Séquence robuste d'optimisation paramétrique d'un système d'inférence floue

Serge Guillaume, Brigitte Charnomordic

▶ **To cite this version:**

Serge Guillaume, Brigitte Charnomordic. Séquence robuste d'optimisation paramétrique d'un système d'inférence floue. Congrès LFA, 21e rencontres francophones sur la Logique Floue et ses Applications, Nov 2012, Compiègne, France. p. 45 - p. 52. hal-00813148

HAL Id: hal-00813148
https://hal.science/hal-00813148v1

Submitted on 15 Apr 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Author-produced version

Séquence robuste d'optimisation paramétrique d'un système d'inférence floue

Robust parameter optimization sequence for Fuzzy Inference Systems

Serge Guillaume[1] Brigitte Charnomordic[2]
[1] Irstea, UMR ITAP, F-34196 Montpellier
[2] INRA, UMR MISTEA, Montpellier

361 rue Jean-François Breton, F-34196 Montpellier, France, serge.guillaume@irstea.fr
2 place Viala, F-34060 Montpellier, France, bch@supagro.inra.fr

Résumé :
Ce travail présente une séquence robuste d'optimisation des paramètres d'un système d'inférence floue. Chacune des étapes permet d'optimiser un ensemble de paramètres interdépendants, suivant des critères de performance numérique mais aussi de couverture. La structure du système n'est pas modifiée par la procédure et des contraintes sont imposées pour garantir son interprétabilité. Dix couples apprentissage-test sont générés pour chaque jeu de données et les systèmes optimisés suivant chacun des sous-échantillons sont agrégés en un système final. L'ensemble des procédures est implémenté dans le logiciel libre *FisPro*.

Mots-clés :
Optimisation, sémantique

Abstract:
This work proposes a robust parameter optimization sequence for fuzzy inference systems. Each step allows for optimizing a set of interrelated parameters according to various criteria such as numerical accuracy and coverage. The fuzzy inference system structure is preserved and constraints are imposed to respect the fuzzy partition semantics. A ten-fold sub-sampling validation is also proposed for each of the datasets and all the systems optimized according to the sub-samples are aggregated in a final system. All the procedures are implemented in the *FisPro* open source software.

Keywords:
Optimization, semantics

1 Introduction

Les systèmes d'inférence floue (SIF) sont largement utilisés pour la modélisation et pour la prédiction, aussi bien en classification qu'en régression. Leur originalité, reconnue, réside dans la forme du modèle lui-même : une base de règles linguistiques faciles à interpréter. Cette interprétabilité est compatible avec un niveau de précision numérique comparable à celui de méthodes alternatives.

Quelle qu'ait été la technique utilisée pour la conception du SIF, sa performance numérique peut être améliorée par des procédures d'optimisation. Il faut toutefois veiller à ce que cet objectif ne remette pas en cause l'interprétabilité [2, 6].

Les travaux de recherche dans ce domaine se sont intéressés à différents niveaux d'optimisation que l'on peut classer en deux catégories : optimisation structurelle et optimisation paramétrique. Différentes méthodes sont disponibles pour ces deux niveaux : méthodes analytiques, principalement basées sur une descente de gradient [3], ou méthodes évolutionnistes, comme les algorithmes génétiques [1] ou les colonies de fourmis [10].

L'optimisation structurelle comprend la détermination des variables et des règles. La sélection de variables peut être réalisée au niveau global, pour l'ensemble du système, ou bien au niveau local pour chacune des règles. Dans le premier cas, la décision de suppression d'une variable implique son retrait de toutes les règles. Le nombre de fonctions d'appartenance par variable peut être également ajusté par des algorithmes appropriés. L'optimisation de la base de règles peut être réalisée par des procédures incrémentales, des fusions d'ensembles flous ou de de clusters, ou encore les règles peuvent être sélectionnées

in : Congrès LFA , 21e rencontres francophones sur la Logique Floue et ses Applications, Compiègne 15-16 Novembre 2012

sur des critères statistiques. Une revue des ces méthodes d'optimisation structurelle est proposée dans [9].

Le travail présenté ici s'intéresse à l'optimisation paramétrique : position des fonctions d'appartenance et conclusions des règles. Il propose une séquence d'optimisation des différents paramètres innovante et indépendante de l'algorithme d'optimisation lui-même. La procédure inclut une validation croisée sur k couples d'échantillons aléatoires et propose d'agréger les SIF optimisés pour chaque couple en un système final. La démarche est illustrée à partir d'un algorithme évolutionniste simple et efficace.

Ces développements sont implémentés dans un logiciel *open source* appelé *FisPro* [1].

FisPro permet de concevoir par apprentissage des SIF dont l'interprétabilité est garantie à tous les niveaux. Le partitionnement des variables est très contraint : seules des partitions floues fortes sont générées, ainsi chacune des fonctions d'appartenance est associée sans ambiguïté à un concept linguistique. Les partitions sont générées à partir de la seule distribution des données de la variable et peuvent donc être utilisées par tous les algorithmes d'induction des règles. Le logiciel est utilisé pour appliquer l'ensemble de la démarche à deux jeux de données bien connus, disponibles dans le dépôt de l'UCI[7].

La section suivante présente la procédure d'optimisation. Dans la section 3, elle est appliquée aux données expérimentales et les résultats sont analysés. La section 4 résume les principales conclusions.

2 La procédure d'optimisation

L'algorithme d'optimisation est au cœur de la procédure. Nous avons choisi d'utiliser celui proposé par Glorennec [8], et inspiré du travail de Solis et Wets [11].

1. http ://www.inra.fr/mia/M/fispro/

Il est décrit dans l'algorithme 1.

Algorithm 1: Algorithme de Solis et Wets

Entrée: Vecteur initial $X^{(0)}$, amplitude du bruit $Nmag$

Sortie : Vecteur optimisé, $X = X^{(Max)}$

1 Initialisation : $k = 0$, $M^{(0)} = 0$.
2 **while** $k \leq$ *Max* **do**
3 Générer un vecteur gaussien $G^{(k)}$,
4 de moyenne $M^{(k)}$,
5 avec un bruit d'amplitude $Nmag$
6 **if** $E(X^{(k)} + G^{(k)}) < E(X^{(k)})$ **then**
7 $X^{(k+1)} = X^{(k)} + G^{(k)}$
8 $M^{(k+1)} = 0.2M^{(k)} + 0.4G^{(k)}$
9 **else if** $E(X^{(k)} - G^{(k)}) < E(X^{(k)})$ **then**
10 $X^{(k+1)} = X^{(k)} - G^{(k)}$
11 $M^{(k+1)} = M^{(k)} - 0.4G^{(k)}$
12 **else**
13 $X^{(k+1)} = X^{(k)}$
14 $M^{(k+1)} = 0.5M^{(k)}$
15 **end**
16 $k = k + 1$
17 **end**

Le vecteur de paramètres est noté $X(.)$ et l'erreur associée au système $E(.)$.

Il s'agit d'un algorithme évolutionniste, dont le comportement est guidé. A chaque itération, le vecteur courant $X(.)$ est modifié par un vecteur gaussien $G^{(k)}$ dont l'écart type est une fonction du paramètre $Nmag$.

Comme l'ensemble des méthodes d'optimisation aléatoire, l'algorithme ne nécessite pas de calcul de gradient et n'impose donc pas de conditions de continuité ou de différentiabilité pour les fonctions. Il diffère d'un algorithme purement aléatoire dans le sens où les *bonnes* directions de recherche sont mémorisées au moyen du vecteur M (lignes 8 et 11). En cas d'échec, le biais ainsi introduit est progressivement oublié, car exponentiellement décrémenté (ligne 14).

Les valeurs des différents coefficients 0.4, 0.2 and 0.5 sont celles issues des expériences de Solis et Wets [11]. Le paramètre $Nmag$ est fixé à

0.005 pour la suite de ce travail.

L'ensemble des paramètres du système d'inférence floue peut être optimisé : partitions des variables d'entrée ou de sortie, conclusions des règles.

2.1 Contraintes et paramètres de la procédure

Plusieurs définitions sont nécessaires pour introduire le paramètre de contrôle *Perte de couverture*.

Definition 2.1 *Les exemples d'apprentissage sont étiquetés comme actifs ou inactifs pour une base de règles donnée. Un exemple est actif si son degré de vérité maximum, sur l'ensemble des règles, est supérieur à un seuil donné, appelé seuil d'activité, inactif dans le cas contraire.*

Definition 2.2 *Le niveau de couverture pour est système est* $CI^k = \frac{A}{N}$*, où* A *est le nombre d'exemples actifs,* N *le nombre total d'exemples, et* k *le numéro d'itération.* CI^0 *est donc le niveau de couverture initial.*

Definition 2.3 *La Perte (relative) de couverture pour l'itération* k *est* $\frac{CI^0 - CI^k}{CI^0}$.

Ce paramètre est utilisé pour rejeter, au cours de la procédure d'optimisation, les partitions floues qui dégraderaient trop la représentativité du système. Une perte de couverture est possible lorsque la base de règles est incomplète, c'est-à-dire lorsque toutes les règles possibles du fait du partitionnement des variables ne sont pas implémentées. Cette situation est très fréquente, notamment dans les espaces de grande dimension.

Des *contraintes* peuvent être imposées pour garantir les propriétés sémantiques des partitions. Lors de l'optimisation des paramètres des fonctions d'appartenance les solutions qui ne conduisent pas à des partitions floues fortes sont rejetées et considérées comme des échecs. Il est également possible d'imposer une distance minimale entre points caractéristiques des ensembles de la partition.

2.2 Une séquence d'optimisation

Tous les paramètres du système pourraient être optimisés en même temps mais, comme la recherche est biaisée par le vecteur M, des éléments qui ont un comportement différent agiraient de façon contradictoire sur le vecteur M, ce qui pénaliserait la recherche. Il est préférable de proposer une séquence et d'optimiser à chaque étape seulement des ensembles d'éléments interdépendants, qui se comportent de manière analogue. Comment définir de tels ensembles ? Les conclusions de toutes les règles doivent être optimisées dans la même étape car un exemple est susceptible de déclencher plusieurs règles. De même, les fonctions d'appartenance d'une variable donnée sont en relation mutuelle et doivent être traitées ensemble. Les partitions floues fortes triangulaires sont entièrement définies par un nombre de points caractéristiques égal au nombre de termes linguistiques, ces valeurs (C_1, C_2, C_3 sur la figure 3), constituent donc le vecteur à optimiser.

L'algorithme de Solis & Wets ne garantit pas de trouver l'optimum global. En conséquence, l'ordre de la séquence est susceptible d'influer sur les résultats, et il est conseillé de réitérer la procédure d'optimisation. Une itération de la séquence est appelée une boucle dans ce qui suit.

La procédure d'optimisation proposée dans *FisPro* est donc la suivante :

1. Ordonner les variables d'entrée avec *Fistree*, l'implémentation de l'arbre de décision flou dans *FisPro*. Les plus importantes apparaissent au sommet de la hiérarchie,
2. Optimiser les partitions des variables

d'entrée sélectionnées suivant cet ordre,

3. Si la sortie est floue, optimiser sa partition,
4. Optimiser les conclusions des règles,
5. Si le nombre de boucles demandé n'est pas atteint, aller à l'étape 2.

Cette procédure, bien qu'elle traite séparément chacune des variables d'entrée, ne fait pas l'hypothèse de leur indépendance.

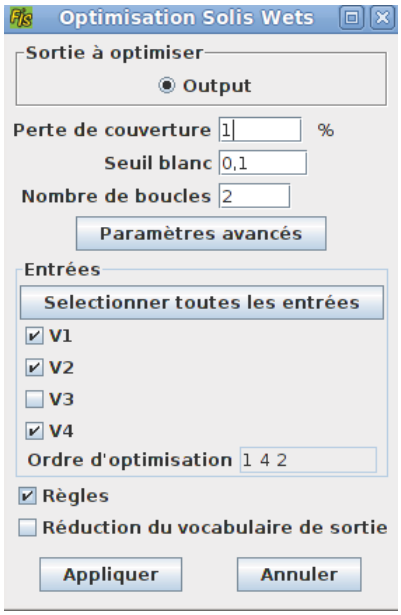

Figure 1 – Interface de saisie des paramètres de la séquence d'optimisation dans *FisPro*

La figure 1 représente une capture d'écran de la fenêtre de saisie des paramètres de la séquence dans *FisPro*. Les paramètres principaux sont le seuil d'activité, *seuil blanc* ainsi que la perte de couverture autorisée et le nombre de boucles, égal à 2 par défaut. Les paramètres avancés permettent de modifier le niveau du bruit gaussien et de contrôler le germe du générateur aléatoire. Cette interface fait appel à une bibliothèque de fonction C++, qui est également disponible en ligne de commande pour permettre des traitements automatisés.

Rappelons que la procédure d'optimisation relève de l'otpimisation paramétrique, et donc ne modifie pas la structure du système : le nombre de termes linguistiques par variable reste le même, aucune variable n'est ajoutée ni enlevée dans la prémisse des règles.

La procédure maintient également l'interprétabilité. Les partitions modifiées sont des partitions floues fortes et l'ordre des termes linguistiques est préservé : la valeur du centre de la $k^{ième}$ fonction d'appartenance ne peut être supérieure à celle de la $(k+1)^{ième}$.

3 Expérimentation

Dans cette section, la procédure d'optimisation est appliquée sur deux jeux de données bien connus de l'UCI [7]. Voici leurs caractéristiques principales :

– *Cpu-performance* (209 exemples) :
Publié par Ein-Dor et Feldmesser [5], ces données contiennent les performances de CPU ainsi que 6 variables descriptives continues telles que la taille mémoire ou la durée du cycle.
– *Auto-mpg* (392 exemples) :
Issu de la bibliothèque StatLib maintenu par la Carnegie Mellon University, ce jeu de données vise à prédire la consommation de carburant en *miles per gallon* à partir de 4 variables continues et 3 discrètes multivaluées.

L'indice d'erreur est le *Mean Absolute Error* (MAE) défini dans l'équation 1, y_i et $\widehat{y}_i$ sont respectivement les sorties observée et inférée pour le $i^{ème}$ exemple.

$$MAE = \frac{1}{A}\sum_{i=1}^{A}|\widehat{y}_i - y_i| \qquad (1)$$

Notons que la valeur de cet indice est calculée à partir de A, le nombre d'exemples actifs (voir la définition 2.1).

Des tests pour évaluer la sensibilité des résultats au seuil d'activité ont été conduits dans la gamme $0.01 - 0.2$. Les résultats se sont révélés robustes vis-à-vis de ce paramètre. Une valeur de 0.1 est utilisé pour la suite de ces tests.

Une vue d'ensemble de la démarche expérimentale est présentée sur la figure 2.

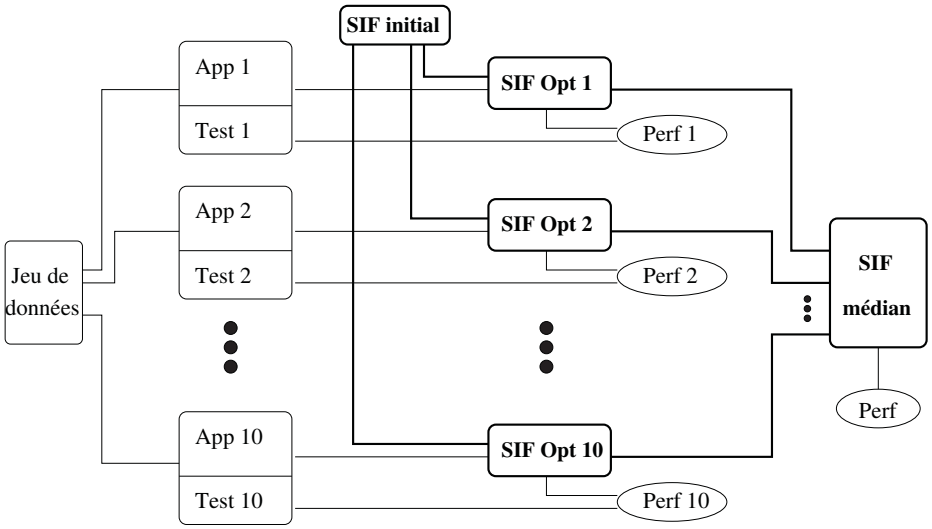

Figure 2 – Démarche d'ensemble de l'expérimentation

3.1 Deux bases de règles

Pour illustrer le caractère générique de la procédure d'optimisation, celle-ci est appliquée pour chaque jeu de données à des bases de règles induites par deux algorithmes différents. Le premier est l'implémentation *FisPro* d'une méthode basée sur une orthogonalisation des règles. Appelé *Fuzzy Orthogonal Least Squares*, il est noté simplement OLS. Le second est appelé FPA, acronyme de *Fast Prototyping Algorithm*. Chacun d'eux est brièvement présenté ci-après.

OLS. Cette implémentation de l'algorithme est décrite dans [4]. D'inspiration statistique, OLS est limité à des sorties continues (cas de régression). Les règles sélectionnées sont celles qui expliquent une nouvelle part de variance. Les résultats sont d'autant plus représentatifs que la taille de l'échantillon d'apprentissage est importante.

FPA. Cet algorithme simple procède en deux étapes : génération de toutes les règles possibles suivant les combinaisons des entrées, puis initialisation des conclusions en fonction des données.
Pour les sorties continues, ce calcul est réalisé suivant l'équation 2.

$$C_r = \frac{\sum\limits_{i \in E_r} \mu_r(x_i) * y_i}{\sum\limits_{i \in E_r} \mu_r(x_i)} \qquad (2)$$

$\mu_r(x_i)$ est le degré de vérité de l'exemple i pour la règle r. E_r est un ensemble d'exemples choisis suivant leur degré de vérité pour la la règle. Sa cardinalité est un paramètre qui permet de contrôler le caractère général de la règle induite. Si le nombre d'exemples qui activent la règle au delà d'un seuil paramétrable est insuffisant, la règle n'est pas générée.

L'algorithme FPA peut traiter des problèmes de classification ou de régression. Plus rapide que l'OLS, il est aussi moins précis lorsque les données disponibles sont nombreuses. Comme la génération des règles obéit à des critères différents, les règles de FPA et d'OLS sont en général différentes.

3.2 Système initial

Il est évident que les résultats de l'optimisation dépendent du système initial : il est très facile d'améliorer un système peu performant. La structure, nombre d'ensembles

flous par variable, nombre de règles, est en effet complètement déterminée par ce point de départ. Pour illustrer le potentiel de la procédure, il est nécessaire de choisir un système initial suffisamment précis et il est donc normal que les gains apportés par l'optimisation ne soient pas très élevés.

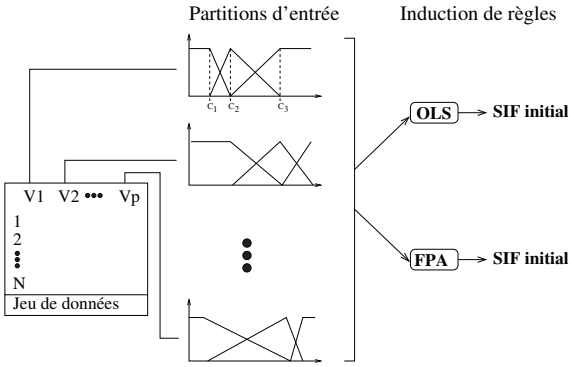

Figure 3 – Conception des systèmes initiaux

La même procédure est appliquée pour les deux jeux de données :

- Les partitions des variables d'entrée sont générées à partir de l'ensemble des données par l'algorithme des *k-means*, avec $k = 3$ groupes par partition. Pas de partition floue pour la sortie, la conclusion des règles sera un scalaire (système de *Sugeno* d'ordre zéro).
- La génération des règles se fait suivant l'algorithme OLS à partir de la totalité des données. Le nombre de règles sélectionnées permet d'atteindre un niveau de précision, mesuré par le MAE (équation 1), comparable à celui publié dans [4], et donné dans le Tableau 1.

Tableau 1 – Caractéristiques du système initial OLS

	MAE	# règles
Auto-mpg	2.02	54
Cpu	28.77	9

- Les mêmes partitions sont utilisées pour générer les règles avec la méthode FPA, les paramètres sont choisis pour conserver le même nombre de règles qu'avec l'OLS, voir Tableau 1.
- L'arbre de décision flou permet d'ordonner les variables suivant leur importance :
  - Auto-mpg : 1 6 4 5
  - Cpu : 3 6 2

3.3 Optimisation

On obtient donc quatre SIF initiaux, deux pour chaque jeu de données. Dix couples d'échantillons apprentissage (75% des données) / test (complément) sont générés de façon aléatoire. La séquence d'optimisation est appliquée sur les quatre SIF initiaux, pour chacun des échantillons d'apprentissage. Le bruit gaussien est fixé à 0.005, la perte de couverture autorisée set de 10%.

Le tableau 2 résume les résultats, avec MAE et le gain relatif moyenné sur les dix échantillons de test. La performance moyenne est améliorée pour les deux systèmes et les deux jeux de données.

Tableau 2 – Optimisation : MAE et gain relatif moyennés sur les dix échantillons de test

		Ols	Fpa
Auto-mpg	Initial	2.02	2.22
	Optimisation	1.96	2.14
	Gain (%)	3.0	3.6
Cpu	Initial	28.77	31.1
	Optimisation	27.47	30.49
	Gain (%)	4.5	2.0

L'optimisation n'efface pas les différences entre les bases de règles : celle générée par l'OLS produit de meilleurs résultats que son homologue FPA.

3.4 Sélection d'un système final

La procédure de ré-échantillonnage produit autant de systèmes que de couples échantillons apprentissage - test. Ces dix systèmes partagent la même structure : même nombre de fonctions d'appartenance par variable et même nombre de règles, prémisses des règles identiques.

Pour faciliter la réutilisation, un SIF est créé

par un processus d'agrégation des dix systèmes optimisés : chacun des paramètres du système est remplacé dans le système final par sa valeur médiane, plus robuste que la moyenne.

Les résultats du *système médian* moyennés sur les dix échantillons de test sont donnés dans le tableau 3. Ils sont significativement meilleurs que la moyenne de ceux des systèmes optimisés par un échantillon d'apprentissage testés sur l'ensemble de test correspondant.

Tableau 3 – Résultats du système médian : MAE et gain relatif moyennés sur les dix échantillons de test

		Ols	Fpa
Auto-mpg	MAE	1.91	2.01
	Gain (%)	5.4	9.5
Cpu	MAE	27.05	29.83
	Gain (%)	6.0	4.1

Cela s'explique par la robustesse du système médian : le nombre d'échantillons test sur lesquels le système optimisé est moins performant que l'initial est très faible (3 pour les deux jeux de données). De plus cette dégradation est nettement plus limitée, moins de 1 % dans deux cas sur les trois.

3.5 Discussion

Comme leurs structures sont identiques, il est facile de comparer les systèmes avant et après optimisation. Dans les tableaux 4 and 5, I signifie initial, et F correspond à final, soit le *système médian*.

Le tableau 4 compare les points caractéristiques (voir figure 3) des partitions des variables optimisées pour le système OLS et les données CPU.

Le tableau 5 montre l'évolution des conclusions des règles pour le même système. La prémisse est définie par le numéro d'ordre de la fonction d'appartenance de chacune des variables.

Comme attendu, le système optimisé n'est pas très différent de l'initial, il présente le même niveau d'interprétabilité linguistique.

Tableau 4 – Système Cpu OLS : évolution des points caractéristiques des fonctions d'appartenance

	C_1		C_2		C_3	
	I	F	I	F	I	F
V2	2292	2634	14484	14259	31310	31379
V3	10404	7789	27254	27838	63110	63168
V6	12021	6916	65833	64972	144000	144000

Tableau 5 – Système Cpu OLS : les bases de règles avant et après optimisation

Prémisse						I	F
1	1	3	2	2	3	1033	1023
1	3	3	2	2	2	1144	1140
1	2	2	1	2	1	339	322
1	1	2	1	1	1	146	133
1	2	3	1	2	1	658	639
1	1	1	1	1	1	37	37
1	2	2	2	2	1	332	314
1	1	2	2	3	2	274	272
1	2	2	3	2	1	395	396

Cependant, les légères modifications introduites par la procédure d'optimisation améliorent la précision numérique même lorsque celle du système initial est acceptable.

4 Conclusion

La séquence d'optimisation pour les paramètres d'un système d'inférence flou, fonctions d'appartenance des variables d'entrée et de sorties, conclusions des règles, est guidée par des critères de précision numérique et de couverture. Elle est illustrée dans ce travail par une technique d'optimisation aléatoire, l'algorithme de Solis et Wets, mais demeure valide pour d'autres méthodes.

L'algorithme de Solis et Wets est une méthode évolutionniste mono-agent. Sa complexité, en temps comme en espace, est notablement réduite comparée à celle d'un algorithme génétique. En effet, il n'est pas nécessaire de maintenir une population (il n'y a pas d'opération de reproduction génétique) et le temps de convergence est rendu plus court par la mémorisation des *bonnes* directions de recherche (le nombre d'itérations par

défaut est 100 dans *FisPro*). Grâce aux contraintes imposées, la performance numérique est améliorée sans perte de sémantique associée aux partitions et aux règles. La procédure d'optimisation requiert un nombre réduit de paramètres, ce qui facilite son utilisation.

La procédure a été appliquée à deux jeux de données de l'UCI, et sur deux bases de règles générées par différents algorithmes. Elle s'est révélée robuste et efficace, indépendamment du système initial. La robustesse a été évaluée par une procédure de validation croisée. Les différents systèmes optimisés ont été agrégés pour former un système final, médian. Pour les quatre configurations étudiées, l'optimisation conduit à une amélioration substantielle de la performance numérique tout en préservant l'interprétabilité.

L'implémentation dans le logiciel *FisPro* met la méthode à la libre disposition de tous, soit sous la forme d'une interface utilisateur conviviale, soit à travers des exemples de scripts pour des traitements automatisés. L'interface comporte la séquence présentée ici et également une procédure complètement personnalisable,

Références

[1] Rafael Alcala, Pietro Ducange, Francisco Herrera, Beatrice Lazzerini, and Francesco Marcelloni. A multiobjective evolutionary approach to concurrently learn rule and data bases of linguistic fuzzy-rule-based systems. *IEEE Transactions on Fuzzy Systems*, 17(5) :1106–1122, 2009.

[2] Jorge Casillas, Oscar Cordón, Francisco Herrera, and Luis Magdalena. Interpretability improvements to find the balance interpretability-accuracy in fuzzy modeling : an overview. In *Interpretability Issues in Fuzzy Modeling*, pages 3–22. Springer, 2003.

[3] Long Chen, C.L.P. Chen, and Witold Pedrycz. A gradient-descent-based approach for transparent linguistic interface generation in fuzzy models. *IEEE Transactions on Systems, Man and Cybernetics, part B : Cybernetics*, 40(5) :1219–1230, 2010.

[4] Sébastien Destercke, Serge Guillaume, and Brigitte Charnomordic. Building an interpretable fuzzy rule base from data using orthogonal least squares- application to a depollution problem. *Fuzzy Sets and Systems*, 158 :2078–2094, 2007.

[5] Phillip Ein-Dor. Grosch's law re-revisited : Cpu power and the cost of computation. *Commun. ACM*, 28(2) :142–151, 1985.

[6] Alexandre G. Evsukoff, Sylvie Galichet, Beatriz S.L.P. de Lima, and Nelson F.F. Ebecken. Design of interpretable fuzzy rule-based classifiers using spectral analysis with structure and parameters optimization. *Fuzzy sets and Systems*, 160 :857–881, April 2009.

[7] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[8] Pierre-Yves Glorennec. *Constrained optimization of FIS using an evolutionary method*, pages 349–368. Genetic Algorithms and Soft Computing. Physica-Verlag, 1996.

[9] Serge Guillaume. Designing fuzzy inference systems from data : an interpretability-oriented review. *IEEE Transactions on Fuzzy Systems*, 9(3) :426–443, 2001.

[10] Chia-Feng Juang and Po-Han Chang. Designing fuzzy-rule-based systems using continuous ant-colony optimization. *IEEE Transactions on Fuzzy Systems*, 18(1) :138–149, 2010.

[11] Francisco J. Solis and Roger J.B. Wets. Minimization by random search techniques. *Mathematics of Operation Research*, 6 :19–30, 1981.