analyse permet de connaitre rapidement les différents nœuds voisins pour chaque nœud du réseau ainsi que la qualité du signal entre ces différents nœuds. Cette qualité de signal peut être estimée de différentes manières exprimées par plusieurs fonctions : triangulaire, uniforme, quadratique, quartique, gaussienne. Nous avons choisi d'utiliser la représentation gaussienne pour son adéquation avec l'atténuation d'un signal

hertzien. Les différents calculs effectués par le modèle suite à la réception d'une liste de coordonnées décrivant une proposition de topologie sont présentés dans le Tableau 33.

Qualité du signal minimum	$\min_{n}(\mathbf{S})$ S : vecteur des qualités de signal entre les différents nœuds n : nombre d'interconnections entre les différents nœuds
Qualité du signal moyenne	$\frac{\sum_{i=0}^{n} S_i}{n}$ S : vecteur des qualités de signal entre les différents nœuds n : nombre d'interconnections entre les différents nœuds
Qualité du signal maximum	$\max_{n}(S_i)$ S : vecteur des qualités de signal entre les différents nœuds n : nombre d'interconnections entre les différents nœuds
Nombre de voisins minimum	$\min_{n}(\mathrm{neighborhood}(N))$ N : vecteur des nombres de voisins pour les différents nœuds n : nombre de nœuds du réseau
Nombre de voisins moyen	$\frac{\sum_{i=0}^{n} \mathrm{neighborhood}(N_i)}{n}$ N : vecteur des nombres de voisins pour les différents nœuds n : nombre de nœuds du réseau
Nombre de voisins maximum	$\max_{n}(\mathrm{neighborhood}(N))$ N : vecteur des nombres de voisins pour les différents nœuds n : nombre de nœuds du réseau

Tableau 33 Sorties du modèle atomique DEVS « Connectivité »

5.3.3. Résultats obtenus

5.3.3.1. Optimisation du déploiement

Ces modèles ont été validés à partir d'un procédé d'optimisation dont la configuration est décrite sur la capture d'écran de la Figure 94. Deux modèles : « AM_Coverage_5 » et « AM_Connectivity_6 » ont été utilisés pour représenter la problématique de déploiement des réseaux de capteurs sans fils. Les deux sorties de ces modèles ont été interprétées par deux modèles d'interprétation distincts. Les résultats de ces deux interprétations sont regroupés par le modèle « Multicreteria_7 » afin de générer et transmettre une évaluation unique au modèle « Optimizer2_1 ». Ce regroupement se présente sous la forme d'une moyenne pondérée avec les coefficients suivants : 0,3 pour le niveau de satisfaction de la connectivité et 0,7 pour le niveau de satisfaction de la couverture.

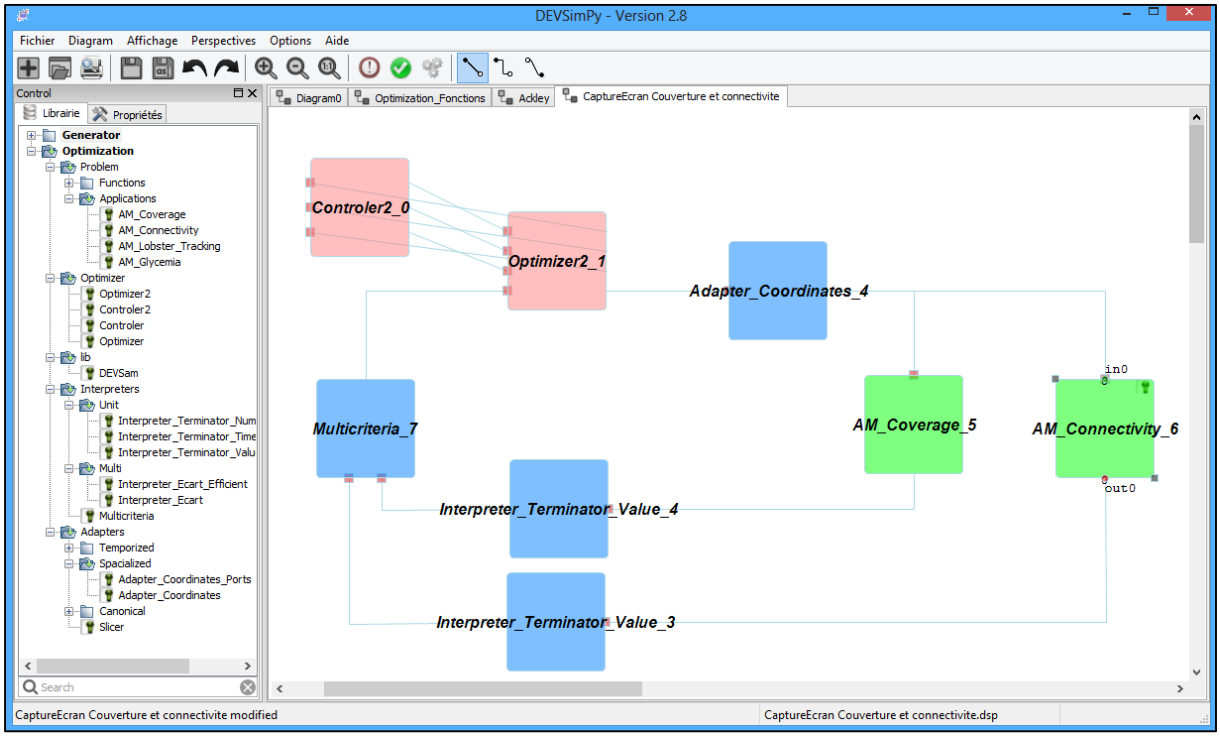


Figure 94 Optimisation de déploiement de RCSF via simulation dans DEVSimPy

Trois scénarios ont été configurés dans les modèles de couverture et de connectivité en utilisant l'interface graphique de DEVSimPy. Chacun de ces scénarios décrit une zone composée elle-même de trois sous-zones ayant différents niveaux d'atténuation: 20%, 80% et 99%.

La configuration de ces différentes zones est illustrée dans les deux captures des Figure 95 et Figure 96. Dans la configuration initiale de ces deux modèles, l'attribut « listSubAreas » contient la description des différentes sous-zones formant la zone étudiée sous la forme de listes de coordonnées. L'attribut « listSubAreasAttenuations » décrit pour chacune de ces sous-zones le taux d'atténuation estimé par l'utilisateur. Ce taux d'atténuation réduira la portée de transmission ainsi que la surface d'observation du phénomène des différents nœuds. On remarque sur ces deux captures d'écran que la portée de mesure « sensingRange » a une valeur de dix, alors que la portée de transmission « communicationRange » a une valeur de vingt.

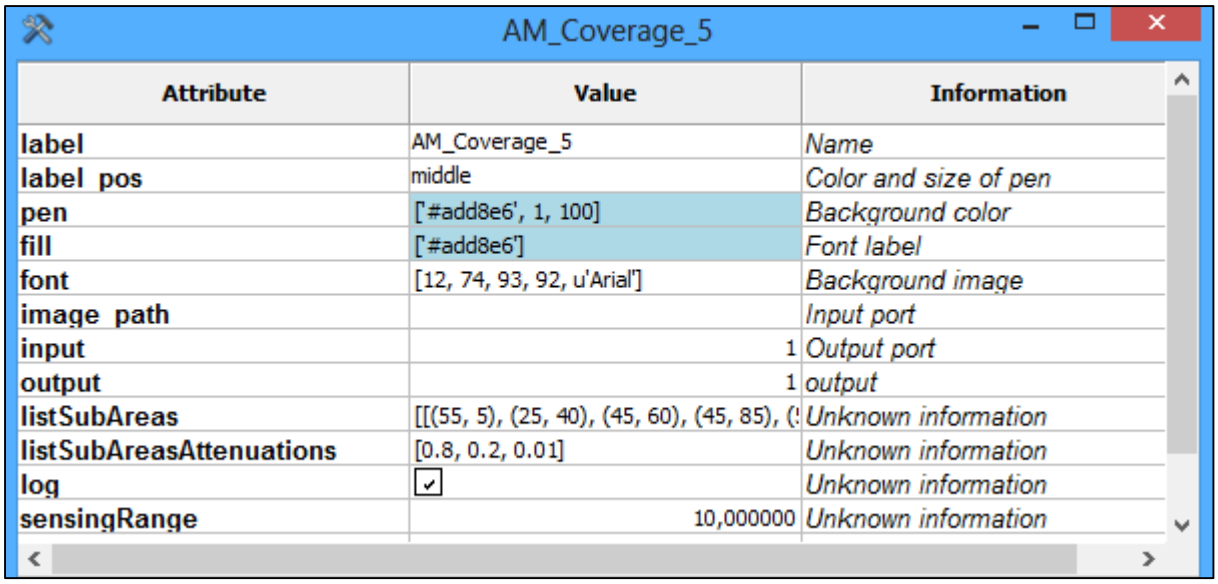


Figure 95 Configuration du modèle de couverture dans DEVSimPy

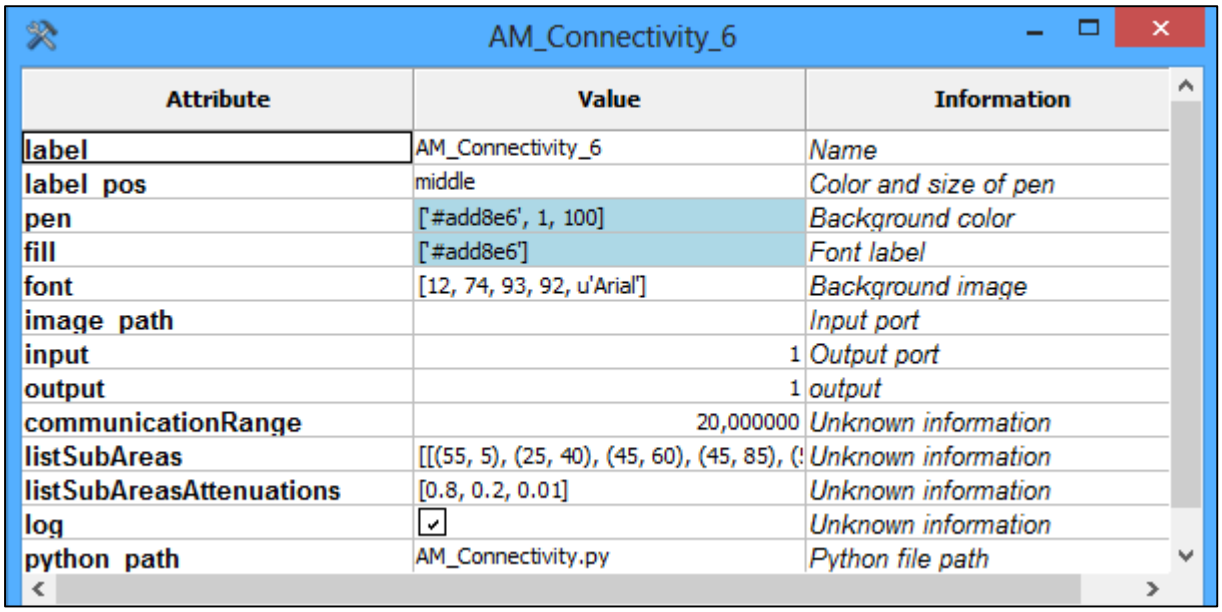


Figure 96 Configuration du modèle de connectivité

Cette configuration a été testée sur trois topologies de terrain distinctes. Pour chacune d'elles, quatre figures montrent les optimisations générées lors de la simulation. Les deux premières présentent les résultats aléatoires et les deux autres les résultats proposés par notre architecture de modèles.

Dans le premier scénario, le positionnement de vingt-cinq capteurs a été optimisé sur une zone composée de trois sous-zones de niveaux d'atténuation hétérogènes. La Figure 97 montre que le déploiement proposé par un raisonnement aléatoire offre une couverture d'environ 13.5 % alors que le déploiement proposé par une optimisation par simulation, visible sur la Figure 99, offre une couverture d'environ 25% soit une amélioration de plus de 92%. Au niveau de la connectivité, une amélioration est également visible entre le déploiement aléatoire du nœud sur la zone et le déploiement optimisé proposé. Sur la Figure 98 onze nœuds sur vingt-cinq ne sont pas connectés au réseau soit 44% des nœuds. Sur la Figure 100, ce nombre chute à sept après l'optimisation soit 28% des nœuds. De plus, le

nombre de nœuds ayant plus de trois voisins passe à douze, alors qu'il était de quatre pour la proposition aléatoire, soit une hausse de 200%.

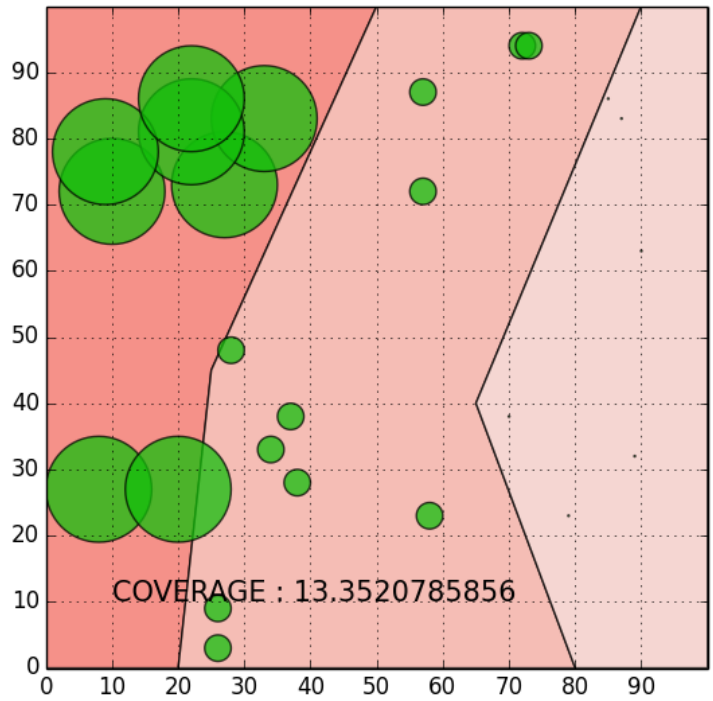


Figure 97 Déploiement aléatoire de 25 capteurs sur la zone n°1 (visulation de la couverture)

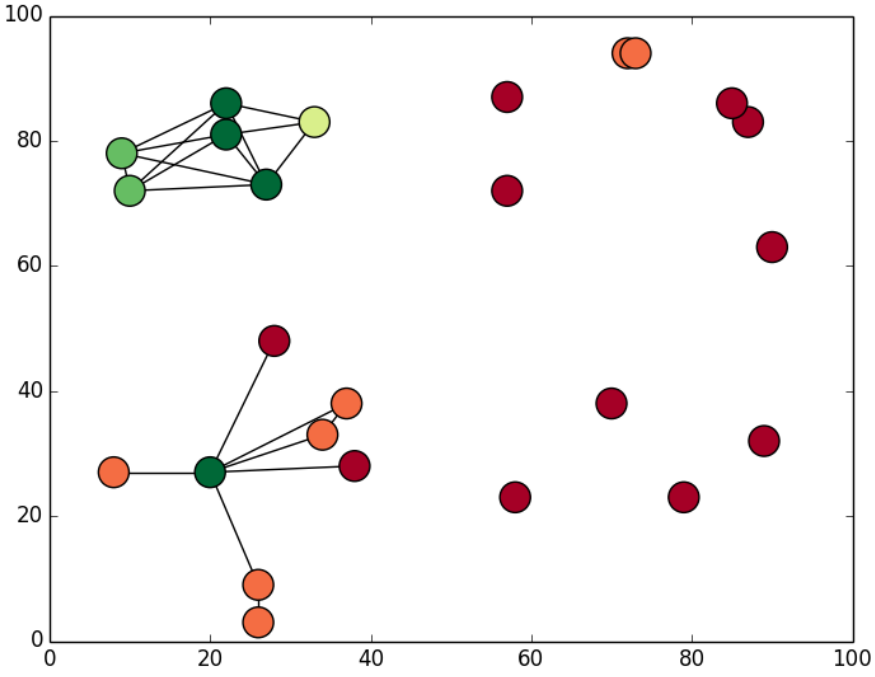


Figure 98 Déploiement aléatoire de 25 capteurs sur la zone n°1 (visulation de la connectivité)

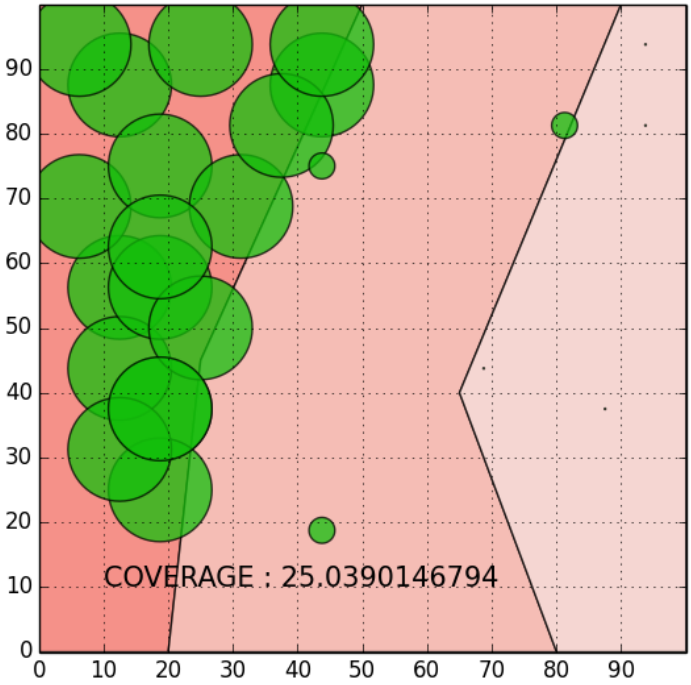


Figure 99 Déploiement optimisé de 25 capteurs sur la zone n°1 (visulation de la couverture)

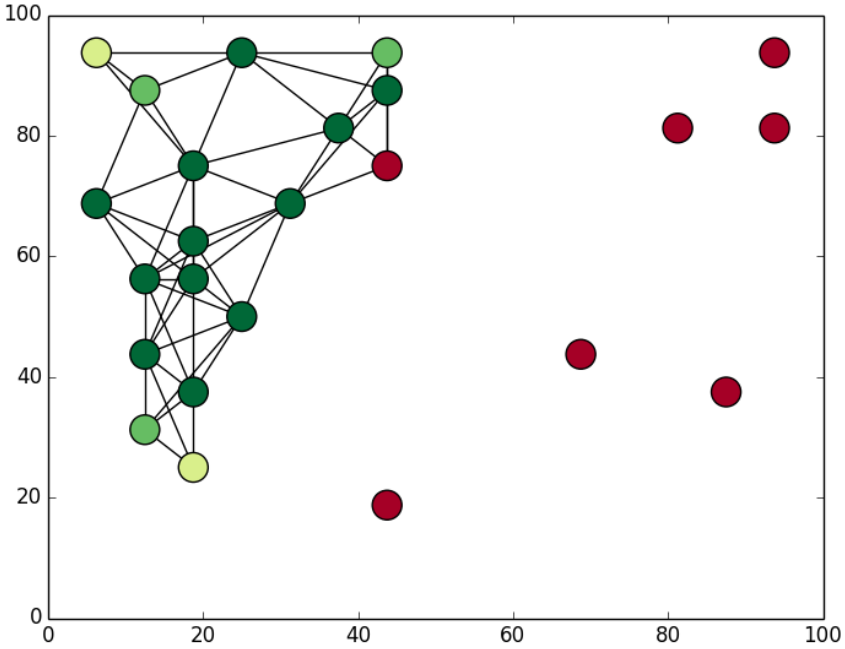


Figure 100 Déploiement optimisé de 25 capteurs sur la zone n°1 (visulation de la connectivité)

L'apport de nos modèles d'optimisation est également visible sur un deuxième scénario dans lequel la sous-zone de déploiement sur laquelle le niveau d'atténuation est le plus faible est de forme non régulière et de taille inférieure aux autres sous-zones. L'optimisation de la couverture pour ce scénario est décrite par la Figure 101 ainsi que la Figure 102. La connectivité est quant à elle décrite par la Figure 103 et la Figure 104. Sur cette zone, nous constatons que l'optimisation permet une amélioration de 315% de la couverture par rapport à un déploiement aléatoire. Le niveau de connectivité offert par le raisonnement aléatoire est quasi-nul. En effet, seul deux des vingt-cinq nœuds sont connectés, soit 8% des nœuds. Une fois optimisé, ce nombre s'élève à douze soit 48 % de nœuds connectés.

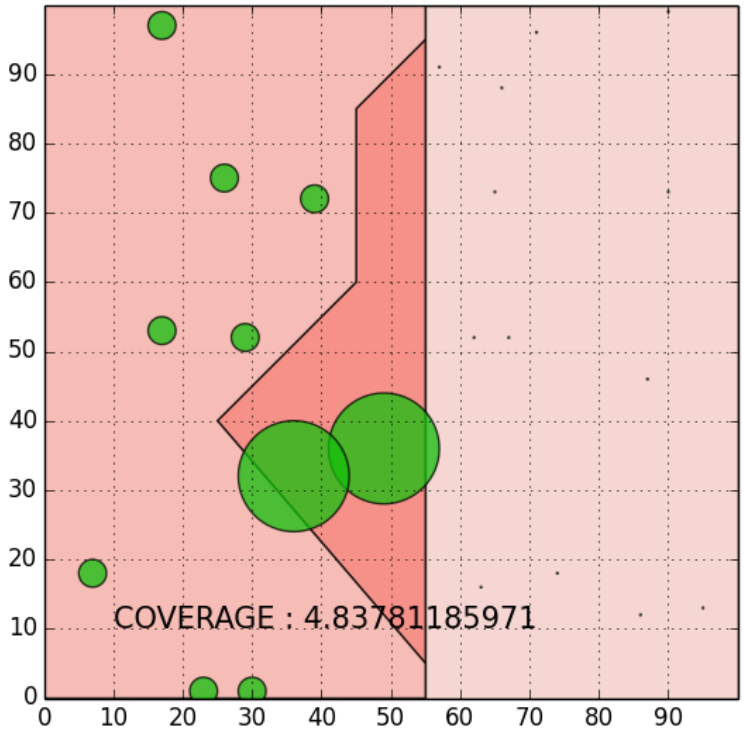


Figure 101 Déploiement aléatoire de 25 capteurs sur la zone n°2 (visulation de la couverture)

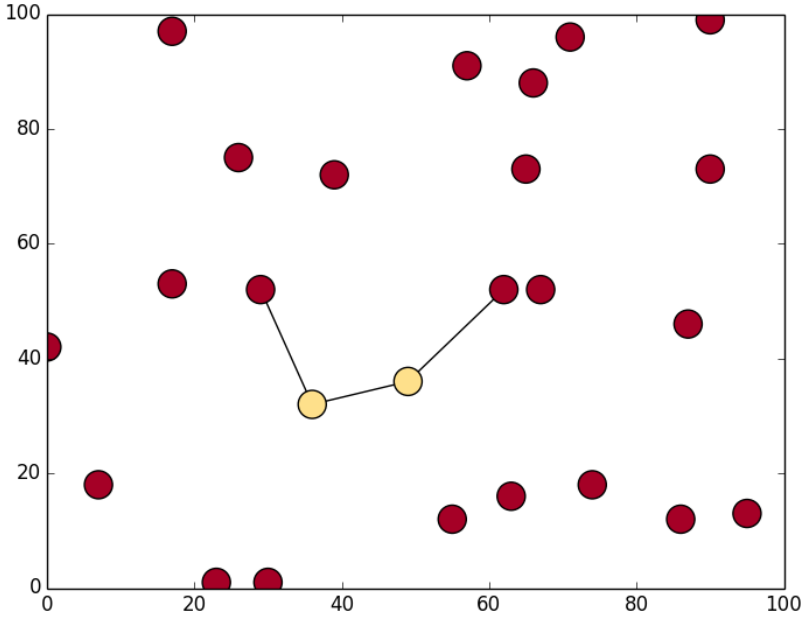


Figure 102 Déploiement alétoire de 25 capteurs sur la zone n°2 (visulation de la connectivité)

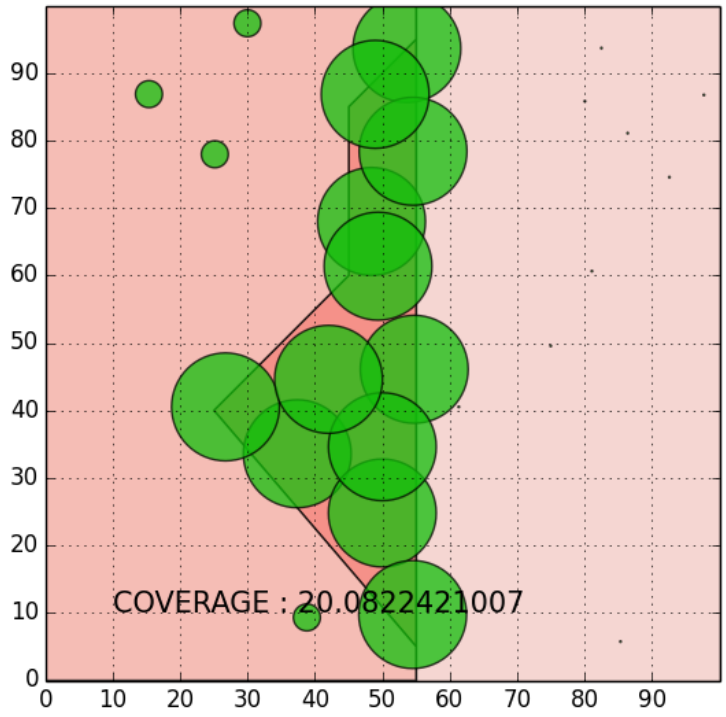


Figure 103 Déploiement optimisé de 25 capteurs sur la zone n°2 (visulation de la couverture)

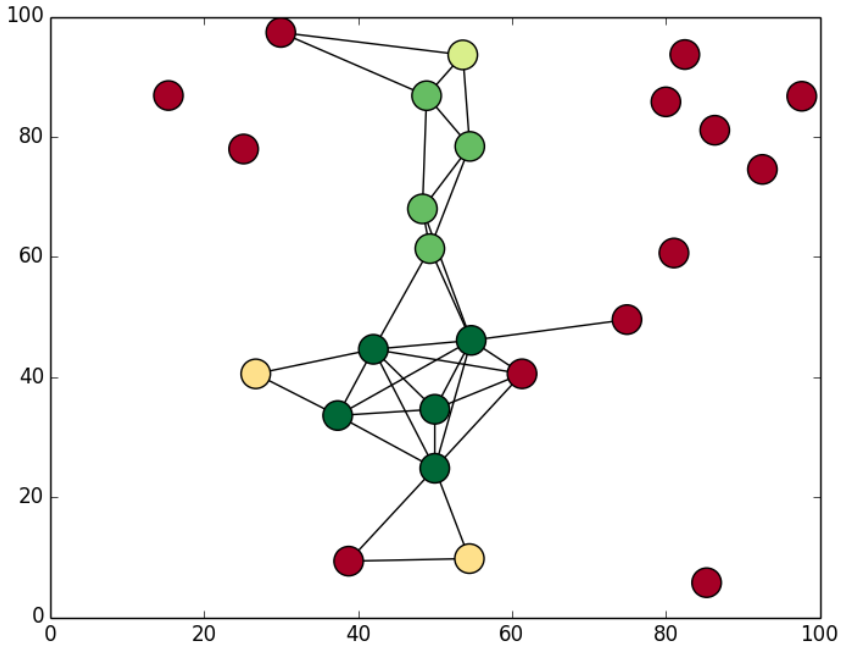


Figure 104 Déploiement optimisé de 25 capteurs sur la zone n°2 (visulation de la connectivité)

Enfin, nous avons également testé notre outil pour l'optimisation d'un déploiement sur une zone à très fortes irrégularités. La Figure 105, la Figure 106, la Figure 107 et Figure 108 confirment l'intérêt de l'optimisation pour l'élaboration de stratégies de déploiement. Une hausse de plus de 177% de la couverture est observable entre le déploiement aléatoire et le déploiement optimisé. Au niveau de la connectivité, nous obtenons un total de vingt nœuds interconnectés si nous utilisons notre outil d'optimisation. Si nous utilisons une stratégie aléatoire, ce nombre est de seulement dix nœuds qui sont connectés sous forme d'ilots, indépendants les uns des autres.

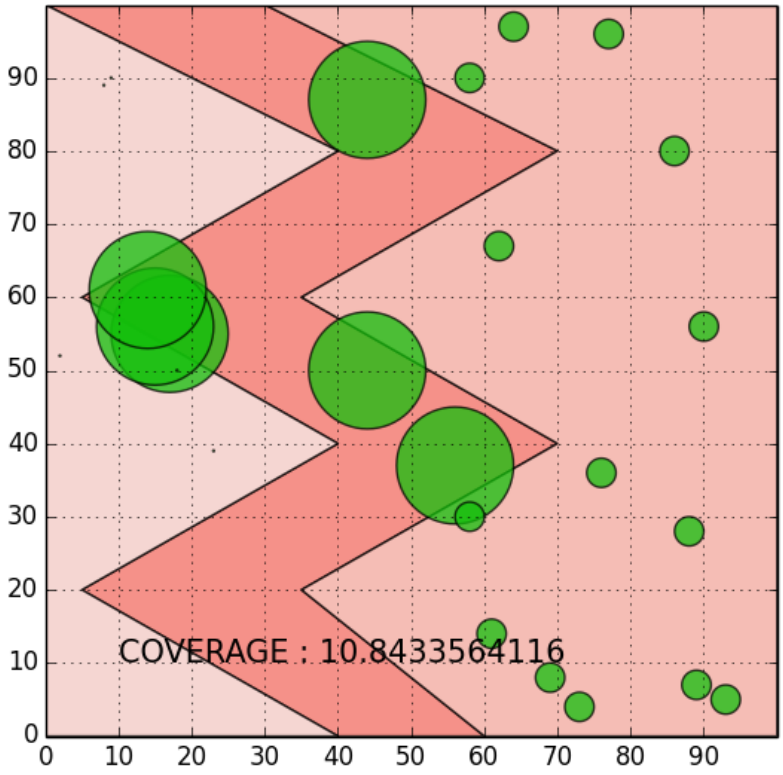


Figure 105 Déploiement aléatoire de 25 capteurs sur la zone n°3 (visulation de la couverture)

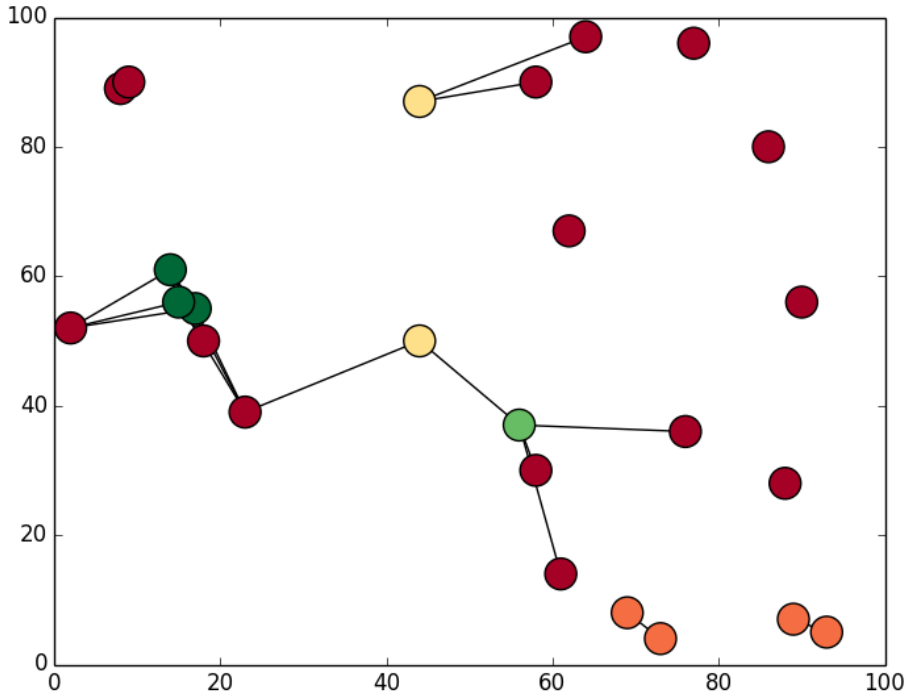


Figure 106 Déploiement alétoire de 25 capteurs sur la zone n°3 (visulation de la connectivité)

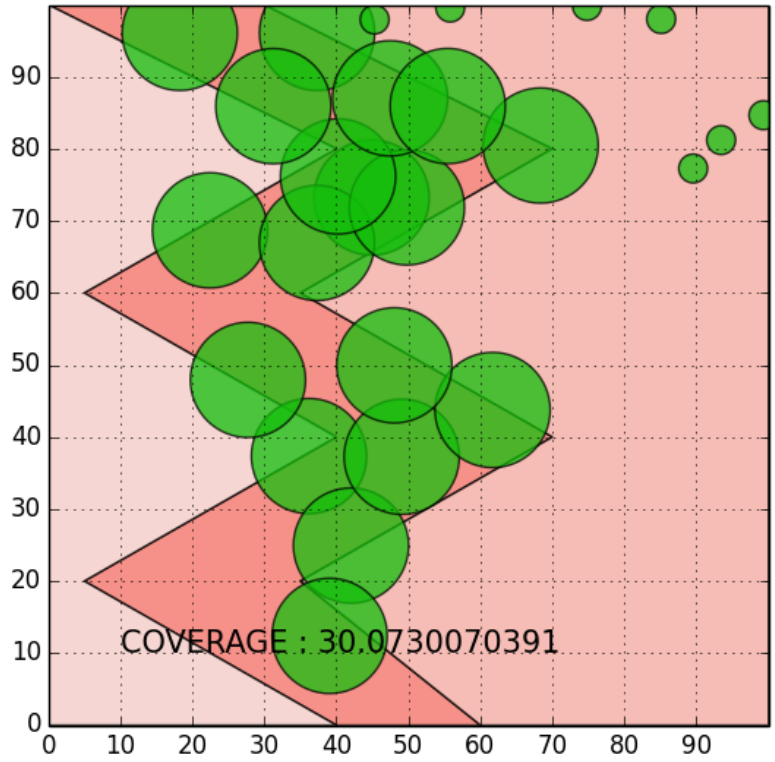


Figure 107 Déploiement optimisé de 25 capteurs sur la zone n°3 (visulation de la couverture)

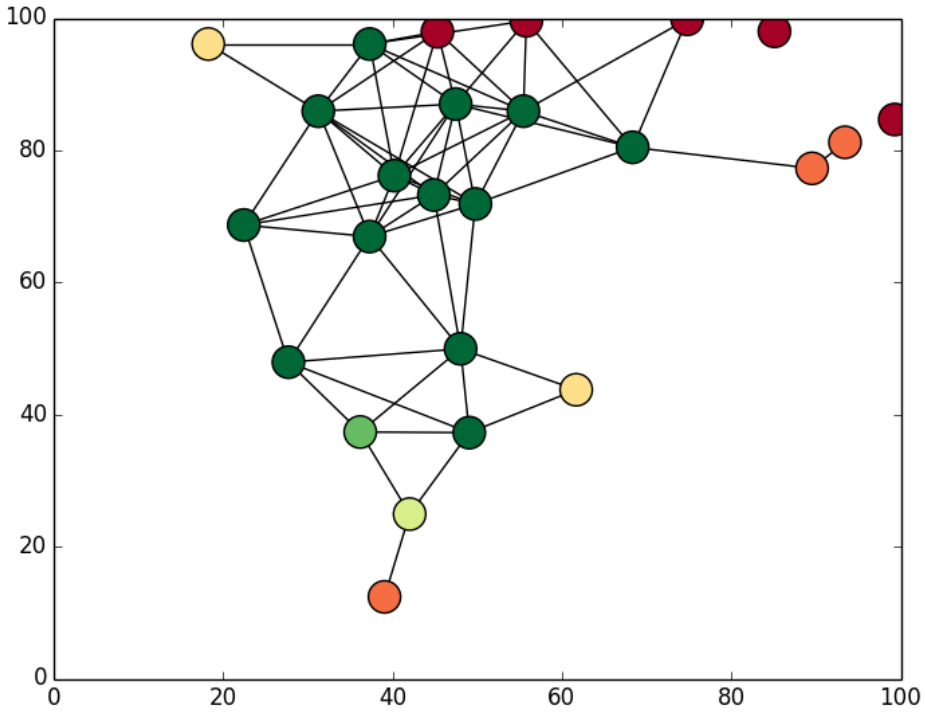


Figure 108 Déploiement optimisé de 25 capteurs sur la zone n°3 (visulation de la connectivité)

5.3.3.2. Optimisation intelligente

Cette application nous a permis d'étudier le concept « d'optimisation intelligente » qui permet la dynamisation du comportement du modèle « optimiseur » (Cf. 4.2.1.). Ce comportement est possible grâce à des changements ponctuels d'algorithmes, de paramètres et des solutions durant le processus d'optimisation par simulation. Le pilotage de ces changements est configuré une fois de plus grâce à l'interface graphique de DEVSimPy comme cela est visible sur la Figure 109.

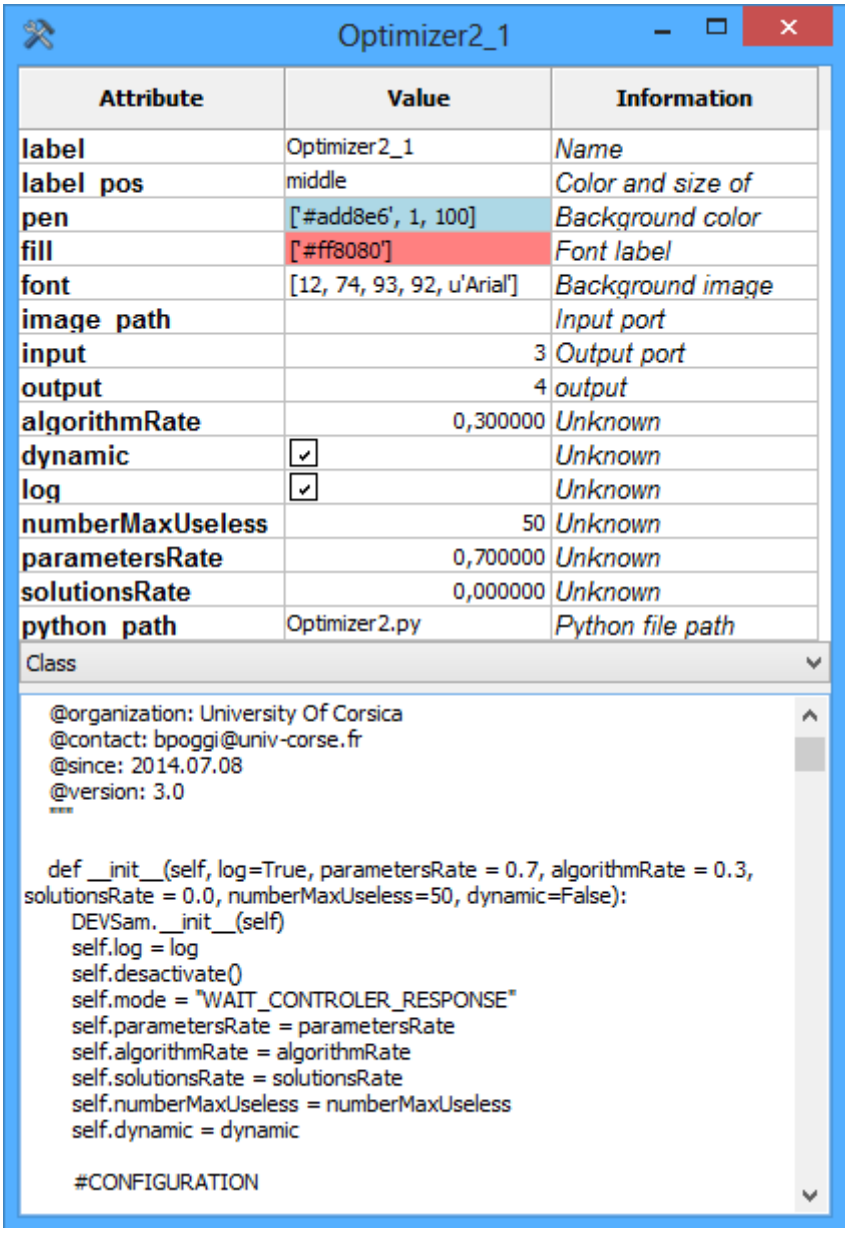


Figure 109 Configuration d'optimisation intelligente dans DEVSimPy

Sur cette figure nous pouvons remarquer que le dynamisme de l'optimisation se structure autour de quatre paramètres : « numberMaxUseless », « algorithmRate », « paramtersRate » et « solutionsRate ».

Le premier paramètre permet de définir le nombre maximum d'évaluations inutiles à partir duquel une demande de nouvelle configuration sera émise par le modèle « optimiseur » en direction du modèle « contrôleur ». Une évaluation est dite « inutile » si sa valeur est de moins bonne qualité que la

valeur des moins bonnes solutions présentes en mémoire. En ce sens, un nombre faible d'évaluations inutiles signifie une bonne progression du processus d'optimisation. À l'inverse un nombre important d'évaluations inutiles témoigne d'une difficulté de progression du processus d'optimisation.

Les trois paramètres suivants permettent de définir la priorité des choix entre : changement de solution, changement de paramètres et changement d'algorithme. Dans cette application, 30% des changements concerneront les algorithmes utilisés et 70% les paramètres utilisés. Aucun changement de solution ne sera effectué. Le choix de ce taux s'explique par des temps de simulations importants (de l'ordre de 0,65 seconde par solution) qui ne permettent pas le remplacement des solutions potentielles par de nouvelles solutions aléatoires.

Pour la zone 1, les nombres de changements durant la simulation sont présentés sur la Figure 110. Comme nous pouvons le voir, six changements de paramètres se sont produits et deux changement d'algorithme. Le détail de ces changements est présenté dans le Tableau 34. Ce tableau présente les valeurs des différents paramètres lors des changements. Trois paramètres sont possibles pour la recherche harmonique : « rd » pour le taux d'improvisation, « hmcr » pour le taux de considération des harmonies précédentes et « par » pour le taux d'ajustement des harmonies considérées. Deux paramètres sont possibles pour les algorithmes génétiques : « mutationRate » qui désigne le taux de mutation des nouveaux individus et « crossoverType » qui désigne un croisement uniforme lorsque sa valeur est égale à un et un croisement aléatoire lorsque sa valeur est égale à deux.

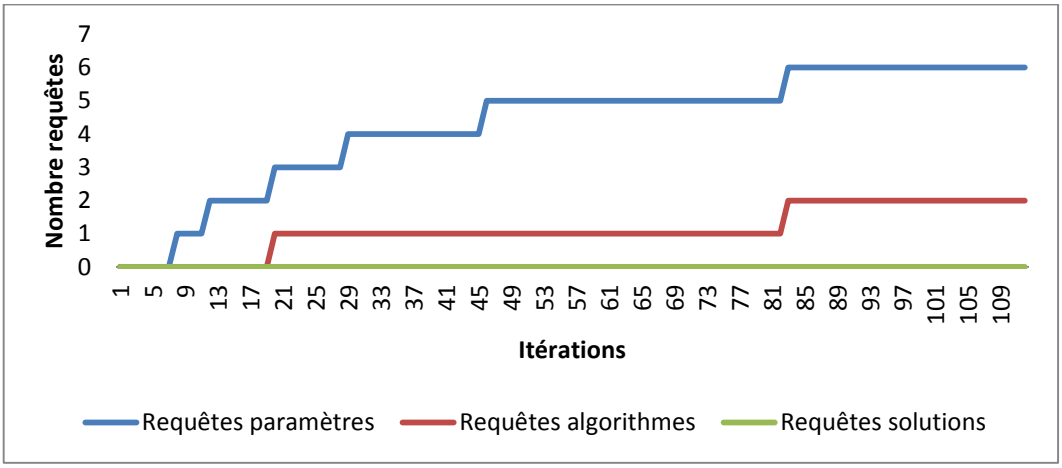


Figure 110 Dynamisme de l'optimisation pour la zone n°1

Itération	Algorithmes				
0	Configuration : hybride (AG + HS alternativement)				
0	rd	hmcr	par	mutationRate	crossoverType
0	0.007	0.619	0.159	0.061	1
7	0.142	0.794	0.299	0.186	2
11	0.282	0.501	0.207	0.246	2
19	Configuration Algorithmes génétiques				
19	mutationRate			crossoverType	
19	0.106			2	
28	0.065			2	
45	0.190			1	
82	Configuration : hybride (AG + HS alternativement)				
82	rd	hmcr	par	mutationRate	crossoverType
82	0.224	0.609	0.23	0.046	2

Tableau 34 Description du dynamisme pour la zone n°1

Pour la zone 2, les différents changements de comportement de « l'optimiser » sont présentés sur la Figure 111 et détaillés dans le Tableau 35. Comme nous pouvons le voir, quatre changements de paramètres se sont produits durant l'optimisation par simulation et trois changements d'algorithme.

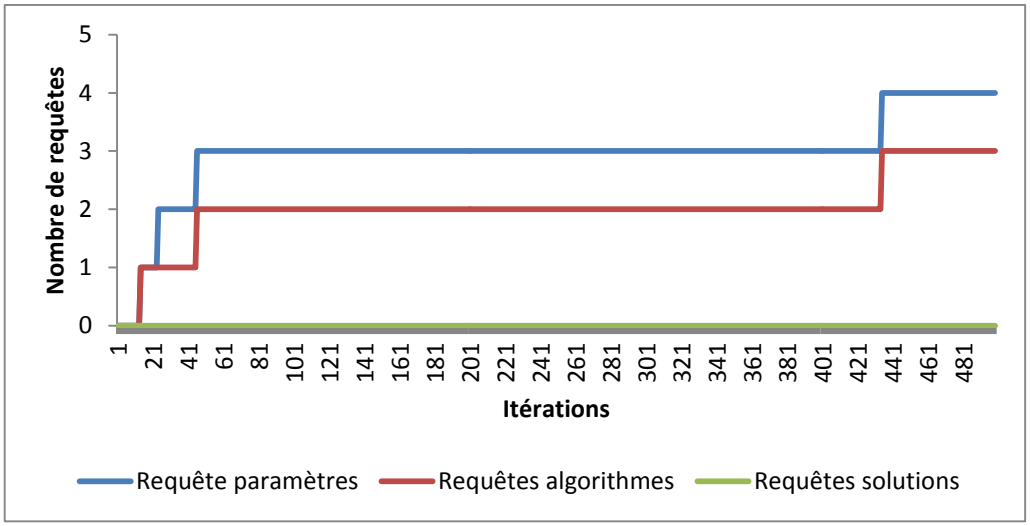


Figure 111 Zone 2 dynamisme de l'optimisation pour la zone n°2

Itération	Algorithmes				
0	Configuration : hybride (AG + HS alternativement)				
0	rd	hmcr	par	mutationRate	crossoverType
0	0.205	0.939	0.233	0.239	2
13	Configuration : hybride (AG + HS alternativement)				
13	rd	hmcr	par	mutationRate	crossoverType
13	0.160	0.962	0.336	0.336	2
23	0.000	0.847	0.363	0.003	1
45	Configuration : recherche harmonique				
45	rd		hmcr		par
45	0.126		0.907		0.093
434	Configuration : hybride (AG + HS alternativement)				
434	rd	hmcr	par	mutationRate	crossoverType
434	0.195	0.903	0.223	0.30	2

Tableau 35 Description du dynamisme pour la zone n°2

Enfin pour la zone 3, les différentes demandes de changements de l'optimiseur sont visibles sur la Figure 36. Les paramètres alors proposés par le « contrôleur » à « l'optimiseur » sont listés dans le Tableau 36. Nous pouvons observer sur ce scénario un seul changement d'algorithme et six changements de paramètres.

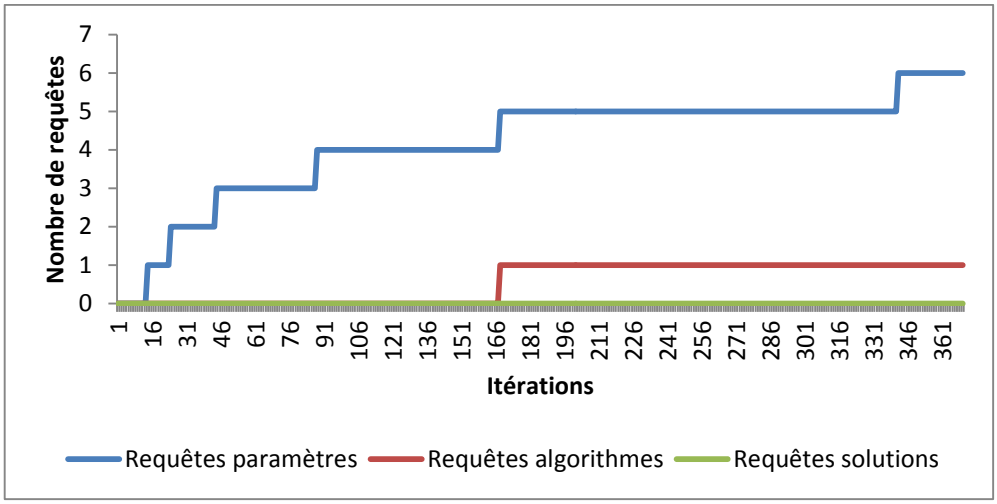


Figure 112 Zone 3 dynamisme de l'optimisation pour la zone n°3

Itération	Algorithmes				
0	Configuration : hybride (AG + HS alternativement)				
0	rd	hmcr	par	mutationRate	crossoverType
0	0.101	0.572	0.283	0.041	2
13	0.179	0.717	0.232	0.217	2
23	0.072	0.804	0.206	0.046	2
43	0.294	0.824	0.343	0.230	1
87	0.094	0.797	0.343	0.631	2
167	Configuration : harmonique search				
167	rd		hmcr		par
167	0.062		0.938		0.379
341	0.118		0.568		0.321

Tableau 36 Description du dynamisme pour la zone n°3

Une fois le déroulement de ces changements de configuration observés et validés, nous avons comparé les résultats obtenus grâce au dynamisme de « l'optimisation intelligente » par rapport à deux processus d'optimisations statiques basés sur l'utilisation des algorithmes génétiques et de la recherche harmonique.

Les paramètres fixés pour les algorithmes génétiques sont : un taux de mutation de 5% et un croisement aléatoire. Les paramètres choisis pour la recherche harmonique sont : un taux de considération de la mémoire harmonique de 95%, un ajustement de 10% et une improvisation de 5%. Le nombre de solutions potentielles est quant à lui de vingt pour les trois méthodes.

La Figure 113 représente l'évolution de la surface couverte en fonction du déploiement proposé par chacune des méthodes. Nous pouvons constater que ces trois méthodes ont un niveau de progression analogue durant les quinze premières secondes. Cependant, nous pouvons observer à partir de la quinzième seconde que la recherche harmonique semble offrir des résultats moins satisfaisants. Les résultats proposés par l'optimisation dynamique sont légèrement inférieurs à ceux proposés par les algorithmes génétiques mais cette tendance semble s'inverser à partir de la cinquante-huitième seconde. Pour confirmer ce constat, nous avons réalisé des optimisations par simulation sur des temps plus importants.

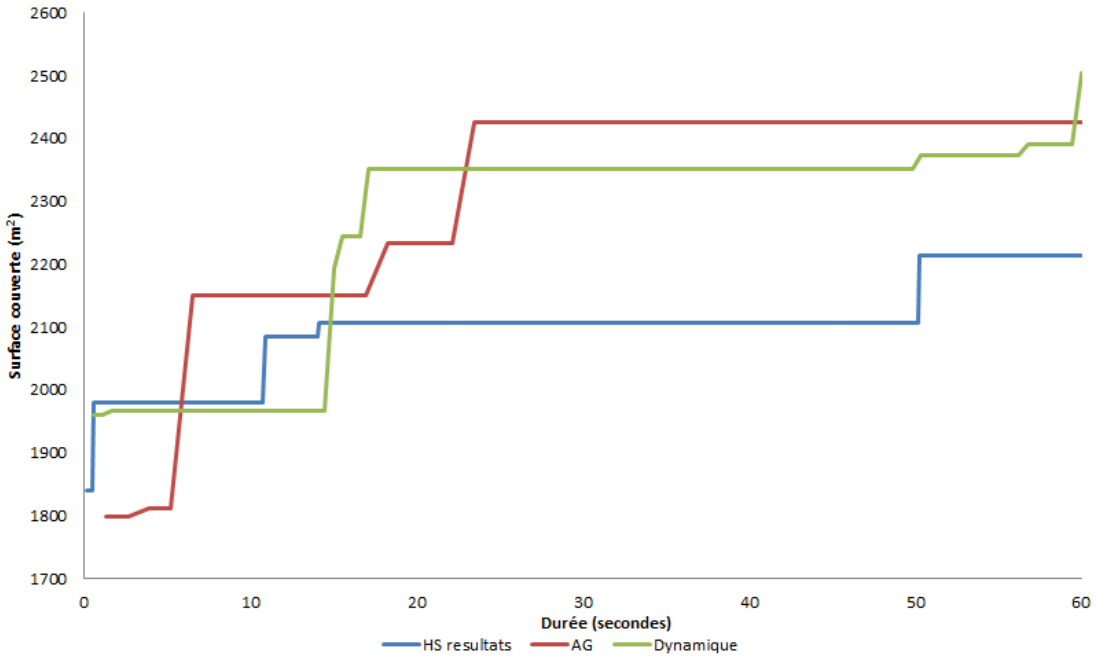


Figure 113 Evaluation de la meilleure solution potentielle pour les 3 méthodes sur une durée d'une minute (ZONE 1)

Les résultats visibles sur la Figure 114, présentent la progression de l'évaluation de la meilleure solution pour chacune des trois configurations sur la deuxième zone de déploiement optimisée. Comme nous pouvons le voir ici, les algorithmes génétiques offrent la meilleure performance lors des cinq premières secondes. Cependant, ceux-ci sont rapidement dépassés par la méthode d'optimisation dynamique que nous proposons. Même si celle-ci est elle-même rattrapée et dépassée aux environs de la trois-centième seconde, sa progression est beaucoup plus régulière et se rapproche beaucoup plus rapidement du niveau de couverture maximal (cent secondes contre trois cent pour la recherche harmonique).

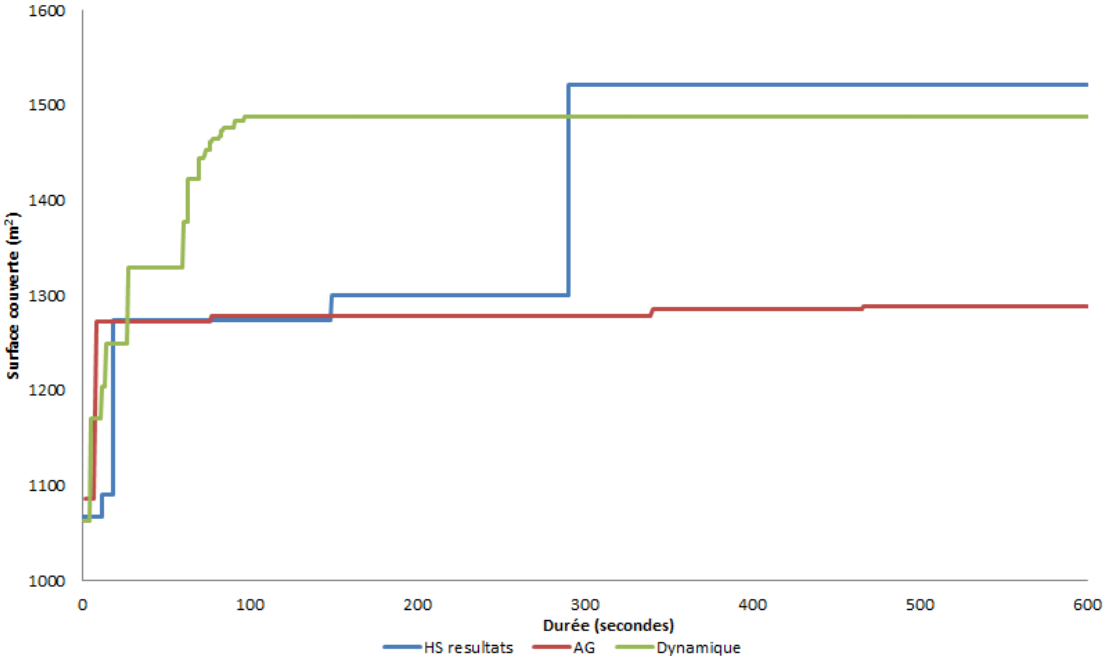


Figure 114 Evaluation de la meilleure solution potentielle pour les 3 méthodes sur une durée de dix minutes (ZONE 2)

Afin de confirmer ces résultats et d'approfondir notre analyse, une troisième optimisation par simulation a été réalisée sur la troisième zone de déploiement étudiée. Les résultats sont présentés sur deux figures. La Figure 115 présente l'évolution de l'évaluation de la meilleure solution pour chacune des trois méthodes d'optimisation sur une durée de deux minutes.

La Figure 116 représente les mêmes informations mais sur une durée d'exécution de vingt minutes. Nous pouvons observer un certain temps de latence de l'optimisation dynamique par rapport aux algorithmes génétiques et à la recherche harmonique. Cependant, ce retard est rapidement comblé à partir de la quatre-vingtième seconde. De plus la meilleure solution finale proposée par l'optimisation dynamique est d'une qualité supérieure de 14% à celle des algorithmes génétiques et de 25% par rapport à celle de la recherche harmonique. Nous observons cependant que le temps de simulation au-delà de la deuxième minute est inutile pour l'optimisation dynamique et ne permet pas l'obtention d'une solution de meilleure qualité, et ce malgré plusieurs variations de paramètres.

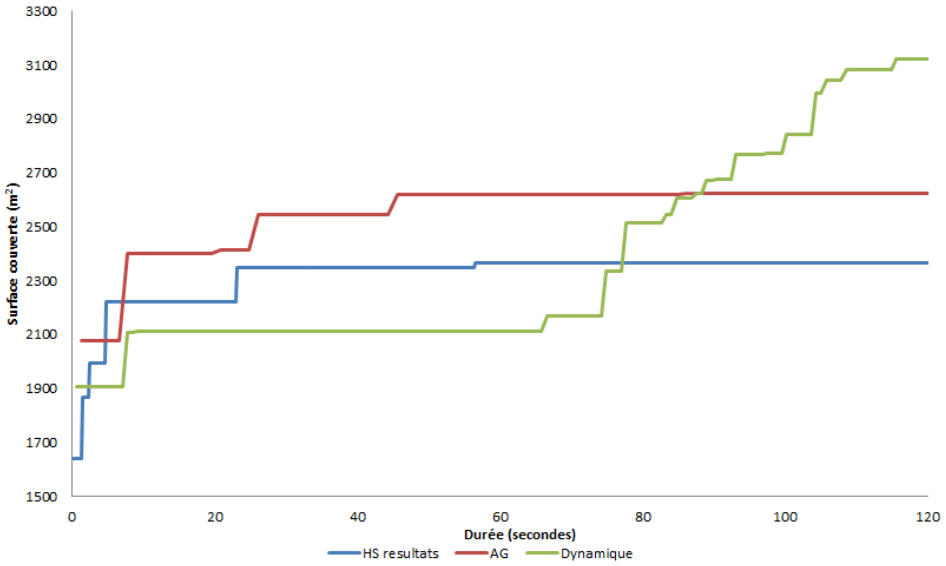


Figure 115 Evaluation de la meilleure solution potentielle pour les 3 méthodes sur une durée de deux minutes (ZONE 3)

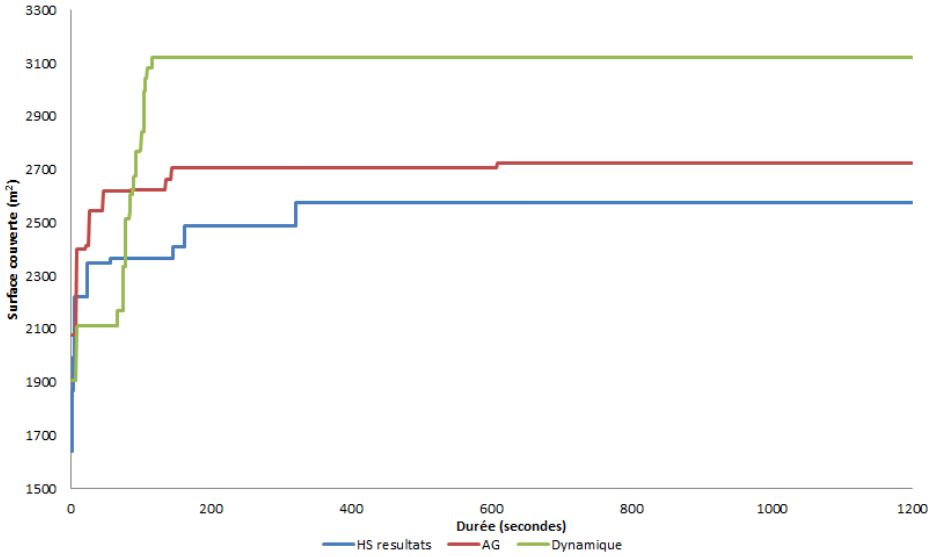


Figure 116 Evaluation de la meilleure solution potentielle pour les 3 méthodes sur une durée de vingt minutes (ZONE 3)

Sur ces trois exemples, nous avons pu observer d'importantes variations de performance entre les algorithmes génétiques et la recherche harmonique. En effet, les algorithmes génétiques offrent de meilleurs résultats que la recherche harmonique sur les zones 1 et 3 alors que la recherche harmonique offre de meilleurs résultats sur la zone 2. Ces variations sont beaucoup moins marquées lors de l'utilisation de « l'optimisation dynamique» que nous proposons. En effet, celle-ci se rapproche de la meilleure méthode d'optimisation dans le pire des cas et la surpasse dans les autres cas. Son utilisation évite donc à l'utilisateur d'effectuer un mauvais choix de méthode qui risquerait de limiter le potentiel d'optimisation des solutions. De plus, il semble d'après ces résultats que l'optimisation dynamique permet de maximiser les chances du processus d'optimisation de se rapprocher de la meilleure solution.

Afin de valider les autres concepts de notre architecture, à savoir : la « temporisation des variables » et la « simulation économe », notre architecture de modèles d'optimisation via simulation a été testée sur une troisième application ayant comme objectif l'optimisation de traitements médicaux.

5.4 Traitements médicaux

5.4.1. Présentation du problème

La médecine moderne utilise de plus en plus souvent des traitements dont le niveau de standardisation est important. Cependant, les limites de cette approche sont de plus en plus visibles. En effet, nous savons que deux patients présentant la même pathologie ne vont pas forcement réagir de la même manière à un même traitement. C'est pourquoi l'élaboration de traitement se dirige de plus en plus vers une médecine individualisée, adaptée aux spécificités de chaque individu.

Dans ce contexte, nous souhaitons utiliser notre outil d'aide à la décision pour l'élaboration de traitements médicaux individualisés en fonction des spécificités connues du patient. Dans l'approche que nous proposons vouée à être améliorée par des spécialistes du domaine, l'élaboration du traitement se structure autour de trois éléments : le choix d'une molécule, les heures de prises ainsi que les quantités de prises de médicaments.

5.4.2. Modélisation du problème

Pour répondre à ce besoins de traitements individualisés, nous avons conçu un modèle DEVSimPy « Taux sanguin » représentant le taux d'une substance quelconque dans le sang et sa réaction à un traitement spécifique. Le modèle dont les différentes entrées et sorties sont représentées sur la Figure 117 comporte un port d'entrée pour le traitement proposé et deux sorties représentant le taux sans traitement et le taux avec traitement. Celui-ci fonctionne sur des cycles de 24 heures représentant l'évolution des valeurs durant une journée type.

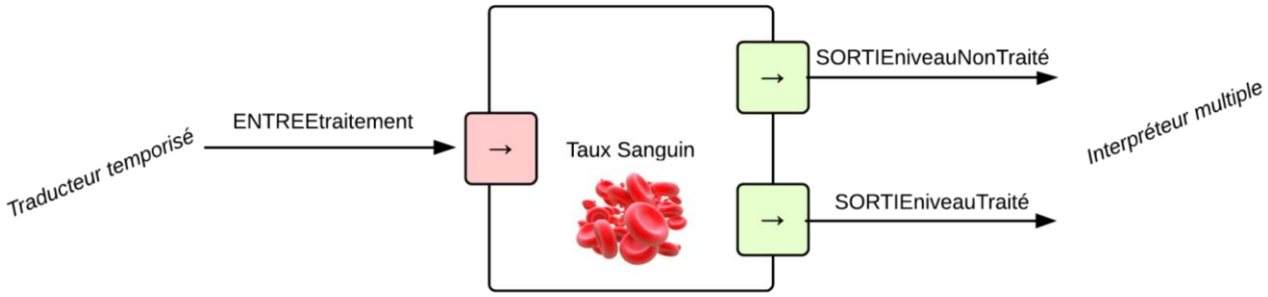
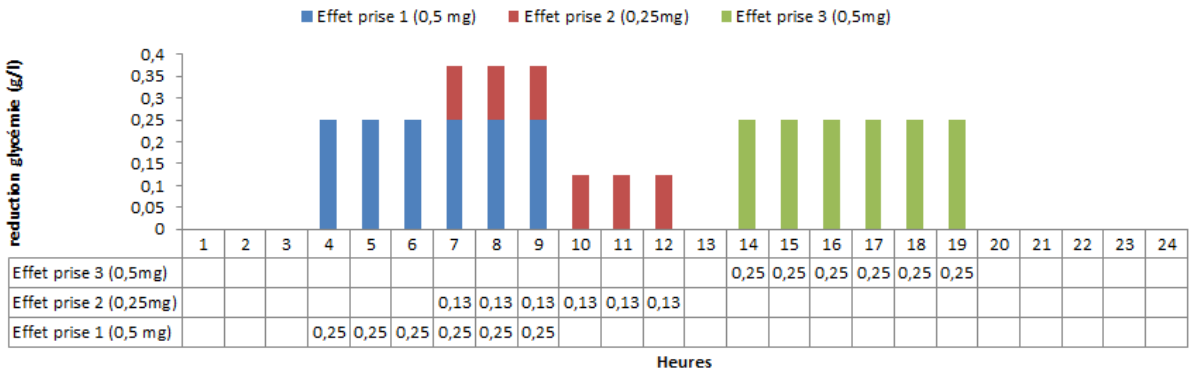


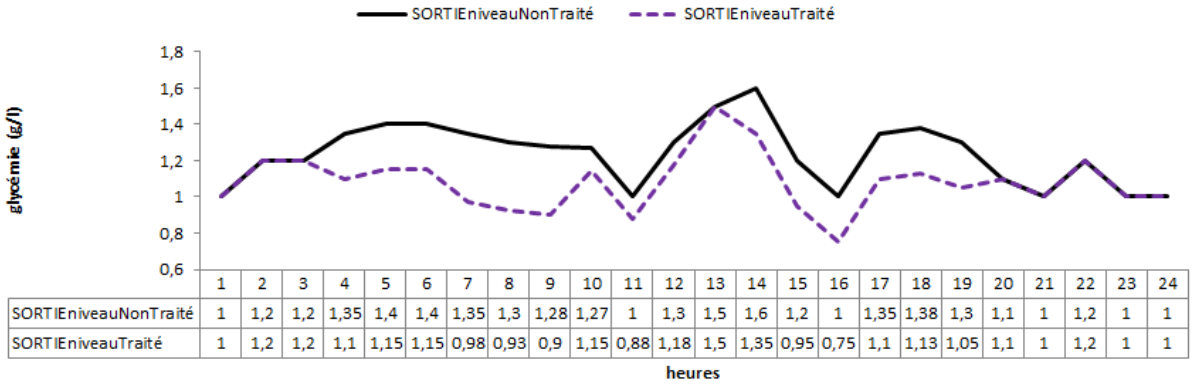
Figure 117 Modèle atomique : « Taux Sanguin »

Lorsqu'un traitement est reçu sur le port d'entrée du modèle celui-ci met à jour sa liste d'effets liés au traitement. La Figure 118 décrit un exemple d'entrées sur le modèle. Nous retrouvons en abscisse les différentes heures de la journée et en ordonnée le niveau de réduction de la glycémie du patient, en fonction des effets du traitement. Comme nous pouvons le voir, trois prises de valeurs différentes (0.5, 0.25 et 0.5) sont reçues à différents instants (t4, t7, t14). Chacune de ces prises offre une durée d'action de six heures. Ces différents effets peuvent se chevaucher et ainsi être cumulés (t7, t8, t9). À partir de cette liste d'effets, le modèle génère deux valeurs en sortie toutes les heures : le niveau traité et le niveau non-traité. Un exemple de sorties pouvant alors être générées par le modèle durant sa simulation est présenté sur la Figure 119.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Effet prise 3 (0,5mg)														0,25	0,25	0,25	0,25	0,25	0,25					
Effet prise 2 (0,25mg)							0,13	0,13	0,13	0,13	0,13	0,13												
Effet prise 1 (0,5 mg)				0,25	0,25	0,25	0,25	0,25	0,25															

Figure 118 Exemple d'effet du traitement



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
SORTIEniveauNonTraité	1	1,2	1,2	1,35	1,4	1,4	1,35	1,3	1,28	1,27	1	1,3	1,5	1,6	1,2	1	1,35	1,38	1,3	1,1	1	1,2	1	1
SORTIEniveauTraité	1	1,2	1,2	1,1	1,15	1,15	0,98	0,93	0,9	1,15	0,88	1,18	1,5	1,35	0,95	0,75	1,1	1,13	1,05	1,1	1	1,2	1	1

Figure 119 Exemple de sorties pour un traitement

L'optimisation paramétrique des entrées de ce modèle a été testée sur une application relative à la régulation du niveau de glycémie.

5.4.3. Résultats obtenus

5.4.3.1. Optimisation du traitement médical de patient diabétique

Actuellement le nombre de diabétiques dans le monde s'élève à 347 millions de personnes. L'OMS estime qu'à partir de 2013, le diabète deviendra la 7ième cause de décès dans le monde. Le traitement de ces individus se doit d'être le plus efficient possible afin de limiter les surcoûts liés à des traitements inadaptés tout en maximisant la qualité de vie des patients.

Trois molécules dont les caractéristiques sont présentées dans le Tableau 37 ont été étudiées sur trois patients ayant des profils de glycémiques distincts. Les différentes molécules proposent des réductions du niveau de glycémie sur des durées variables. Des doses maximales sont également définies.

	Effet /Mg	Durée (heures)	Dose maximale (Mg)
Molécule A	-0,5	6	5Mg
Molécule B	-0,25	9	20Mg
Molécule C	-1,00	3	5Mg

Tableau 37 Propriétés des différentes molécules

L'objectif de l'application est donc de minimiser des niveaux de glycémie non traités de trois patients visibles sur la Figure 120, vers un niveau de glycémie se rapprochant au maximum de la

valeur moyenne journalière (1g / litre de sang). Comme nous pouvons le voir sur cette figure chacun des trois patients présente un profil glycémique spécifique, très différent des autres.

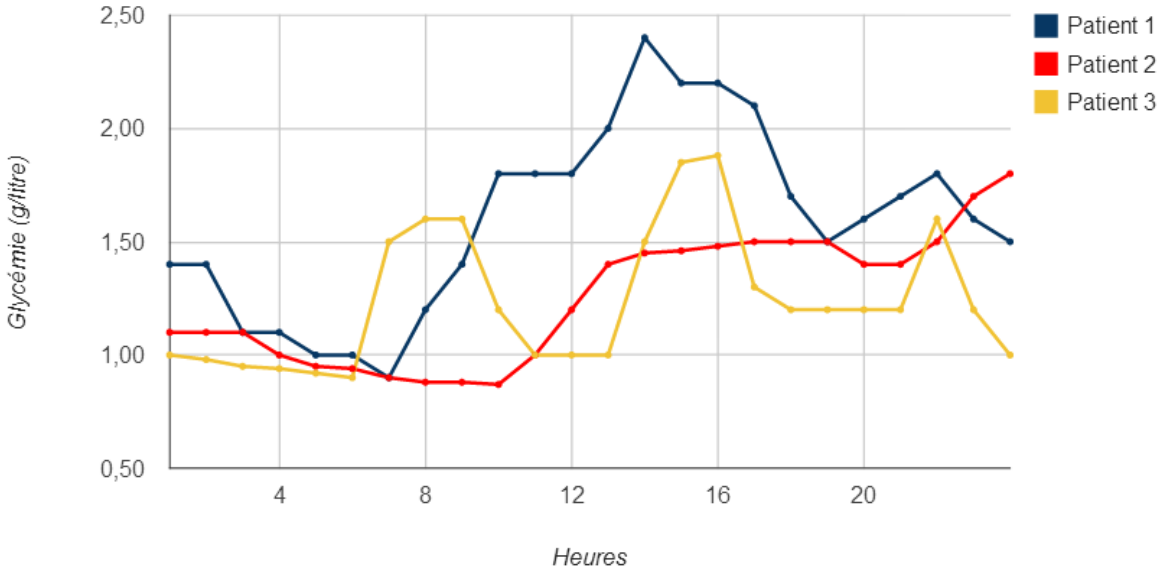


Figure 120 Profils glycémiques des trois patients (non traités)

Afin de répondre à cette demande, nous avons défini la structure de modèles visible sur la Figure 121. Comme nous pouvons le constater, un adaptateur temporisé de nombres flottants (Adapter_Temporized_Float_2) et un interpréteur multiple (Interpreter_Ecart_Efficient_4) sont utilisés pour l'adaptation de l'optimisation sur le modèle représentant le problème (AM_Glycemia_3).

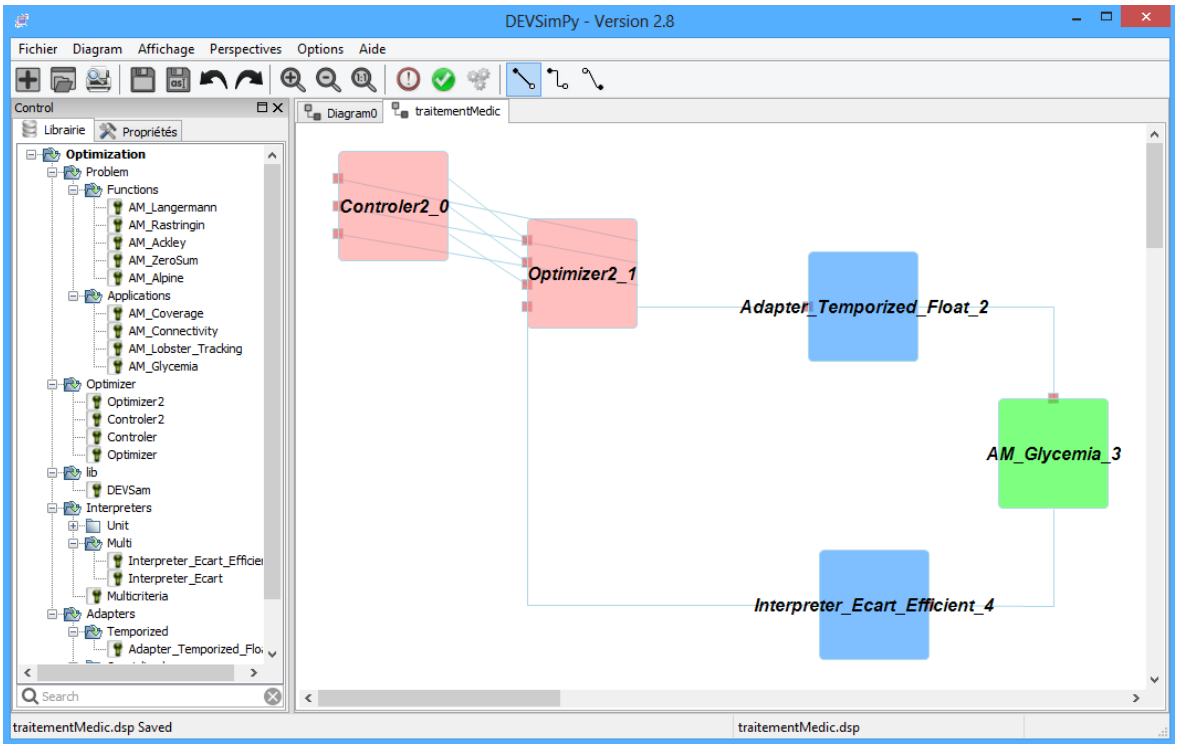
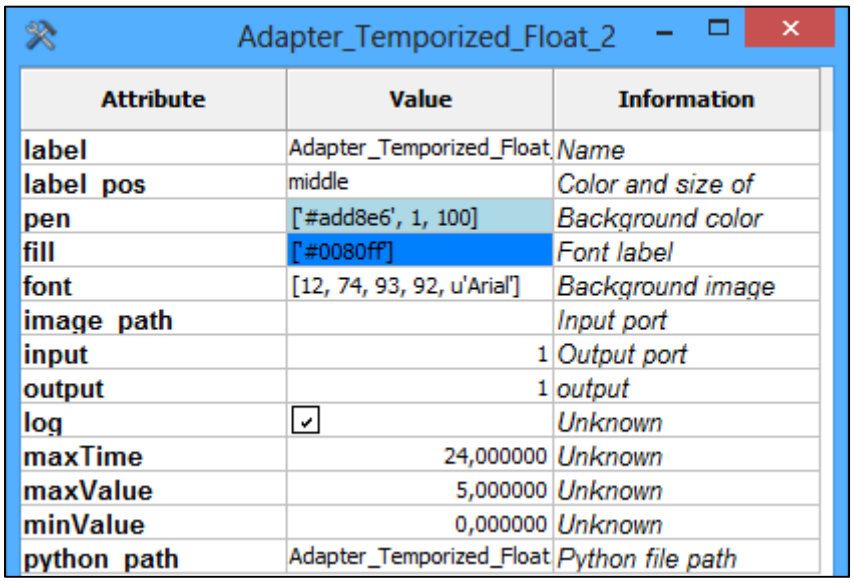


Figure 121 Optimisation d'un traitement médical via simulation dans DEVSimPy

La configuration de l'adaptateur est décrite sur la Figure 122. Nous pouvons constater que ce modèle d'adaptation DEVSimPy va traduire un flux binaire en une suite de valeurs temporisées.

L'attribut « maxTime » permet de décrire la plus grande valeur de temps associée à une variable. Notre modèle de glycémie fonctionnant sur des cycles de vingt-quatre heures, c'est cette valeur qui a été définie. Les attributs « maxValue » et « minValue » permettent de décrire la valeur minimale et la valeur maximale de la traduction. Pour cette application, l'attribut « maxValue » permet de décrire la quantité maximale de prise d'un traitement (e. g. 5mg).

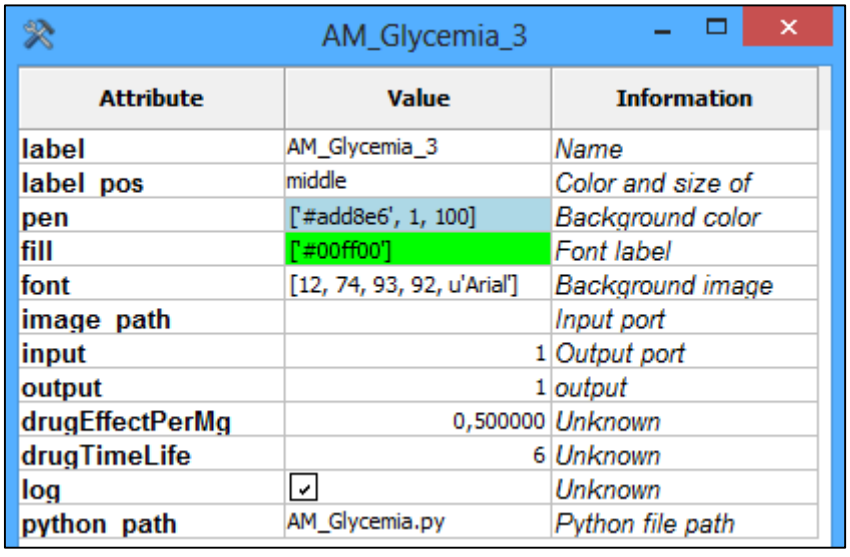


Attribute	Value	Information
label	Adapter_Temporized_Float	Name
label_pos	middle	Color and size of
pen	['#add8e6', 1, 100]	Background color
fill	['#0080ff']	Font label
font	[12, 74, 93, 92, u'Arial']	Background image
image_path		Input port
input	1	Output port
output	1	output
log	☑	Unknown
maxTime	24,000000	Unknown
maxValue	5,000000	Unknown
minValue	0,000000	Unknown
python_path	Adapter_Temporized_Float	Python file path

Figure 122 Configuration de l'optimisation de variables temporisées

La configuration du modèle représentant le niveau de glycémie est visible sur la Figure 123. Deux attributs sont essentiels :

- « drugEffectPerMg » qui permet de décrire la réduction du taux pour un milligramme de molécule
- « drugTimeLife » qui permet de décrire la durée de cette réduction en heures



Attribute	Value	Information
label	AM_Glycemia_3	Name
label_pos	middle	Color and size of
pen	['#add8e6', 1, 100]	Background color
fill	['#00ff00']	Font label
font	[12, 74, 93, 92, u'Arial']	Background image
image_path		Input port
input	1	Output port
output	1	output
drugEffectPerMg	0,500000	Unknown
drugTimeLife	6	Unknown
log	☑	Unknown
python_path	AM_Glycemia.py	Python file path

Figure 123 Configuration du métabolisme du patient

Enfin, la configuration du modèle d'interprétation des résultats est décrite dans la Figure 124. Nous retrouvons sur cette capture d'écran l'attribut « bufferSize » qui permet de définir le nombre

maximum de résultats que le modèle pourra collecter avant d'émettre un message « stop » en direction du modèle de représentation de la glycémie (AM_Glycémia_3). L'attribut « expectedValues » permet, quant à lui, de définir les vingt-quatre valeurs que le modèle souhaite collecter et à partir desquelles sera mesurée la qualité de la solution évaluée. Enfin, les attributs « memoryBestSize » et « numberOutputToStart » permettent de spécifier la taille de la liste des meilleures évaluations mémorisées ainsi que le nombre de sorties à collecter avant de pouvoir démarrer le mode « simulation économe », que nous présenterons ultérieurement.

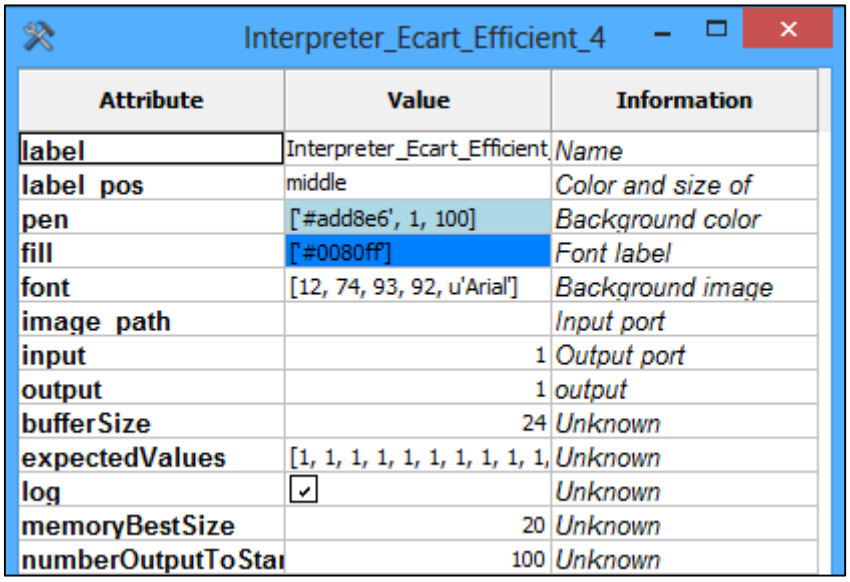


Figure 124 Configuration de l'interprétation des résultats

Pour chaque patient, la posologie (heure de prise et quantité) de trois traitements sera optimisée en vue de choisir la mieux adaptée aux spécificités du patient.

- Pour le patient 1, dont les résultats pour chacune des molécules sont présentés sur la Figure 125, la Figure 126 et la Figure 127 et résumés dans le Tableau 38. Sur les trois figures, trois courbes sont visibles. Elles décrivent les niveaux de glycémie dans la journée avec traitement, sans traitement et présente le niveau de glycémie idéal. A partir des valeurs présentes dans ces courbes, nous pouvons déduire que la molécule la mieux adaptée est la molécule B. En effet, la somme des écarts entre le niveau de diabète souhaité et le niveau de diabète observé durant l'optimisation par simulation est de 3.62, soit une réduction de 74% par rapport au niveau non traité. La molécule A offre des résultats analogues à la molécule B, son choix peut donc être envisagé par le décideur. Ce n'est pas le cas de la molécule C qui de par sa faible durée de vie, ne convient pas au profil glycémique du patient.
- Pour le patient 2 les résultats sont présentés sur les figures Figure 128, Figure 129 et Figure 130 et sont résumés dans le Tableau 39. Comme nous pouvons l'observer, la molécule B semble la mieux adaptée. Le meilleur résultat obtenu est de 1.85, soit une réduction elle aussi égale à 74% en comparaison du niveau non traité. Les molécules A et C présentent ponctuellement des écarts de niveau importants, qui pénalisent l'évaluation finale et ne peuvent par conséquent pas être choisies.
- Pour le patient 3 les résultats sont visibles sur les figures Figure 131, Figure 132 et Figure 133 et sont résumés dans le Tableau 40. Suite aux différentes optimisations par simulation pour les trois molécules, il semble que la molécule C soit la plus adaptée avec une évaluation égale à 2.80, soit une réduction des hypo et hyper glycémies de 55%. En effet, celle-ci réduit la

somme des écarts entre les valeurs de glycémie idéales et celles observées par le traitement à une valeur de 2.80.

	1er prise		2ième prise		3ième prise		Evaluation
Molécule A	1.2890151 mg	6h	2.4902926 mg	11h	1.35929668 mg	16h	3.8909313
Molécule B	3.2300090 mg	8h	2.4042661 mg	11h	3.1226972 mg	19h	3.627765
Molécule C	1.1831233 mg	10h	1.0093572 mg	13h	0.53276329 mg	17h	7.4222881

Tableau 38 Traitements proposés pour le patient 1

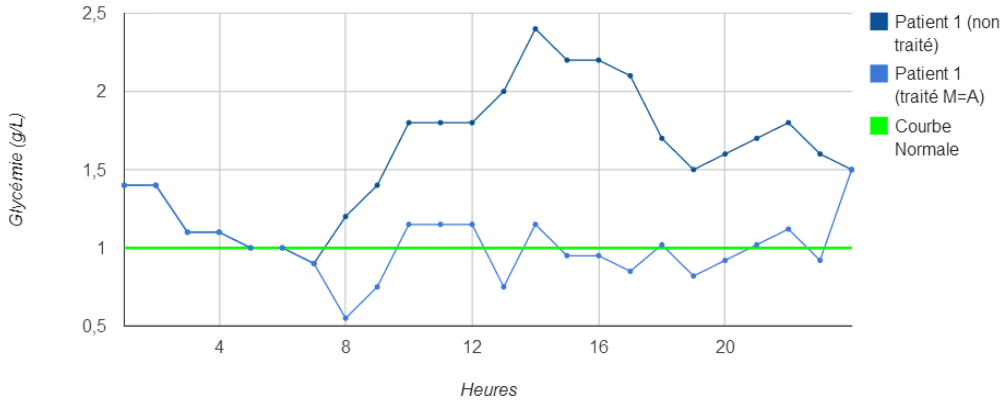


Figure 125 Evolution de la Glycémie pour le patient 1 avec la molécule A

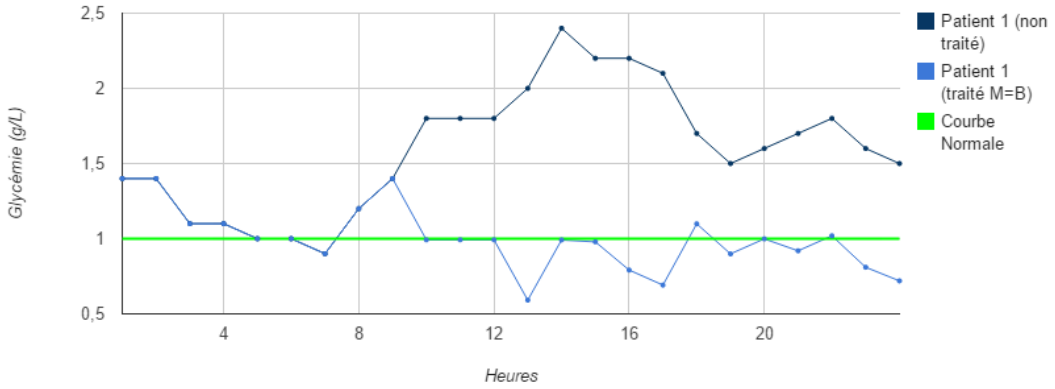


Figure 126 Evolution de la Glycémie pour le patient 1 avec la molécule B

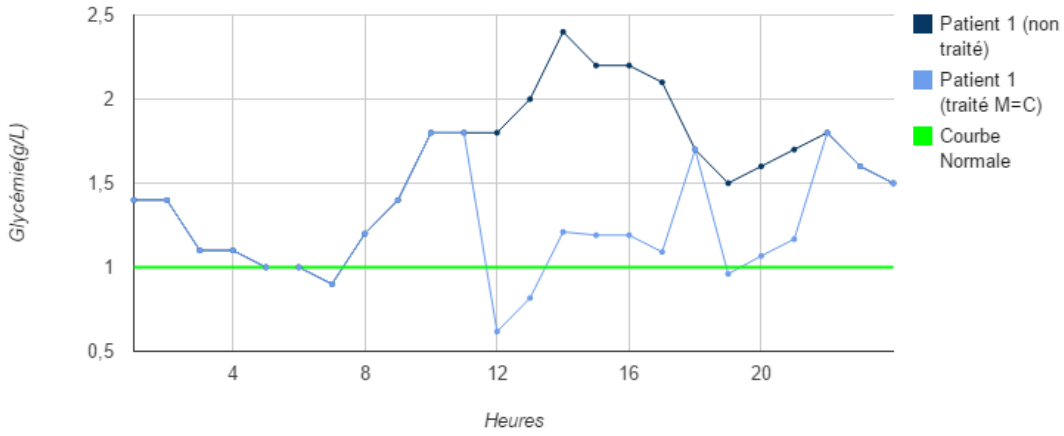


Figure 127 Evolution de la Glycémie pour le patient 1 avec la molécule C

	1er prise		2ième prise		3ième prise		Evaluation
Molécule A	0.96259911 mg	12h	1.210052 mg	18h	0.26580811 mg	21h	2.205078
Molécule B	0.0067225145 mg	6h	1.6366597 mg	11h	2.8350724 mg	20h	1.8546345
Molécule C	0.45571163 mg	12h	0.40658115 mg	15h	0.66715988 mg	21h	3.2602251

Tableau 39 Traitements proposés pour le patient 2

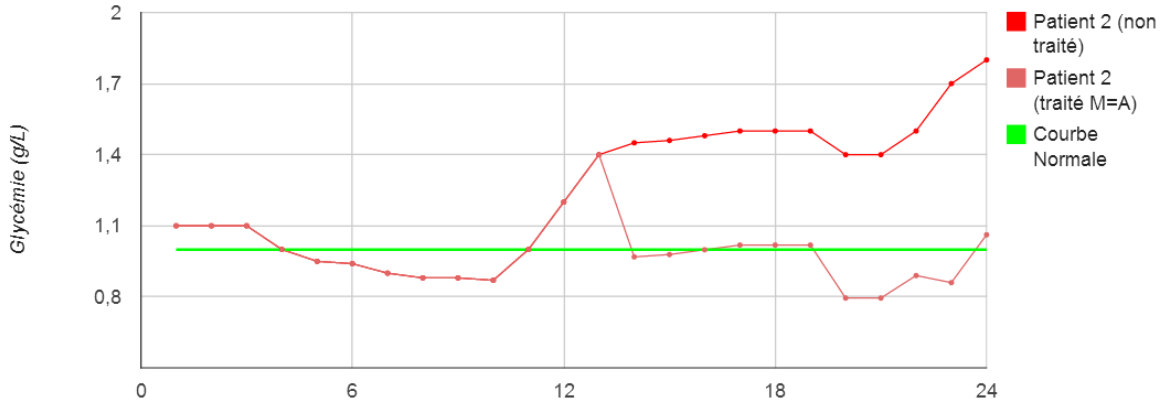


Figure 128 Evolution de la Glycémie pour le patient 2 avec la molécule A

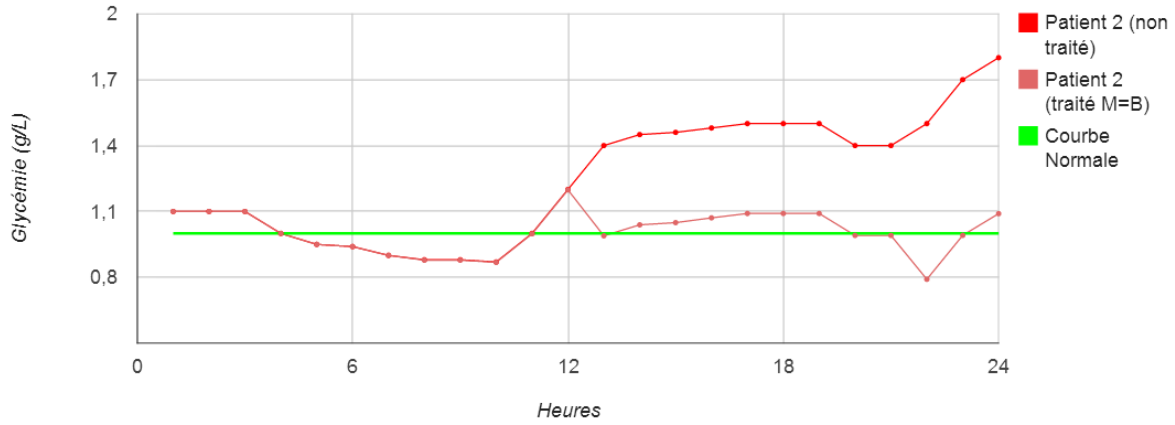


Figure 129 Evolution de la Glycémie pour le patient 2 avec la molécule B

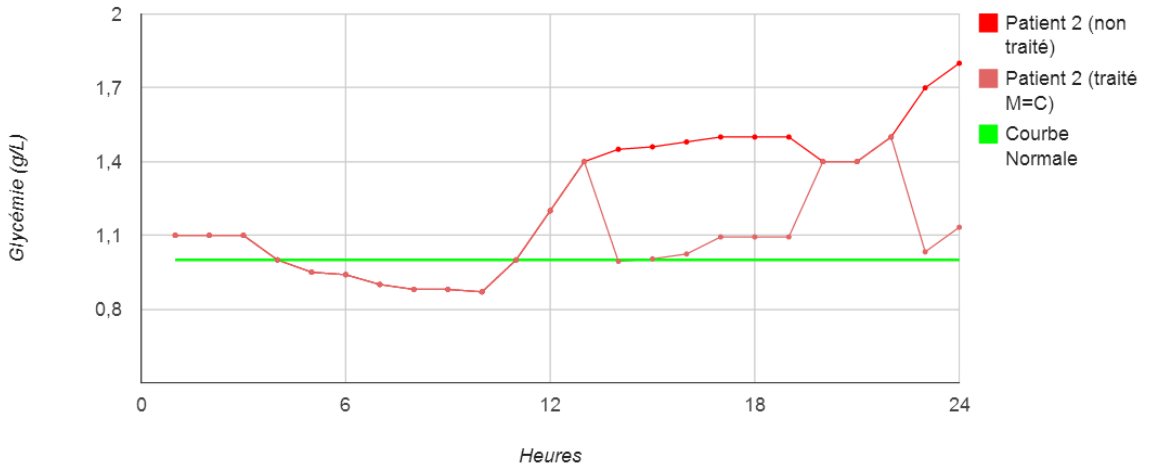


Figure 130 Evolution de la Glycémie pour le patient 2 avec la molécule C

	1er prise		2ième prise		3ième prise		Evaluation
Molécule A	0.6274507 mg	4h	1.0519677 mg	11	0.4633260 mg	16h	3.7926360
Molécule B	0.7325793 mg	7h	1.3158151 mg	12	0.44256814 mg	21h	4.1268613
Molécule C	0.6311306 mg	5h	0.55368963 mg	12	0.15683620 mg	15h	2.8091937

Tableau 40 Traitements proposés pour le patient 3

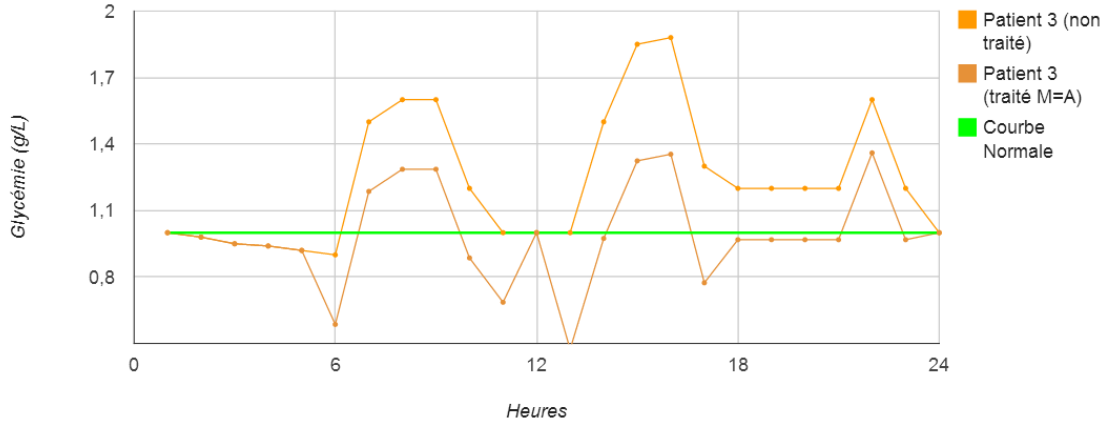


Figure 131 Evolution de la Glycémie pour le patient 1 avec la molécule A

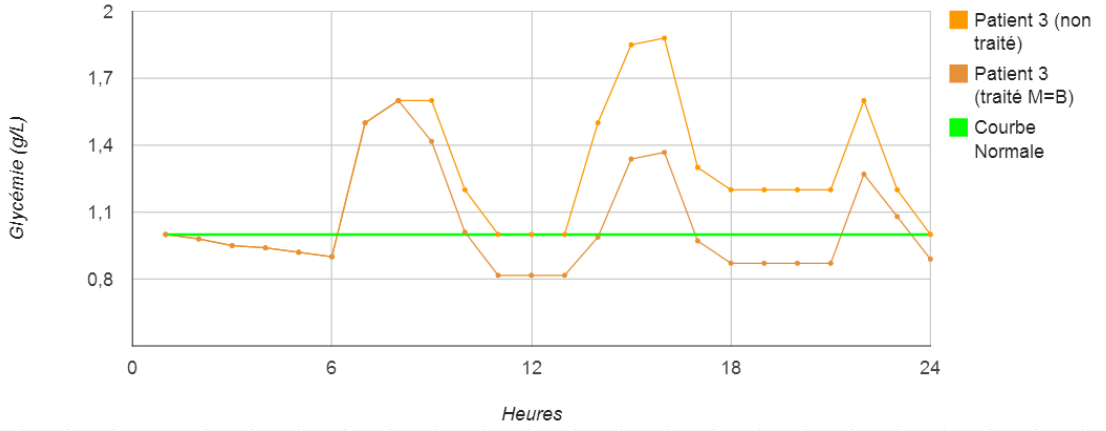


Figure 132 Evolution de la Glycémie pour le patient 3 avec la molécule B

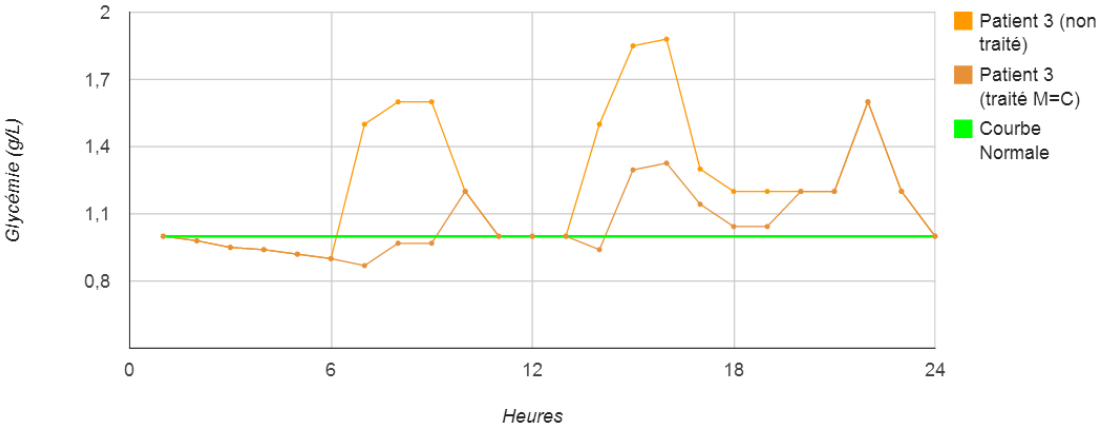


Figure 133 Evolution de la Glycémie pour le patient 3 avec la molécule C

5.4.3.2. Simulation économe

L'optimisation d'un traitement médical se base sur des simulations d'une durée maximale de vingt-quatre heures. Cependant les simulations de certains traitements inadaptés peuvent être interrompues avant cette durée. Ces interruptions peuvent être déterministes ou probabilistes en fonction du choix du décideur. Elles permettront de réduire le temps total du processus d'optimisation.

Les « interruptions déterministes » interrompent une simulation dans l'unique cas où les résultats collectés pour la solution en cours d'évaluation assurent de ne pas obtenir une évaluation de meilleure qualité que la moins bonne solution potentielle conservée en mémoire, quelles que soient les prochaines valeurs collectées.

Les « interruptions probabilistes » peuvent interrompre une simulation avant que les résultats produits assurent d'obtenir une évaluation inférieure à la moins bonne solution potentielle conservée en mémoire. En effet, la probabilité d'interruption d'une solution est proportionnelle à la proximité des résultats collectés avec ceux de la plus mauvaise solution.

Les pourcentages de simulations interrompues durant l'optimisation pour chacun de ces deux modes sont visibles sur la Figure 134. Comme nous pouvons le voir sur la Figure 135 et la Figure 136, deux groupes d'évaluations peuvent être isolées : les « simulations complètes » et les « simulations interrompues ». Dans les deux modes, nous pouvons observer une progression logarithmique du nombre de « simulation complète » et une progression polynomiale du nombre de « simulations interrompues ». Cependant, nous constatons que le nombre de simulations interrompues est plus important lors de l'utilisation du mode probabiliste qui intègre une certaine anticipation des résultats, que dans le mode déterministe qui ne raisonne que sur des résultats avérés.

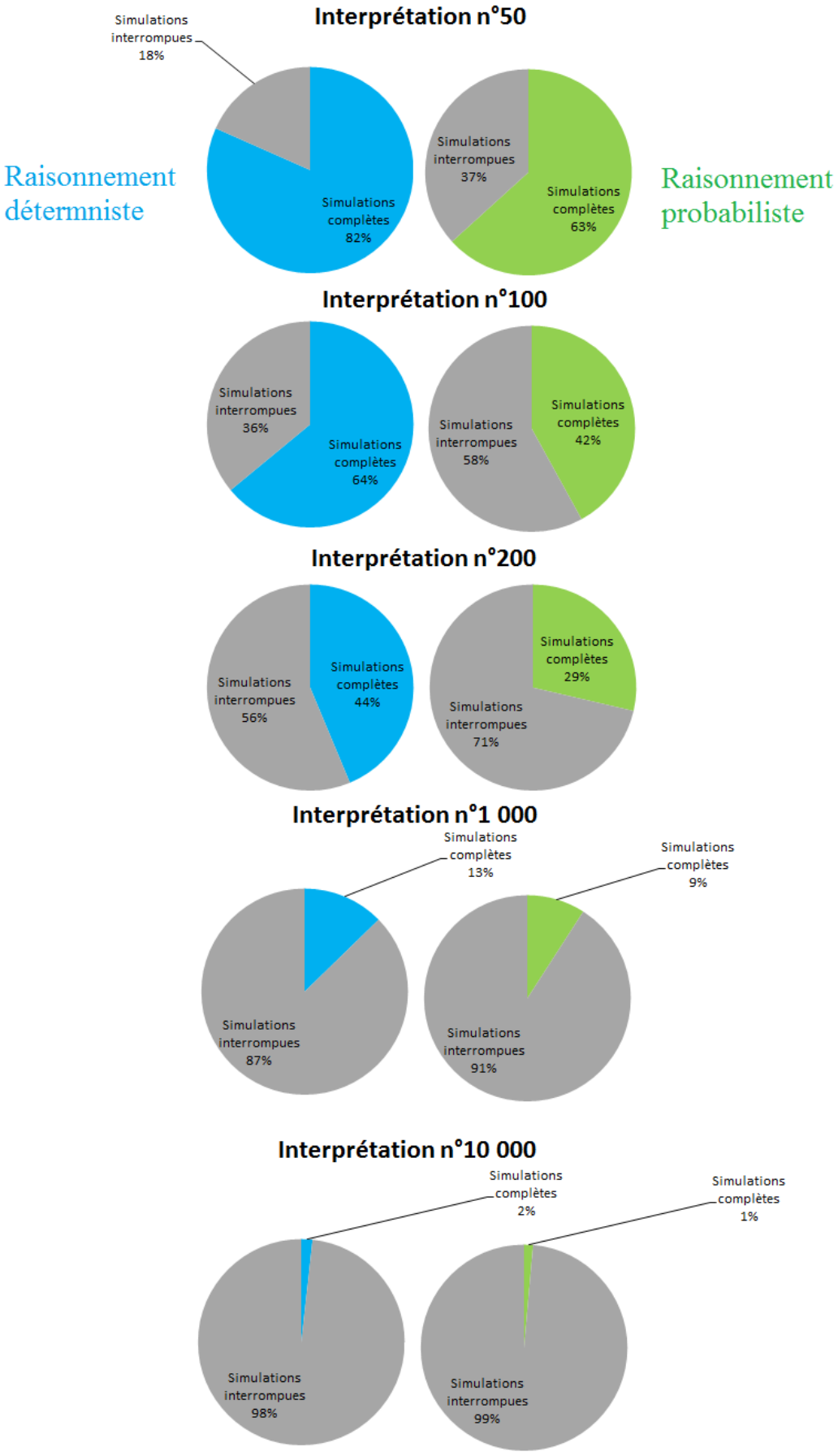


Figure 134 Proportions de simulations complètes et interrompues

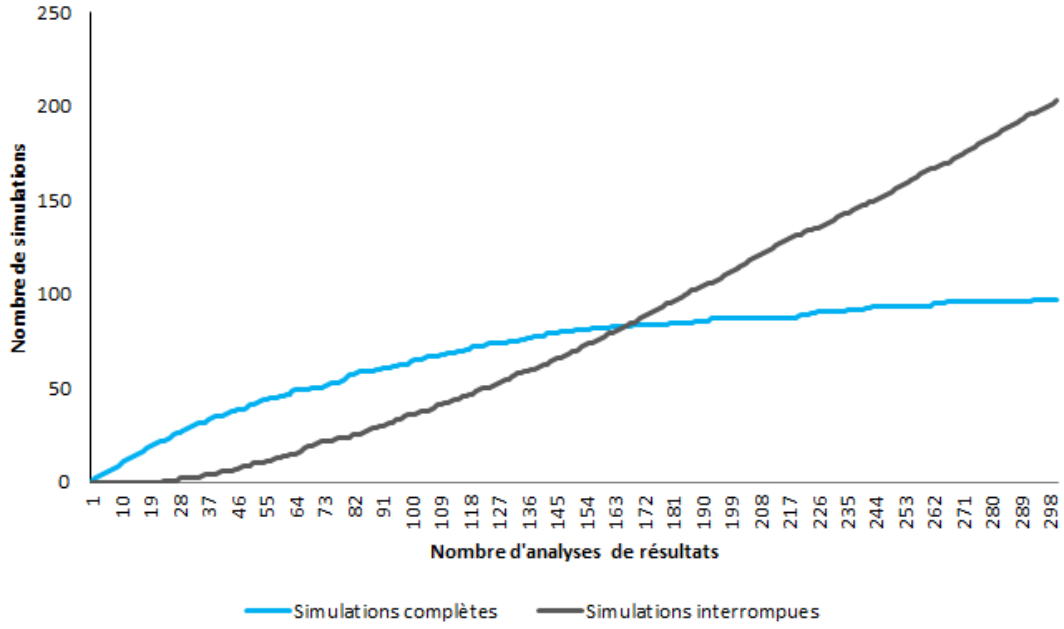


Figure 135 Evolution du nombre de simulations complètes et interrompues (mode déterministe)

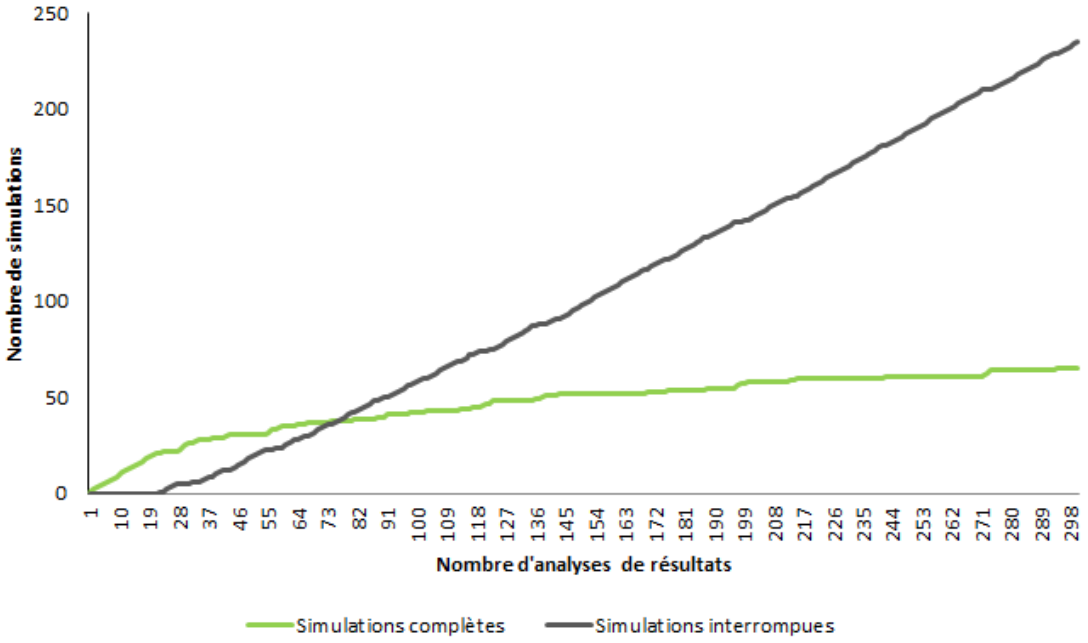


Figure 136 Evolution du nombre de simulations complètes et interrompues (mode probabiliste)

Une fois le nombre d'interruptions mis en évidence, nous avons cherché à connaitre l'ampleur des économies réalisées. En effet l'économie de temps de simulation ne sera pas la même si l'interruption de simulation se produit à la deuxième ou à la vingt troisième heure. Pour mesurer cette ampleur, nous nous sommes basés sur la durée moyenne des simulations au fil des itérations. Les résultats sont présentés sur la Figure 137. Comme nous pouvons le voir, lors du processus itératif les interruptions (émission d'un message « stop » du modèle d'analyse vers le modèle de glycémie) sont de plus en plus précoces. Nous remarquons qu'à la dix millième itération, les simulations de traitement sont en moyenne interrompues à la quatorzième heure pour le mode déterministe et à la douzième heure pour le mode probabiliste. Cela représente une réduction de plus de 41% de la durée de

simulation pour le mode déterministe et 50% pour le mode probabiliste. Dans le cadre de cette application, il semble donc difficile d'obtenir des interruptions avant la dixième heure de simulation du niveau de glycémie. Cependant pour d'autres applications, des interruptions pourront être réalisées encore plus précocement, ou plus tardivement. Cela dépend de l'impact de chaque variable sur la qualité de la solution.

Malgré un nombre d'évaluations plus important (de 16%) et se produisant plus rapidement, la qualité des solutions proposées par le mode d'interruption probabiliste est inférieure à celle du mode déterministe. En effet, comme nous pouvons le voir sur la Figure 138, l'écart entre le niveau de glycémie traité et le niveau de glycémie souhaité est constamment inférieur lors de l'utilisation du mode déterministe. Nous pouvons supposer que le mode probabiliste interrompt certaines évaluations qui pourraient contribuer à l'amélioration des solutions proposées si elles étaient menées à leur terme, réduisant ainsi les performances du processus d'optimisation. Nous pouvons donc envisager d'affiner l'équation du raisonnement probabiliste présente dans le modèle interpréteur de « simulation économe probabiliste » afin de diminuer le nombre d'interruptions de simulation qui peuvent être utiles à la progression de l'optimisation.

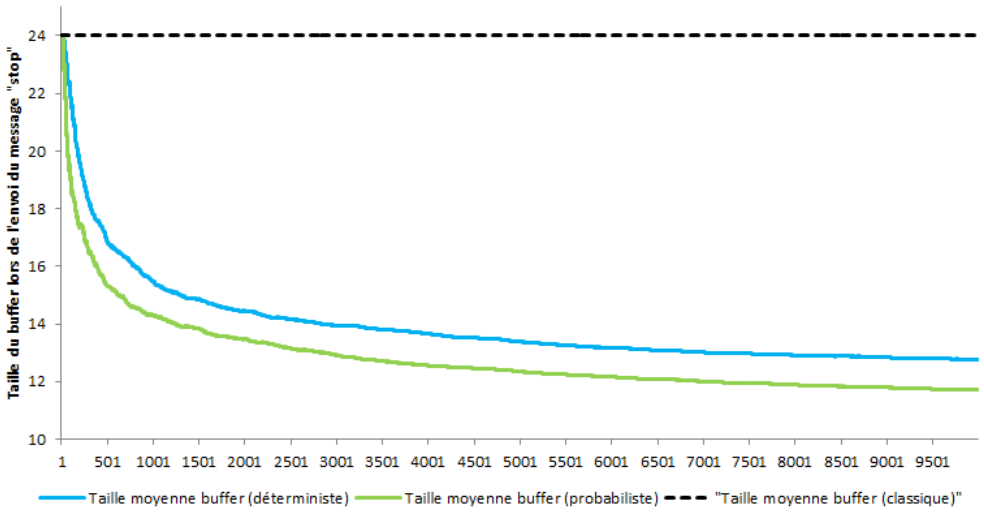


Figure 137 Evolution de la durée moyenne d'une simulation selon les modes d'interruption

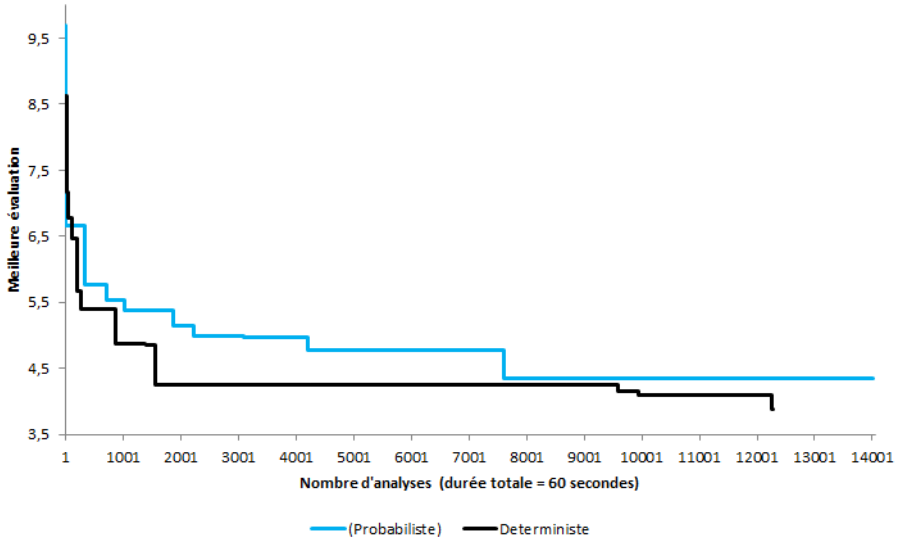


Figure 138 Comparaison de la qualité des solutions proposées

5.5 Conclusion

Ce chapitre a permis de valider l'apport de notre architecture de modèle pour les processus d'optimisation via simulation grâce à la plateforme de modélisation et de simulation DEVSimPy. Cette validation s'est construite autour de trois applications distinctes.

La première application a consisté en l'intégration de fonctions traditionnellement utilisées pour la validation de métaheuristiques sous la forme de modèles DEVS. Cette application a permis de valider notre concept « d'évaluation externalisée » par l'utilisation d'un modèle « adapter » et d'un modèle « interpreter ». L'utilisation de notre librairie de modèles « Optimization » par l'intermédiaire de l'interface graphique de DEVSimPy a démontré la facilité de manipulation de nos modèles ainsi que leur généricité.

La seconde application, dont l'objectif était l'optimisation d'un déploiement de réseaux de capteurs, a généré de nombreux résultats encourageants et a permis de valider trois concepts : « la spatialisation des variables », « l'évaluation multicritère » et « l'optimisation intelligente ». Grâce à « la spatialisation des variables » nous avons proposé différentes stratégies de déploiement déterministes de qualités supérieures à celles proposées par les déploiements aléatoires généralement utilisés pour ce type de réseau. « L'évaluation multicritère » effectuée par le modèle « Grouper » (regroupeur), a permis l'interprétation de résultats de simulation provenant de deux modèles distincts (couverture et connectivité) pour générer une seule et même évaluation. « L'optimisation intelligente » qui consiste à changer d'algorithme, de paramètres et de solutions durant le processus d'optimisation a également pu être testée avec succès. Dans la majorité des scénarios, une convergence plus rapide vers l'optimum a pu être observée. Dans les autres cas, une convergence analogue à celle de la meilleure méthode statique était visible.

La troisième et dernière application concernait l'optimisation de traitements médicaux pour des patients diabétiques. Les résultats générés ont permis d'atteindre, pour trois molécules distinctes, des posologies adaptées aux spécificités de patients diabétiques en réduisant considérablement leur niveau de glycémie. Cette application a dans un même temps, permis de valider les concepts de « temporisation des variables décisionnelles » sur une durée de vingt-quatre heures et de « simulation économe ». L'utilisation de la « simulation économe » a permis de réduire considérablement les temps de simulation grâce à deux modes de fonctionnement (déterministe et probabiliste) qui ont permis de stopper des simulations inutiles au processus d'optimisation par l'émission de message DEVS « Stop ». Les gains de temps que nous avons illustrés confirment l'intérêt de l'intégration de l'optimisation sous forme de modèles simulés. En effet, si l'optimisation était isolée de la simulation, chaque simulation aurait une durée fixe, ce qui n'est pas le cas dans notre approche où les résultats de simulation sont analysés pendant la simulation et non à son terme.

Chapitre 6
Conclusion générale

« L'imagination gouverne le monde »

Napoléon Bonaparte

6.1 Bilan des travaux

L'objectif de nos travaux était de développer une architecture d'optimisation via simulation dans le formalisme DEVS. Concernant l'aspect modélisation et simulation, nous avons naturellement choisi d'utiliser le formalisme « Discret Event System Specification » (DEVS) pour sa généricité, sa fiabilité, sa facilité d'utilisation. Concernant l'optimisation, nous nous sommes focalisés sur les méthodes modernes d'optimisation : les « métaheuristiques ». Ce choix se justifie par les performances, la rapidité et la qualité des solutions obtenues, la facilité d'intégration à DEVS.

Si des travaux ont déjà été réalisés pour des problématiques analogues, ceux-ci se sont concentrés uniquement sur un usage de la simulation comme source d'évaluation pour les méthodes d'optimisation. Ces approches maintiennent une séparation entre simulation et optimisation. Comme nous pouvons le voir sur la Figure 139, notre approche intègre le processus d'optimisation dans le processus de simulation. Cette approche que nous avons choisi de nommée OvS « Full simulation », permet de dynamiser la configuration des modèles d'optimisation qui sont eux même simulés, permettant ainsi des changements d'algorithmes, de paramètres et de solutions durant l'optimisation, apportant ainsi des solutions de meilleure qualité que celles proposées par le choix empirique d'algorithmes.

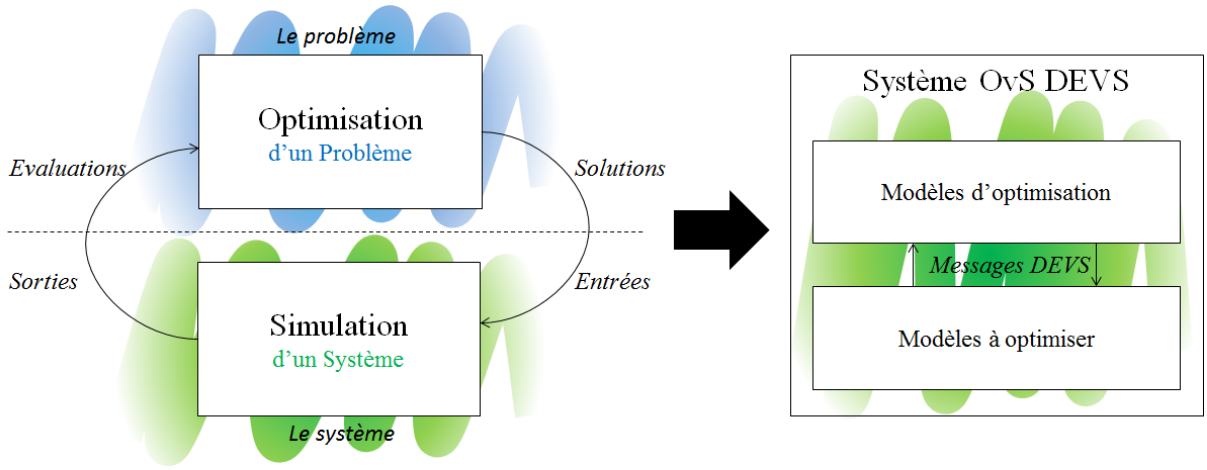


Figure 139 OvS « Full simulation »

Notre approche est fondée sur une architecture de modèles DEVS. De cette architecture émergent deux concepts : « l'optimisation intelligente » et « l'évaluation externalisée ». « L'optimisation intelligente » est modélisée à l'intérieur de deux modèles : « le contrôleur » et « l'optimiseur ». « Le contrôleur » permet la mise à jour dynamique de l'algorithme d'optimisation, de ses paramètres ainsi que de solutions aléatoires nécessaires lors de certaines étapes. Ce modèle peut être sollicité à tout moment durant la simulation afin de générer une nouvelle configuration. « L'optimiseur » permet quant à lui, la coordination des évaluations externalisées ainsi que la mise à jour des solutions en fonction des résultats collectés. Cette « évaluation externalisée » repose sur deux groupes de modèles : « les adaptateurs » et les « interpréteurs ». « Les adaptateurs » permettent d'adapter la description binaire d'une solution envoyée par l'optimiseur en direction du modèle représentant le problème étudié. À ce jour, les flux binaires peuvent être traduits dans les différents types primitifs (entiers, flottants, caractères, booléens) généralement utilisés dans le contenu des messages circulant entre des modèles DEVS. Le second groupe de modèle contient les « interpréteurs ». Ces modèles permettent la collecte et l'analyse des sorties du modèle représentant le

problème durant la simulation. Les « interpréteurs unitaires » collectent une seule valeur. Dans le cas de collecte d'un ensemble de valeurs nous appelons ces modèles « interpréteurs multiples ». Ces derniers permettent d'utiliser notre concept de « simulation économe ». Ils peuvent alors interrompre la simulation du modèle représentant le problème étudié, lorsque les résultats fournis par la simulation ne sont pas satisfaisants. Ce fonctionnement permet de réduire le temps de simulation de certaines solutions potentielles et par conséquent de diminuer le temps total du processus d'optimisation. Ces interruptions reposent sur l'émission de messages « stop » dont le déclenchement évènementiel peut être réalisé à partir de deux modes : le mode déterministe et le mode probabiliste, que nous avons présentés.

Ces différents concepts ont été implémentés dans le logiciel de modélisation et de simulation DEVSimPy et forment une « librairie de modèles réutilisables ». Trois types d'applications ont été développés pour mettre en lumière les apports de nos travaux. La première série de tests a permis l'optimisation des vecteurs de variables en vue de trouver les minimums globaux de fonctions multidimensionnelles traditionnellement utilisées pour la validation de méthodes d'optimisation. Ces tests ont permis de mettre en évidence la facilité d'utilisation de nos modèles pour la modélisation et la simulation du processus d'optimisation et ont débouché sur l'obtention de résultats cohérents pour des temps acceptables.

Une seconde série de tests concerne l'optimisation de la couverture et de la connectivité d'un réseau de capteurs sans-fils en fonction des caractéristiques de différentes zones. À travers trois scénarios, nous avons pu observer la bonne construction des stratégies de déploiement et valider nos concepts « d'optimisation spatialisée » et « d'optimisation intelligente ». En effet, cette application a permis à notre modèle d'optimisation de s'adapter au modèle représentant le problème qui impliquait un aspect géographique. Le choix automatique des algorithmes utilisés ainsi que leurs paramétrages ont été comparés à deux configurations statiques d'optimisation. Les résultats issus de cette automatisation offrent une convergence plus rapide des solutions en direction de l'optimum, que celles des configurations statiques.

Enfin, l'architecture de modèles proposée a également été mise en œuvre pour l'optimisation de traitements médicaux de personnes diabétiques. Les résultats obtenus ont permis de choisir parmi trois molécules celle offrant le meilleur potentiel d'optimisation en fonction du profil de chaque patient. Ces simulations de processus d'optimisation ont permis de valider notre concept « d'optimisation temporisée ». Dans le même temps elles ont aussi mis en lumière l'apport de notre concept de « simulation économe » et ses deux modes : déterministe et probabiliste. Bien que ces deux modes aient réduit de moitié les temps de simulation, pour cette application, c'est le mode déterministe qui a présenté les meilleures performances aux niveaux de la qualité des solutions et de la rapidité d'obtention.

Les premiers résultats obtenus sont plus qu'encourageants et confirment l'intérêt de notre démarche(Poggi et al., 2014). Dans ce contexte nous souhaitons approfondir nos travaux de recherche et élargir ainsi les domaines d'application. Ces différentes perspectives sont présentées dans la partie suivante.

6.2 Perspectives

Notre outil d'optimisation par simulation ouvre plusieurs perspectives pouvant être définies par quatre axes :

- l'amélioration des performances
- l'accessibilité par un langage dédié
- l'optimisation comportementale
- l'usage de l'outil comme plateforme de test et de comparaison de métaheuristiques

L'utilisation de simulations exécutées en parallèle semble être un élément incontournable pour la réduction du temps de processus d'optimisation via simulation. Face à ce constat, nous souhaitons permettre l'externalisation des évaluations de manière parallèle. Cette parallélisation des processus pourra se faire à une échelle locale mais également être délocalisée sur des serveurs de simulation grâce à l'utilisation de web services. Le basculement de nos modèles vers l'extension « Parallel DEVS » (PDEVS) ne nécessiterait que peu de modifications. Il est important de préciser que la parallélisation ne se limiterait pas à l'évaluation des solutions potentielles par la simulation, mais pourrait également être utilisée pour la mise à jour des différentes solutions potentielles. Il serait donc intéressant de multiplier le nombre de modèles optimiseurs utilisés lors d'un même processus d'optimisation afin de permettre des simulations indépendantes pouvant ponctuellement échanger des informations avec leurs homologues en vue de « doper » l'optimisation.

Pour améliorer la facilité d'utilisation de nos méthodes et concepts, nous souhaitons créer un langage dédié à l'optimisation via simulation. Ce langage permettrait à la fois de modéliser des systèmes et de définir rapidement les critères souhaités pour leur optimisation paramétrique. Celui-ci reposerait sur une syntaxe simple et universelle appréhendable rapidement, comme celle présente sur la Figure 140. Sur cet exemple, nous pouvons voir la description d'un processus d'optimisation paramétrique de deux variables : « x et y » transmises sur les entrées du modèle « addition » générant une sortie « somme » qui représente la somme des deux entrées « x et y ». L'objectif de ce processus d'optimisation est la « minimisation » de cette sortie par l'utilisation de la recherche harmonique.

```
DEFINE x MIN = 0 MAX=10
DEFINE y MIN = 0 MAX=10
MODEL addition INPUT x,y OUTPUT somme
GENERATE addition OUTPUT somme IS x+y WHEN RECEIVE x AND RECEIVE y
OPTIMIZE MODE minimisation
OPTIMIZE ALGORITHM recherche harmonique
OPTIMIZE somme
```

Figure 140 Exemple de syntaxe pour l'optimisation via simulation

Nous souhaitons aussi permettre l'optimisation du comportement interne de certains modèles étudiés à partir de jeux de données. Comme cela est visible sur la Figure 141, cette optimisation du comportement se baserait sur un modèle interpréteur spécifique contenant dans sa mémoire interne un jeu de données regroupant différents couples « entrées-sorties » observés sur le système réel. À partir de deux entrées : le comportement et le contexte d'étude, le modèle générerait des sorties. Celles-ci seraient comparées aux sorties attendues par l'interpréteur. Cette comparaison permettrait de générer

une évaluation qui serait utilisée par l'optimiseur pour mettre à jour les différentes solutions potentielles représentant chacune une proposition de comportement.

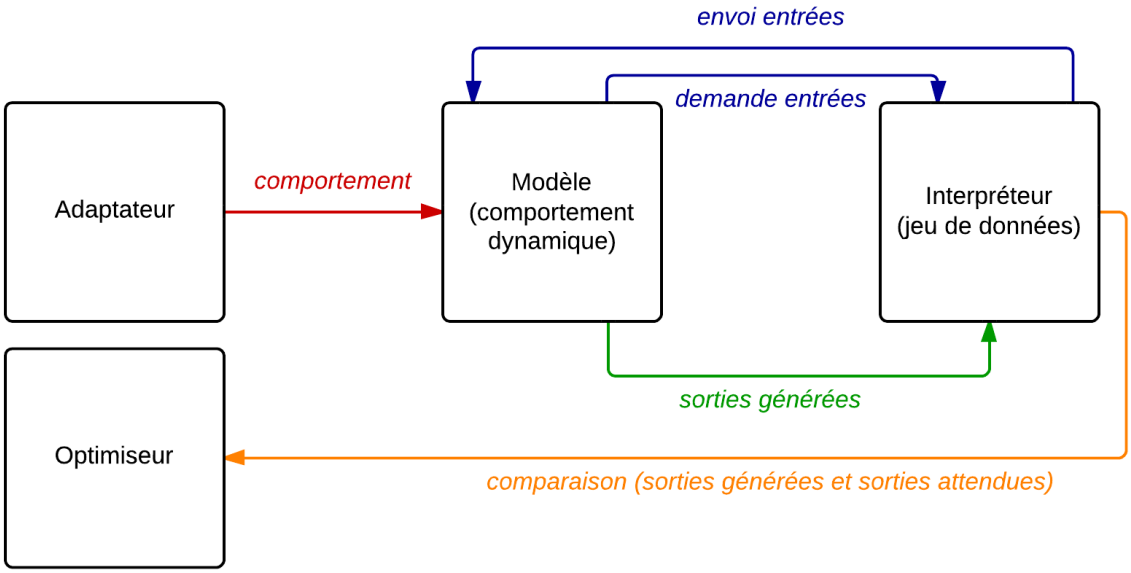


Figure 141 Optimisation comportementale

Un exemple d'optimisation comportementale est présent sur la Figure 142. Comme nous pouvons le voir sur cet exemple deux messages parviennent aux ports d'entrées du modèle contenant un « contexte » et un « comportement ». Dans cet exemple, le comportement est décrit sous la forme d'une liste d'entiers permettant de choisir différents chemins sur un graphe d'opérateurs. La formule obtenue par ce parcours est appliquée aux différentes valeurs décrivant le contexte et permet ainsi de gérer une sortie qui sera ensuite comparée aux résultats attendus pour ce contexte.

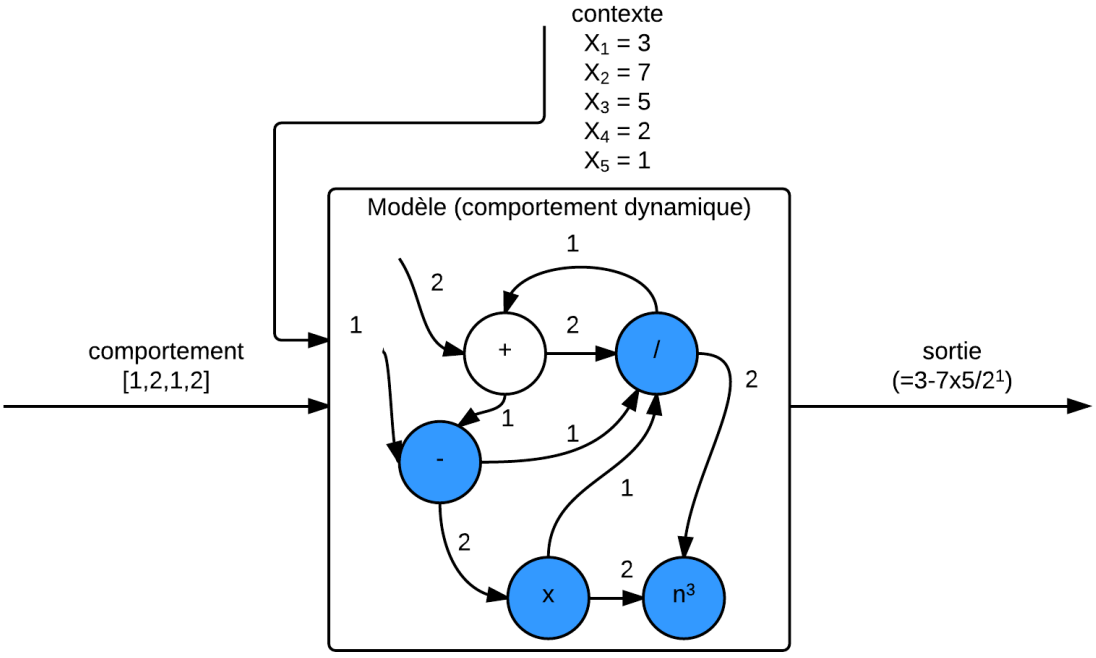


Figure 142 Exemple d'optimisation comportementale

Enfin, nous envisageons également d'utiliser notre outil comme plateforme de test, de comparaison et de validation de métaheuristiques. En effet, de nombreuses méthodes d'optimisation sont régulièrement développées. Leur intégration dans notre outil serait relativement simple et permettrait la génération de résultats permettant leur comparaison à partir de différents « benchmarks » telles que les fonctions multidimensionnelles déjà implémentées. Un exemple de comparaison envisagé est donné sur la Figure 143. Comme nous pouvons le voir, dans cet exemple trois métaheuristiques sont choisies à l'intérieur de trois optimiseurs. Chacun d'eux dispose de son propre réseau d'optimisation composé d'un adaptateur, du modèle représentant le « benchmark » ainsi que des interpréteurs. L'évaluation calculée par l'interpréteur est bien entendu transmise à l'optimiseur associé, afin de permettre la mise à jour des solutions. Cependant, cette évaluation est également transmise à un modèle comparateur dont l'objectif est la collecte de ces résultats à l'intérieur d'un tableau qui permettra de comparer l'évolution des différentes solutions évaluées. De plus, la description des « benchmark » sous forme de modèles simulables, permet d'enrichir de nombreux problèmes d'optimisation traditionnellement utilisés. Cet aspect permettrait de tester la robustesse des méthodes d'optimisation sur des problèmes toujours plus complexes. Nous pouvons par exemple imaginer un problème « des voyageurs de commerce avec embouteillages » inspiré par le problème « du voyageur de commerce » dans lequel les coûts de chaque lien entre chaque ville seraient conditionnés par le nombre de voyeurs l'empruntant ainsi que les conditions de circulation en fonction de l'heure.

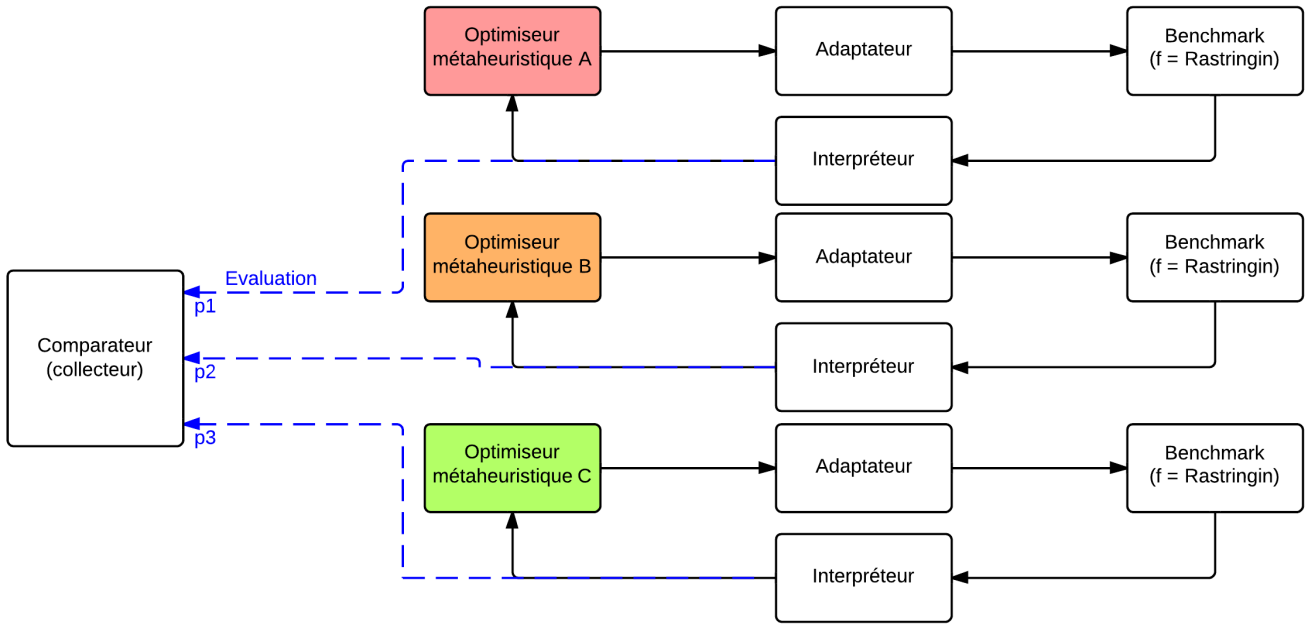


Figure 143 Exemple d'utilisation pour la comparaison de métaheuristiques

Bibliographie

Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E., 2002. Wireless sensor networks: a survey. Comput. Netw. 38, 393–422.

Alba, E., 2005. Parallel Metaheuristics: A New Class of Algorithms. John Wiley & Sons.

Antoine-Santoni, T., Santucci, J.-F., 2009. Performance of a protected wireless sensor network in a fire. Analysis of fire spread and data transmission. Sensors 9, 5878–93. doi:10.3390/s90805878

April, J., Glover, F., Kelly, J.P., Laguna, M., 2003. Practical introduction to simulation optimization, in: Simulation Conference, 2003. Proceedings of the 2003 Winter. Presented at the Simulation Conference, 2003. Proceedings of the 2003 Winter, pp. 71–78 Vol.1. doi:10.1109/WSC.2003.1261410

Bailleux, O., 1996. Contribution à l'étude des paysages de recherche locale associés au problème SAT. Université de Bourgogne.

Barros, F.J., 1995. Dynamic structure discrete event system specification: a new formalism for dynamic structure modeling and simulation, in: Simulation Conference Proceedings, 1995. Winter. Presented at the Simulation Conference Proceedings, 1995. Winter, pp. 781–785. doi:10.1109/WSC.1995.478858

Battiti, R., Tecchiolli, G., Nazionale, I., Nucleare, F., 1993. The Reactive Tabu Search, ORSA Journal on computing. Operations research society of America.

Bergero, F., Kofman, E., 2011. PowerDEVS: a tool for hybrid system modeling and real-time simulation. SIMULATION 87, 113–132. doi:10.1177/0037549710368029

Bisgambiglia, P.-A., de Gentili, E., Bisgambiglia, P.A., Santucci, J.F., 2009. Fuzz-iDEVS: Towards a Fuzzy Toolbox for Discrete Event Systems, in: Proceedings of the 2Nd International Conference on Simulation Tools and Techniques, Simutools '09. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, pp. 57:1–57:10. doi:10.4108/ICST.SIMUTOOLS2009.5691

Bisgambiglia, P.-A., Poggi, B., Nicolai, C., others, 2011. Models-based Optimization Methods for the Specication of Fuzzy Inference Systems in Discrete EVent Simulation., in: EUSFLAT Conf. pp. 957–964.

Boccara, N., 2010. Modeling Complex Systems, 2nd ed. 2010 edition. ed. Springer, New York.

Bolduc, J.-S., Vangheluwe, H., 2001. The modelling and simulation package PythonDEVS for classical hierarchical DEVS (Technical report No. MSDL-TR-2001-01). McGill University.

Boukelkoul, S., Redjimi, M., 2013. Mapping between Petri nets and DEVS models, in: 2013 3rd International Conference on Information Technology and E-Services (ICITeS). Presented at the 2013 3rd International Conference on Information Technology and e-Services (ICITeS), pp. 1–6. doi:10.1109/ICITeS.2013.6624067

Broutin, E., Bisgambiglia, P.-A., Santucci, J.-F., 2010. Multilayered DEVS Modeling and Simulation implementation, validation on concrete example: prediction of the behavior of a catchment basin, in: SCS (Ed.), Proceeding of the SCS 24th European Conference on Modelling and Simulation ECMS 2010. SCS, Kuala Lumpur, Malaisie, p. 8 pages.

Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R., 2013. Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. 64, 1695–1724. doi:10.1057/jors.2013.71

Capocchi, L., Bernardi, F., Federici, D., Bisgambiglia, P.-A., 2005. A DEVS-based Concurrent and Comparative Fault Simulation Algorithm, in: SCS (Ed.), SCS Summer Computer Simulation Conference Proceedings. SCS, San Diego, États-Unis, pp. 130–136.

Capocchi, L., Santucci, J.-F., 2013. Discrete optimization via simulation of catchment basin management within the devsimpy framework, in: Simulation Conference (WSC), 2013 Winter. Presented at the Simulation Conference (WSC), 2013 Winter, pp. 205–216. doi:10.1109/WSC.2013.6721420

Capocchi, L., Santucci, J.F., Poggi, B., Nicolai, C., 2011. DEVSimPy: A Collaborative Python Software for Modeling and Simulation of DEVS Systems, in: 2012 IEEE 21st International

Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. IEEE Computer Society, Los Alamitos, CA, USA, pp. 170–175. doi:10.1109/WETICE.2011.31
Casas, P.F. i (Ed.), 2013. Formal Languages for Computer Simulation: Transdisciplinary Models and Applications. IGI Global.
Cassandras, C.G., Lafortune, S., 2008. Introduction to Discrete Event Systems. Springer Science & Business Media.
Castro, R., Kofman, E., 2006. STDEVS: A Novel Formalism for Modeling And Simulation of Stochastic Discrete Event Systems, in: IN AADECA 2006, BUENOS AIRES.
Castro, R., Kofman, E., Wainer, G., 2010. A Formal Framework for Stochastic Discrete Event System Specification Modeling and Simulation. Simulation 86, 587–611. doi:10.1177/0037549709104482
Chakhlevitch, K., Cowling, P., 2005. Choosing the Fittest Subset of Low Level Heuristics in a Hyperheuristic Framework, in: Proceedings of the 5th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP'05. Springer-Verlag, Berlin, Heidelberg, pp. 23–33. doi:10.1007/978-3-540-31996-2_3
Chiong, R., 2009. Nature-Inspired Algorithms for Optimisation. Springer Science & Business Media.
Choi, B.K., Kang, D., 2013. Modeling and Simulation of Discrete Event Systems. John Wiley & Sons.
Chow, A.C.H., Zeigler, B.P., 1994. Parallel DEVS: A Parallel, Hierarchical, Modular, Modeling Formalism, in: Proceedings of the 26th Conference on Winter Simulation, WSC '94. Society for Computer Simulation International, San Diego, CA, USA, pp. 716–722.
Cowling, P., Kendall, G., Soubeiga, E., 2001. A Hyperheuristic Approach to Scheduling a Sales Summit, in: Burke, E., Erben, W. (Eds.), Practice and Theory of Automated Timetabling III, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 176–190.
D'Abreu, M.C., Wainer, G., 2003. Models for continuous and hybrid system simulation, in: Simulation Conference, 2003. Proceedings of the 2003 Winter. Presented at the Simulation Conference, 2003. Proceedings of the 2003 Winter, pp. 641–649 Vol.1. doi:10.1109/WSC.2003.1261479
Dantzig, G.B., Orden, A., Wolfe, P., others, 1955. The generalized simplex method for minimizing a linear form under linear inequality restraints. Pac. J. Math. 5, 183–195.
Dargie, W., 2010. Fundamentals of Wireless Sensor Networks: Theory and Practice, 1 edition. ed. Wiley, Chichester, West Sussex, U.K. ; Hoboken, NJ.
De Castro, L.N., Von Zuben, F.J., 2000. The clonal selection algorithm with engineering applications, in: Proceedings of GECCO. pp. 36–39.
Dorigo, M., Blum, C., 2005. Ant Colony Optimization Theory: A Survey. Theor Comput Sci 344, 243–278. doi:10.1016/j.tcs.2005.05.020
Dorigo, M., Maniezzo, V., Colorni, A., 1996. Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. Part B Cybern. 26, 29–41. doi:10.1109/3477.484436
Drake, F., Rossum, G., 2011. The Python Language Reference Manual. Network theory Ltd.
Eiben, A.E., Smit, S.K., 2011. Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol. Comput. 1, 19–31. doi:10.1016/j.swevo.2011.02.001
Ficici, S.G., 2008. Multiobjective optimization and coevolution, in: Multiobjective Problem Solving from Nature. Springer, pp. 31–52.
Filippi, J.-B., Bisgambiglia, P., 2004. JDEVS: an implementation of a DEVS based formal framework for environmental modelling. Environ. Model. Softw., Concepts, Methods and Applications in Environmental Model Integration 19, 261–274. doi:10.1016/j.envsoft.2003.08.016
Filippi, J.-B., Pialat, X., 2013. Assessment of ForeFire/Meso-NH for wildland fire/atmosphere coupled simulation of the FireFlux experiment. Proc. Combust. Inst. 34, 2633–2640. doi:10.1016/j.proci.2012.07.022
Fishman, G.S., 2001. Discrete-Event Simulation: Modeling, Programming, and Analysis. Springer Science & Business Media.
Franceschini Romain, Bisgambiglia, P.-A., 2014. DEVS-Ruby: a Domain Specific Language for DEVS Modeling and Simulation (WIP), in: TMS DEVS 2014. Presented at the Spring Simulation 2014 Multi-Conference, SCS, Tampa Bay, Florida.
Fu, M.C., Andradóttir, S., Carson, J.S., Glover, F., Harrell, C.R., Ho, Y.-C., Kelly, J.P., Robinson, S.M., 2000. Integrating optimization and simulation: research and practice, in: Simulation

Conference, 2000. Proceedings. Winter. Presented at the Simulation Conference, 2000. Proceedings. Winter, pp. 610–616. doi:10.1109/WSC.2000.899770
Fu, M.C., Glover, F.W., April, J., 2005. Simulation optimization: a review, new developments, and applications, in: Proceedings of the 37th Conference on Winter Simulation. Winter Simulation Conference, pp. 83–95.
Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman.
Geem, Z.W., 2009. Music-Inspired Harmony Search Algorithm: Theory and Applications. Springer Science & Business Media.
Gendreau, M., Potvin, J.-Y., 2010. Handbook of Metaheuristics, 2nd ed. 2010 edition. ed. Springer, New York.
Giambiasi, N., Escude, B., Ghosh, S., 2001. GDEVS: a generalized discrete event specification for accurate modeling of dynamic systems, in: 5th International Symposium on Autonomous Decentralized Systems, 2001. Proceedings. Presented at the 5th International Symposium on Autonomous Decentralized Systems, 2001. Proceedings, pp. 464–469. doi:10.1109/ISADS.2001.917452
Glover, F., 1989. Tabu Search—Part I. ORSA J. Comput. 1, 190–206. doi:10.1287/ijoc.1.3.190
Glover, F., 1990. Tabu Search — Part II. ORSA J. Comput. 2, 4–32.
Glover, F., 1990. Tabu Search: A Tutorial. Interfaces 20, 74–94. doi:10.1287/inte.20.4.74
Glover, F., Kochenberger, G.A., 2003. Handbook of Metaheuristics. Springer Science & Business Media.
Goti, A., 2010. Discrete Event Simulations. Sciyo.
Hagendorf, O., 2009. Simulation based parameter and structure optimisation of discrete event systems (thesis). Liverpool John Moores University.
Halim, R.., Seck, M.D., 2011. The simulation-based multi-objective evolutionary optimization (SIMEON) framework, in: Simulation Conference (WSC), Proceedings of the 2011 Winter. Presented at the Simulation Conference (WSC), Proceedings of the 2011 Winter, pp. 2834–2846. doi:10.1109/WSC.2011.6147987
Himmelspach, J., Uhrmacher, A., 2007. Plug'n Simulate, in: Simulation Symposium, 2007. ANSS '07. 40th Annual. Presented at the Simulation Symposium, 2007. ANSS '07. 40th Annual, pp. 137–143. doi:10.1109/ANSS.2007.34
Holland, J.H., 1992. Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA, USA.
Hong, J.S., Song, H.-S., Kim, T.G., Park, K.H., 1997. A Real-Time Discrete Event System Specification Formalismfor Seamless Real-Time Software Development. Discrete Event Dyn. Syst. 7, 355–375. doi:10.1023/A:1008262409521
Hopcroft, J.E., Motwani, R., Ullman, J.D., 2006. Introduction to Automata Theory, Languages, and Computation, 3 edition. ed. Prentice Hall, Boston.
Horn, J., Nafpliotis, N., Goldberg, D.E., 1994. A niched Pareto genetic algorithm for multiobjective optimization, in: , Proceedings of the First IEEE Conference on Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence. Presented at the , Proceedings of the First IEEE Conference on Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence, pp. 82–87 vol.1. doi:10.1109/ICEC.1994.350037
Hu, X., Zeigler, B.P., Mittal, S., 2005. Variable Structure in DEVS Component-Based Modeling and Simulation. Simulation 81, 91–102. doi:10.1177/0037549705052227
Hwang, M.H., 2007. Modeling and Simulation using DEVS#, first. ed. http://xsy-csharp.sourceforge.net/DEVSsharp.
Iassinovski, S., Artiba, A., Bachelet, V., Riane, F., 2003. Integration of simulation and optimization for solving complex decision making problems. Int. J. Prod. Econ., Planning and Control of Productive Systems 85, 3–10. doi:10.1016/S0925-5273(03)00082-3
Igel, C., 2014. No Free Lunch Theorems: Limitations and Perspectives of Metaheuristics, in: Borenstein, Y., Moraglio, A. (Eds.), Theory and Principled Methods for the Design of Metaheuristics, Natural Computing Series. Springer Berlin Heidelberg, pp. 1–23.
Ingalls, R.G., 2008. Introduction to simulation, in: Proceedings of the 40th Conference on Winter Simulation. Winter Simulation Conference, pp. 17–26.

Janoušek, V., Kironskỳ, E., 2006. Exploratory Modeling with SmallDEVS. Presented at the EMS 2006, Toulouse, France, pp. 122–126.

Kennedy, J., Eberhart, R., 1995. Particle swarm optimization, in: , IEEE International Conference on Neural Networks, 1995. Proceedings. Presented at the , IEEE International Conference on Neural Networks, 1995. Proceedings, pp. 1942–1948 vol.4. doi:10.1109/ICNN.1995.488968

Kim, J., Zeigler, B.P., 1996. Hierarchical Distributed Genetic Algorithms: A Fuzzy Logic Controller Design Application. IEEE Expert Intell. Syst. Their Appl. 11, 76–84. doi:10.1109/64.506756

Kim, T.G., 1994. DEVSim++ User's Manual:C++ Based Simulation with Hierarchical, Modular DEVS M.

Kim, T.G., Chang Ho Sung, Hong, S.-Y., Jeong Hee Hong, Chang Beom Choi, Jeong Hoon Kim, Kyung Min Seo, Jang Won Bae, 2011. DEVSim++ Toolset for Defense Modeling and Simulation and Interoperation. J. Def. Model. Simul. Appl. Methodol. Technol. 8, 129–142. doi:10.1177/1548512910389203

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by Simulated Annealing. Science 220, 671–680. doi:10.1126/science.220.4598.671

Kofman, E., Junco, S., 2001. Quantized-state Systems: A DEVS Approach for Continuous System Simulation. Trans Soc Comput Simul Int 18, 123–132.

Kwon, Y.W., Wan, Y., Hyu, K., Park, C., Hoon, S., Tag, J., Kim, G., 1996. Fuzzy-DEVS Formalism: Concepts, Realization and Applications. Presented at the Simulation and planning in High Autonomy Systems, University of Arizona, San Diego.

Laarhoven, P.J.M., Aarts, E.H.L. (Eds.), 1987. Simulated Annealing: Theory and Applications. Kluwer Academic Publishers, Norwell, MA, USA.

Lara, J. de, Vangheluwe, H., 2002a. Computer Aided Multi-paradigm Modelling to Process Petri-Nets and Statecharts, in: Proceedings of the First International Conference on Graph Transformation, ICGT '02. Springer-Verlag, London, UK, UK, pp. 239–253.

Lara, J. de, Vangheluwe, H., 2002b. AToM3: A Tool for Multi-formalism and Meta-modelling, in: Kutsche, R.-D., Weber, H. (Eds.), Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 174–188.

Lee, H., Zeigler, B.P., Kim, D., 2008. A DEVS-based framework for simulation optimization: Case study of Link-11 gateway parameter tuning, in: IEEE Military Communications Conference, 2008. MILCOM 2008. Presented at the IEEE Military Communications Conference, 2008. MILCOM 2008, pp. 1–7. doi:10.1109/MILCOM.2008.4753119

Lee, K.Y., El-Sharkawi, M.A., 2008. Modern Heuristic Optimization Techniques: Theory and Applications to Power Systems. John Wiley & Sons.

Leemis, L.M., Park, S.K., 2006. Discrete-event Simulation: A First Course. Pearson Prentice Hall.

Liu, Q., Wainer, G., 2007. Parallel Environment for DEVS and Cell-DEVS Models. Simulation 83, 449–471. doi:10.1177/0037549707085084

Luke, S., 2013. Essentials of Metaheuristics, second. ed. Lulu.

Michel, F., Ferber, J., Drogoul, A., 2009. Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. Multi-Agent Syst. Simul. Appl.

Mittal, S., Risco-Martin, J.L., Zeigler, B.P., 2009. DEVS/SOA: A Cross-Platform Framework for Net-centric Modeling and Simulation in DEVS Unified Process. SIMULATION 85, 419–450. doi:10.1177/0037549709340968

Moallemi, M., Wainer, G., Awad, A., Tall, D.A., 2010. Application of RT-DEVS in Military, in: Proceedings of the 2010 Spring Simulation Multiconference, SpringSim '10. Society for Computer Simulation International, San Diego, CA, USA, pp. 29:1–29:8. doi:10.1145/1878537.1878568

Molga, M., Smutnicki, C., 2005. Test functions for optimization needs. Test Funct. Optim. Needs Internal report.

Molter, H.G., 2012. SynDEVS Co-Design Flow: A Hardware / Software Co-Design Flow Based on the Discrete Event System Specification Model of Computation. Springer Science & Business Media.

Murata, T., Ishibuchi, H., 1995. MOGA: multi-objective genetic algorithms, in: , IEEE International Conference on Evolutionary Computation, 1995. Presented at the , IEEE International Conference on Evolutionary Computation, 1995, p. 289–. doi:10.1109/ICEC.1995.489161

Ntaimo, L., Hu, X., Sun, Y., 2008. DEVS-FIRE: Towards an Integrated Simulation Environment for Surface Wildfire Spread and Containment. Simulation 84, 137–155. doi:10.1177/0037549708094047

Ntaimo, L., Khargharia, B., 2006. Two-dimensional fire spread decomposition in cellular DEVS models, in: Proceedings of 2006 Spring Simulation Multi-Conference. pp. 2–5.

Nutaro, J.J., 2011. Building Software for Simulation: Theory and Algorithms, with Applications in C++. John Wiley & Sons.

Özcan, E., Bilgin, B., Korkmaz, E.E., 2008. A Comprehensive Analysis of Hyper-heuristics. Intell Data Anal 12, 3–23.

Peterson, J.L., 1981. Petri Net Theory and the Modeling of Systems. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Poggi, B., Santucci, J.-F., Thierry, A.-S., 2014. Optimization Based on Dynamic and Hybrid Metaheuristics via DEVS Simulation, in: Symposium on Theory of Modeling & Simulation - DEVS (TMS/DEVS). Presented at the Spring Simulation 2014 Multi-conference, Tampa, Florida.

Puccinelli, D., Haenggi, M., 2005. Wireless sensor networks: applications and challenges of ubiquitous sensing. IEEE Circuits Syst. Mag. 5, 19–31. doi:10.1109/MCAS.2005.1507522

Quesnel, G., Duboz, R., Ramat, É., 2009. The Virtual Laboratory Environment – An operational framework for multi-modelling, simulation and analysis of complex dynamical systems. Simul. Model. Pract. Theory 17, 641–653. doi:10.1016/j.simpat.2008.11.003

Raidl, G.R., 2006. A Unified View on Hybrid Metaheuristics, in: Almeida, F., Aguilera, M.J.B., Blum, C., Vega, J.M.M., Pérez, M.P., Roli, A., Sampels, M. (Eds.), Hybrid Metaheuristics, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 1–12.

Rao, S.S., 2009. Engineering Optimization: Theory and Practice, 4 edition. ed. Wiley, Hoboken, N.J.

Rashedi, E., Nezamabadi-pour, H., Saryazdi, S., 2009. GSA: A Gravitational Search Algorithm. Inf. Sci., Special Section on High Order Fuzzy Sets 179, 2232–2248. doi:10.1016/j.ins.2009.03.004

Reisig, W., 2013. Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer Science & Business Media.

Ricart, J., Hüttemann, G., Lima, J., Barán, B., 2011. Multiobjective Harmony Search Algorithm Proposals. Electron. Notes Theor. Comput. Sci., Proceedings of the 2011 Latin American Conference in Informatics (CLEI) 281, 51–67. doi:10.1016/j.entcs.2011.11.025

Risco-Martín, J.L., Mittal, S., Atienza, D., Hidalgo, J.I., Lanchares, J., 2008. Optimization of Dynamic Data Types in Embedded Systems Using DEVS/SOA-based Modeling and Simulation, in: Proceedings of the 3rd International Conference on Scalable Information Systems, InfoScale '08. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, pp. 17:1–17:11.

Rothlauf, F., 2011. Design of Modern Heuristics: Principles and Application. Springer Science & Business Media.

Sarjoughian, H.S., Cellier, F.E., 2001. Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and Ai-Based Theories and Methodologies : A Tribute to the 60th Birthday of Bernard P. Zeigler. Springer-Verlag New York Inc., New York.

Shannon, R.E., 1998. Introduction to the art and science of simulation, in: Proceedings of the 30th Conference on Winter Simulation. IEEE Computer Society Press, pp. 7–14.

Simon, D., 2013. Evolutionary Optimization Algorithms. Wiley-Blackwell, Hoboken, New Jersey.

Smit, S.K., Eiben, A., 2009. Comparing parameter tuning methods for evolutionary algorithms, in: IEEE Congress on Evolutionary Computation, 2009. CEC '09. Presented at the IEEE Congress on Evolutionary Computation, 2009. CEC '09, pp. 399–406. doi:10.1109/CEC.2009.4982974

Sokolowski, J.A., Banks, C.M., 2010. Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains. John Wiley & Sons.

Srinivas, N., Deb, K., 1994. Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. Evol Comput 2, 221–248. doi:10.1162/evco.1994.2.3.221

Talbi, E.-G., 2009. Metaheuristics: From Design to Implementation. Wiley-Blackwell, Hoboken, N.J.

Vangheluwe, H., 2001. The discrete event system specification (DEVS) Formalism. Course Notes, Course : Modeling and Simulation (COMP522A), McGill University, Montreal Canada.
Vangheluwe, H., De Lara, J., Mosterman, P., 2002. An introduction to multi-paradigm modelling and simulation. Proc. AIS2002 Conf. AI Simul. Plan. High Auton. Syst. April 2002 Lisb. Port. F Barros N Giambiasi Eds 9–20.
Wainer, G., 2002. CD++: A Toolkit to Develop DEVS Models. Softw Pr. Exper 32, 1261–1306. doi:10.1002/spe.482
Wainer, G., Giambiasi, N., 1997. Cell-DEVS models with transport and inertial delays Proceedings of the 9th. European Simulation Symposium and Exhibition. Passau, Germarny.
Wainer, G., Qi Liu, Dalle, O., Zeigler, B.P., 2010. Applying Cellular Automata and DEVS Methodologies to Digital Games: A Survey. Simul. Gaming 41, 796–823. doi:10.1177/1046878110378708
Wainer, G.A., 2009. Discrete-Event Modeling and Simulation: A Practitioner's Approach. CRC Press.
Wainer, G.A., Giambiasi, N., 2001. Application of the Cell-DEVS Paradigm for Cell Spaces Modelling and Simulation. SIMULATION 76, 22–39. doi:10.1177/003754970107600102
Wainer, G.A., Mosterman, P.J., 2010. Discrete-Event Modeling and Simulation: Theory and Applications. CRC Press.
Weise, T., 2008. Global Optimization Algorithms – Theory and Application –.
Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Trans. Evol. Comput. 1, 67–82. doi:10.1109/4235.585893
Yang, X.-S., 2008. Nature-Inspired Metaheuristic Algorithms. Luniver Press.
Yang, X.-S., Koziel, S., 2011. Computational Optimization: An Overview, in: Koziel, S., Yang, X.-S. (Eds.), Computational Optimization, Methods and Algorithms, Studies in Computational Intelligence. Springer Berlin Heidelberg, pp. 1–11.
Yick, J., Mukherjee, B., Ghosal, D., 2008. Wireless sensor network survey. Comput. Netw. 52, 2292–2330.
Zeigler, B.P., 1984. Multifaceted Modelling and Discrete Event Simulation. Academic Press Inc, London ; Orlando.
Zeigler, B.P., 2003. DEVS today: recent advances in discrete event-based information technology, in: 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. Presented at the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003, pp. 148–161. doi:10.1109/MASCOT.2003.1240652
Zeigler, B.P., Chi, S., 1992. Symbolic discrete event system specification. IEEE Trans. Syst. Man Cybern. 22, 1428–1443. doi:10.1109/21.199467
Zeigler, B.P., Hammonds, P.E., 2007. Modeling & Simulation-based Data Engineering: Introducing Pragmatics Into Ontologies for Net-centric Information Exchange. Academic.
Zeigler, B.P., Lee, J.S., 1998. Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment. pp. 49–58. doi:10.1117/12.319354
Zeigler, B.P., Moon, Y., Kim, D., Kim, J.G., 1996. DEVS-C++: a high performance modelling and simulation environment, in: System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on ,. Presented at the System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on ,, pp. 350–359 vol.1. doi:10.1109/HICSS.1996.495481
Zeigler, B.P., Moon, Y., Lopes, V.L., Kim, J., 1996. DEVS approximation of infiltration using genetic algorithm optimization of a fuzzy system. Math. Comput. Model. 23, 215–228. doi:10.1016/0895-7177(96)00074-X
Zeigler, B.P., Praehofer, H., Kim, T.G., 2000. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. Academic Press.
Zeigler, B.P., Sarjoughian, H.S., 2003. Introduction to devs modeling and simulation with java: Developing component-based simulation models. Tech. Doc. Univ. Ariz.
Zeigler, B.P., Sarjoughian, H.S., 2012. Guide to Modeling and Simulation of Systems of Systems, 2012 edition. ed. Springer, New York.

Zeigler, B.P., Vahie, S., 1993. Devs Formalism And Methodology: Unity Of Conception/Diversity Of Application, in: In Proceedings of the 25th Winter Simulation Conference. ACM Press, pp. 573–579.

Zimmer, S., Buchholz, M., Pflüger, D., 2013. Modeling and Simulation: An Application-Oriented Introduction, 2014 edition. ed. Springer.

Zong Woo Geem, Joong Hoon Kim, Loganathan, G.V., 2001. A New Heuristic Optimization Algorithm: Harmony Search. SIMULATION 76, 60–68. doi:10.1177/003754970107600201

Publications associées

Poggi, B., Antoine-Santoni, T., 2012. Wireless Sensor Network deployment using DEVS formalism and GIS representation, in: 2012 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS). Presented at the Summer Simulation 2012 Multi-Conference, Genoa, Italy.

Poggi, B., Santucci, J.-F., Antoine-Santoni, T., 2013. Wireless sensor deployment decision support using genetics algorithm and DEVS formalism. Presented at the EMSS: European Modelling & Simulation Symposium, 25th Edition, Part of I3M2013, Athens, Greece.

Poggi, B., Santucci, J.-F., Antoine-Santoni, T., 2014. Optimization Based on Dynamic and Hybrid Metaheuristics via DEVS Simulation, in: Symposium on Theory of Modeling & Simulation - DEVS (TMS/DEVS). Presented at the Spring Simulation 2014 Multi-conference, Tampa, Florida.

Liste des figures

Liste des tableaux

Algorithmes