

Why leximin FCSPs are so hard for local search

João Moura Pires, Fernando Moura Pires, Rita Almeida Ribeiro

Departamento de Informática - FCT

Universidade Nova de Lisboa

2825 Monte Caparica, Portugal

{jmp, fmp, rr}@di.fct.unl.pt

Abstract: In this paper we investigate why solving leximin fuzzy constraint satisfaction problems (FCSP) with non-enumerative algorithms, such as hill-climbing, simulated annealing and genetic algorithms, provide disappointing results. Specifically, we analyse the structure and properties of the leximin FCSPs and its influence on the search strategy of non-enumerative algorithms.

1. Introduction

This paper suggests that the use of non-enumerative optimisation algorithms, without including further information to guide the search, is not a good approach to solve leximin-optimal Fuzzy constraint satisfaction problems (FCSP). Due to the nature and shape of the leximin it is not easy to find a good solution without adding more information so as to locate areas containing “good” solutions and in order to guide the search

Fuzzy constraint satisfaction problems are an extension of constraint satisfaction problems (CSP). The crisp constraints are replaced by fuzzy constraints [1] to express the gradual violation of elastic constraints. In a FCSP a constraint is satisfied to a degree (rather than just satisfied or not satisfied), hence the acceptability of a potential solution also becomes a gradual notion. FCSPs are applied in many different areas such as structural design [16] and scheduling problems [9]

Solving a FCSP amounts, generally, to find an instantiation for its variables that maximizes the satisfaction level of the least satisfied constraint. Indeed, the approach aggregates the constraints by means of the min operator. It has been shown that due to the idempotence of the min operation, most CSP computation techniques easily extend to the fuzzy setting [3], [4]. Since min-optimal solutions to a FCSP may be numerous, two refinements of the ranking of solutions have been proposed, namely those referred to as the **discrimin** and **leximin** orderings [5]. These refinements do not only take into account the least satisfied constraint, but also compare other satisfaction levels. In this paper, we focus on the **leximin FCSP** because a leximin-optimal solution is also a discrimin-optimal solution.

Non-enumerative algorithms can be classified into three main categories: population-based, multi-point and single-point. The search process of non-enumerative algorithms requires two *à priori* definitions: selection of the number of points to be used simultaneously, and selection of the starting points. Some algorithms use single starting points, as for instance Hill Climbing (HC), Tabu Search (TS) [14][15] and Simulated Annealing (SA) [2]. Other algorithms are based in a population of points, as for instance Genetic Algorithms (GA) [13]. Further, single-point algorithms can also be used as multi-run versions. In these multi-run versions, the starting points

may be randomly chosen or constructed with heuristics, when available. The perspective here is not to discuss the algorithms, per se, but to address their suitability to solve leximin FCSP.

The use of non-enumerative algorithms to solve CSP has proven to be rather successful [12]. Hence, the use of non-enumerative optimisation algorithms to solve FCSP should also be a valuable tool (some attempts along these lines already exist in the literature [20]).

The paper is organized as follows. Section 2 provides the background of FCSP's. Section 3 discusses the structure and properties of the leximin FCSP. Section 4 suggests some hints on how to improve the search process of non-enumerative algorithms in order to solve leximin FCSP. Section 5 presents future research topics.

2. Background on FCSP's

Crisp constraint-satisfaction-based models offer a general framework for interpretation design and decision problems [20], [18]. A constraint satisfaction problem is defined by a set of variables on finite domains and a set of constraints related to these variables. A CSP solution is an instantiation of the problem variables that satisfy all the constraints. When such solution exist the CSP it is said to be consistent, otherwise it is called inconsistent.

When faced with CSP's that have too many equally satisfying solutions, these problems can be solved with an extension of the CSP, called **fuzzy constraint satisfaction problems** [3]. The extension is achieved by replacing crisp constraints with fuzzy constraints [1], where the idea of satisfying a constraint is replaced by a satisfaction level (degree) for the constraints.

Formally, a FCSP is defined by a set of decision variables $\mathbf{X} = \{X_1, \dots, X_N\}$, a set of domains $\mathbf{D} = \{D_1, \dots, D_N\}$ where D_i is the domain of X_i , and is supposed to be finite (with cardinality $|D_i| = d_i$) and a set of fuzzy constraints $\mathbf{C} = \{C_1, \dots, C_K\}$ where C_k is a fuzzy relation defined over the Cartesian product of the domains of the variables included in C_k . Thus, a membership value is associated to each tuple of values of the variables related by C_k . The membership degree express preferences among solutions by ranking the instantiations which are more or less acceptable with respect to the satisfaction of the flexible constraint C_k . In this paper we are dealing with FCSP's where no natural order exists for the variable domains.

In the sequel, we consider fuzzy sets defined by a finite linearly-ordered valuation set, where only the ordering between grades is meaningful as, for instance $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. With such a valuation set, fuzzy-set intersection, union and complementation can only be defined by min, max and the order-reversing operation of the valuation set. The valuation set is denoted L , with $L + 1$ elements, where 0 and L denote the bottom and top elements, respectively

A solution to a FCSP is denoted by \mathbf{x} and is a vector of values in $\mathbf{X} = D_1 \times \dots \times D_N$. The level of satisfaction for the constraints, denoted $Sat(\mathbf{x})$, is defined, as proposed by Bellman and Zadeh [1], as follows:

$$Sat(\mathbf{x}) = \min_k \mu_{C_k}(\mathbf{x}), \quad (1)$$

and the optimal solution, \mathbf{x}^* , so called "min optimal" is:

$$Sat(\mathbf{x}^*) = \max_{\mathbf{x}} \min_k \mu_{C_k}(\mathbf{x}). \quad (2)$$

In a fuzzy framework, solving a FCSP problem amounts to find at least one solution that is maximal for $>_{min}$. Hence, finding a solution for a FCSP is an optimization problem.

The satisfaction level of a min-optimal solution for a FCSP is said to be:

- **inconsistent**, if it does not satisfy at least one constraint.
- **fully consistent**, if it completely satisfies all constraints.
- **partially consistent**, if there is at least one constraint - not fully satisfied - and the satisfaction levels of all constraints are positive.

For the min-based FCSP's, the solutions can be discriminated by (at most) L levels of consistency. In each equivalence class, all solutions share the same satisfaction level, which is the one of the least satisfied constraint(s). Two solutions in the same equivalence class are not discriminated, even if they could be distinguished by considering other levels of satisfaction for the remaining constraints.

In addition, it is well known that CSP problems are NP-complete [17]. The main tools for solving CSP's are backtracking-based search algorithms and constraint-propagation techniques. Finding a min-optimal solution is a NP-hard problem, which can be solved using a branch and bound-based algorithm, [3], [4]. The use of the min-operator allows for a direct generalization to the FCSP of the constraint propagation algorithms used in the classical CSP framework (for example, the AC3 by Mackworth [17]). The levels of satisfaction are propagated by extensions of existing algorithms (see, for example, [3], [4]). This is essentially due to the idempotence of the min-operator.

2.1. Refinements of the min-ordering

There are two main refinements for the min-ordering in a FCSP framework: the **discrimin**, proposed by Fargier *et al.* [10], and the **leximin** (see Moulin, [19]). A detailed study of these refinements, with different equivalent formulations, can be found in [5],

In general, for each \mathbf{x} in X we can define a fuzzy set over the set of constraints C which represents the constraints satisfied by \mathbf{x} . Here, such a fuzzy set is denoted by $\mathcal{S}(\mathbf{x})$, defined as $\{C_k / \mu_{C_k}(\mathbf{x})\}$.

Let \mathbf{x} and \mathbf{y} be two solutions and let $\mathcal{S}(\mathbf{x})$ and $\mathcal{S}(\mathbf{y})$ be the corresponding fuzzy sets, respectively. With the **discrimin** ordering, we compare the least satisfied discriminating constraint, i.e., we look for the lowest-satisfied constraint among the constraints that are not equally satisfied. Let us denote by $\Delta(\mathbf{x}, \mathbf{y})$, the subset of constraints that are not equally satisfied, by \mathbf{x} and \mathbf{y} , i.e., the constraints C_k for which $\mu_{C_k}(\mathbf{x}) \neq \mu_{C_k}(\mathbf{y})$. Formally $\mathbf{x} >_{\text{discrimin}} \mathbf{y}$ iff:

$$\min_{C_k \in \Delta(\mathbf{x}, \mathbf{y})} \mu_{C_k}(\mathbf{x}) > \min_{C_k \in \Delta(\mathbf{x}, \mathbf{y})} \mu_{C_k}(\mathbf{y}) \quad (3)$$

Whenever the least satisfied discriminating constraints are equally satisfied by \mathbf{x} and \mathbf{y} , neither $\mathbf{x} >_{\text{discrimin}} \mathbf{y}$ nor $\mathbf{x} <_{\text{discrimin}} \mathbf{y}$ and \mathbf{x} and \mathbf{y} are said to be *indifferent*.

The **leximin** ordering refines the discrimin ordering. A solution \mathbf{x} is preferred to a solution \mathbf{y} , in a *leximin* order, if there is a threshold α such that for all $\beta < \alpha$, the

number of constraints satisfied by \mathbf{x} (at level at least β) is equal to the number of constraints satisfied by \mathbf{y} , and it satisfies more constraints than \mathbf{y} at level α . This is equivalent to comparing the cardinalities at increasing level cuts.

Let $\mathbf{u}(\mathbf{x})$ denote the ranking vector corresponding to $\mathcal{S}(\mathbf{x})$ where the elements of $\mathbf{u}(\mathbf{x})$ are in an increasing order, i.e., in such way that $u(\mathbf{x})_k < u(\mathbf{x})_{k+1}$ for all k from 1 to $K - 1$. Then, formally, $\mathbf{x} >_{\text{leximin}} \mathbf{y}$ iff $\exists j < K$, such that $\forall i < j$, $u(\mathbf{x})_i = u(\mathbf{y})_i$ and $u(\mathbf{x})_j > u(\mathbf{y})_j$, and furthermore, $\mathbf{x} =_{\text{leximin}} \mathbf{y}$ iff $\mathbf{u}(\mathbf{x}) = \mathbf{u}(\mathbf{y})$. Fargier, [8], has shown that a leximin optimal solution is a solution that violates the smallest number of fuzzy constraints, in the sense of fuzzy cardinality.

These different orderings (min, discrimin and leximin) correspond to an increasing refinement. Non-distinguishable solutions for one order may be distinguished by the next one.

Dubois *et al* [7], discussed how to compute discrimin-optimal and leximin-optimal solutions for special classes of FCSP's, and they present algorithms for some specific problems such as, for instance, scheduling problems with flexible constraints, as well as flexible assignment problems. The underlying idea is to find a min-optimal solution and then, by saturating the least satisfied constraint(s), to generate a new sub-problem, whose min-optimal solutions has been improved (in the sense of discrimin or leximin) with respect to the first solution found. This process is repeated until no more sub-problems can be generated; and a leximin or discrimin optimal solution is reached. However, for a general FCSP the generation of the sub-problems requires an NP-complete algorithm.

In the next section we will discuss the structure and properties of leximin FCSP's. Since, a *leximin* optimal solution is also a *discrimin* optimal one, the *discrimin* solution is not further discussed in this paper

3. Leximin Structure and Properties

Let \mathcal{L}_{KL} be the set of lexi-vectors for K constraints and a satisfaction scale with $L + 1$ levels, as defined in the previous section. In the following we show how to calculate the cardinality of \mathcal{L}_{KL} and discuss various mappings from \mathcal{L}_{KL} to $[0, 1]$.

3.1. $\mathcal{L}_{K,L}$ Cardinality

For a problem with K constraints and by using a satisfaction scale with $L + 1$ degrees, the number of possible different leximin vectors grow exponentially with both K and L . Let *lexi-levels*(K, L) be a function that calculates the cardinality $|\mathcal{L}_{KL}|$, defined by (see appendix for details):

$$\text{lexi-levels}(K, L) = \begin{cases} 1 & L = 0 \\ L + 1 & K = 1 \\ \sum_{i=0}^L \text{lexi-levels}(K - 1, L - i) & \end{cases}$$

Note that, for implementation purposes, the above can be efficiently calculated by the following expression $\text{lexi-levels}(K, L) = \text{lexi-levels}(K - 1, L) + \text{lexi-levels}(K, L - 1)$ (see appendix).

To find a min-optimal solution of a FCSP corresponds to an optimization problem with an optimization function that can have $L + 1$ different values. The corresponding lexicmin-optimal problem may use a very large number of different values. Figure 1, shows how many lexi-vectors are needed for a problem characterized by $L = 4$ (with a satisfaction scale of $\{0, 1, 2, 3, 4\}$) and for a number of constraints varying from 2 to 10.

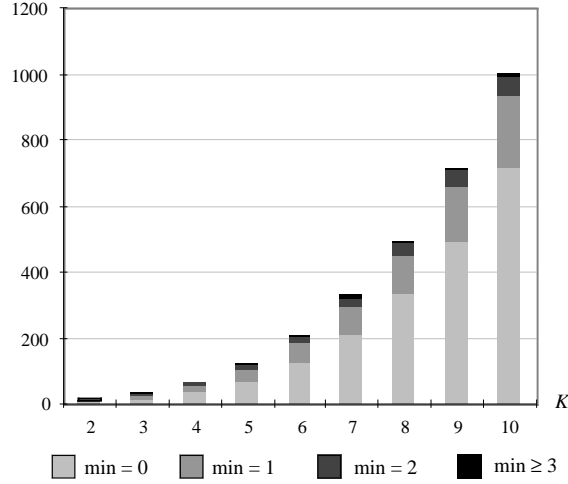


Figure 1. Lexi-vectors for $L = 4$ and K in $\{1, \dots, 10\}$.

The lexicmin order is a refinement of the min order. Figure 1 shows how each min-level is refined in terms of how many lexi-vectors correspond to each min-level. Specifically, the number of lexi-vectors indicating inconsistency (when at least one of the constraints is not satisfied) is $lexi-level(K - 1, L)$.

Even when lexicmin ordering is a total order, it is important to note that each lexi-vector corresponds to a set of possible satisfaction levels on the constraints, i.e., all the possible permutations of the satisfaction levels used in that lexi-vector. The lexicmin is also a refinement of discrimin-order and some discrimin incomparable solutions can be discriminated by the lexicmin. Let's denote by $Inc_{ordering}(\mathcal{S})$ the set of fuzzy sets that are incomparable to or equal to \mathcal{S} by one specific ordering. For instance, for $L = 2$ and $K = 3$, a solution that fully satisfies constraint C_2 , and does not satisfies C_1 and C_3 , is the fuzzy set $\mathcal{S} = \{C_1/0, C_2/2, C_3/0\}$. The discrimin considers this solution incomparable with the following, $Inc_{discrim}((0\ 2\ 0)) = \{(0\ 0\ 1), (0\ 0\ 2), (1\ 0\ 0), (1\ 0\ 1), (1\ 0\ 2), (2\ 0\ 0), (2\ 0\ 1), (2\ 0\ 2)\}$. With the lexicmin we are able to discriminate $(0\ 2\ 0)$ from all the cases in $Inc_{discrim}$, except $(0\ 0\ 2)$ and $(2\ 0\ 0)$.

3.2 Optimisation Function

The choice of the mapping to be used in the optimization function is relevant for those algorithms that use the reward value, i.e., the difference in the objective function between a point and a neighbor. This includes the simulated-annealing (SA), tabu-search (TS), and genetic-algorithms (GA), when the later use roulette

wheel or integral selection for instance. In such choices, one important factor is the sensibility variation along the lexi-vectors.

We will consider any mapping $T: \mathcal{L}_{K,L} \rightarrow [0, 1]$ such that:

- $T((0 \dots 0)) = 0$;
- $T((L \dots L)) = 1$;
- T is strictly increasing with respect to the lexicmin order.

Let \mathbf{u} denote a lexi-vector and $Pos(\mathbf{u})$ is its ranking in the lexicmin-ordering, such that $Pos((0 \dots 0)) = 1$ and $Pos((L \dots L)) = |\mathcal{L}_{K,L}|$ (see appendix for details). In this paper, we will consider four types of mappings: linear, logarithmic, OWA (ordered weighted average), and logarithmic-OWA. Figure 2 shows these mappings, for a problem with $K = 3$ and $L = 4$.

A linear mapping from the lexi-vectors, ranked by lexicmin ordering, to $[0, 1]$, is denoted by T_{linear} and defined:

$$T_{\text{linear}}(\mathbf{u}) = (Pos(\mathbf{u}) - 1) / (|\mathcal{L}_{K,L}| - 1)$$

Note that with T_{linear} the distance between two consecutive lexi-vectors is always the same no matter which consecutive ones are considered. With the mapping denoted by $T_{\text{loglinear}}$ defined as:

$$T_{\text{loglinear}}(\mathbf{u}) = \log_{|\mathcal{L}_{K,L}|}(Pos(\mathbf{u})),$$

the distance between two consecutive lexi-vectors decreases towards the top lexi-vector, and therefore the scale sensitivity is greater at lower lexi-vectors (which represent more inconsistency)

The lexicmin can be expressed as an OWA [22], as shown by [6]:

$$OWA_{\lambda}(\mathbf{u}) = \sum_{i=1}^K u_i \cdot \lambda^{K-i},$$

where λ is a parameter that should be greater than L in order to guarantee that this additive version ranks the lexi-vectors, as the lexicmin ordering do. Hence:

$$T_{OWA(\lambda)}(\mathbf{u}) = OWA_{\lambda}(\mathbf{u}) / OWA_{\lambda}((L \dots L))$$

With T_{OWA} the greater distances occur at each modification of the min, i.e., when the lexi-vectors as $(\alpha - 1, \alpha, \dots, \alpha)$ and (α, \dots, α) for $\alpha = 1$ to L . But this mapping shows a better sensibility on top lexi-vectors than on lower lexi-vectors.

Therefore, we can define $T_{\text{LogOWA}(\lambda)}$ as

$$T_{\text{LogOWA}(\lambda)}(\mathbf{u}) = \log_{OWA_{\lambda}((L, \dots, L)+1)}(OWA_{\lambda}(\mathbf{u}) + 1),$$

(see also Figure 2.) which runs a better sensibility on lower lexi-vectors than the $OWA_{(\lambda)}$.

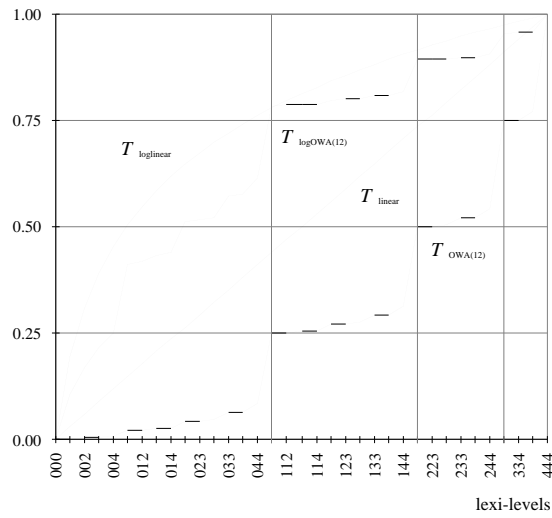


Figure 2. Examples of leximin optimisation functions for $K = 3$ and $L = 4$.

Figure 3 shows for $T_{\text{loglinear}}$, $T_{\text{OWA}(12)}$ and $T_{\text{LogOWA}(12)}$, for each lexi-vector the incremental change from the previous one. As expected, $\Delta T_{\text{loglinear}}$ continuously decreases; $\Delta T_{\text{LogOWA}(12)}$ and $\Delta T_{\text{OWA}(12)}$ show a peak at each min transition, but the former increase the peak amplitude as we go to the top lexi-vectors while the second decreases them. Furthermore, in the inconsistency region (min = 0), only $\Delta T_{\text{LogOWA}(12)}$ shows a good variation.

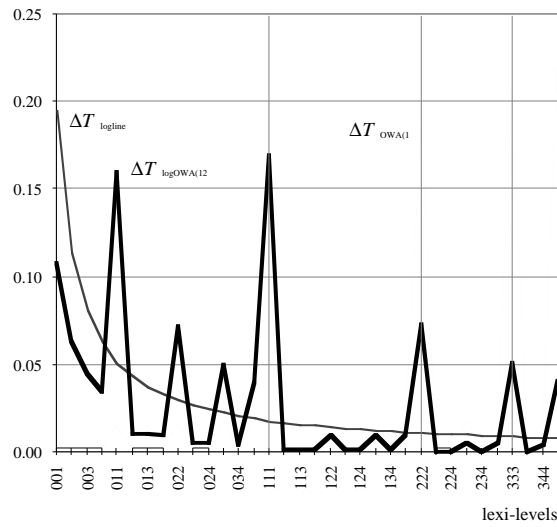


Figure 3. Sensibility examples of leximin optimisation functions for $K = 3$ and $L = 4$.

3.3. T on a Neighbourhood of \mathbf{x}

The guidance of single point optimization algorithms is based on the variation of the value of the optimization function on the current point and a set of points that are accessible from the current point (it's neighbours). When this variation is “big” all over the solution space, this kind of algorithm faces more difficulty and can be compared to a random search search.

Consider an FCSP, P . The leximin version of P can be stated as the following optimization problem: find a $\mathbf{x}^* \in X$ such that

$$\mathbf{x}^* = \max_{\mathbf{x} \in X} T(\mathbf{u}(\mathbf{x})),$$

where $\mathbf{u}: X \rightarrow \mathcal{L}_{K,L}$ encodes the FCSP (as defined above) and $T: \mathcal{L}_{K,L} \rightarrow [0, 1]$. We can say that \mathbf{u} sets the *structural* aspects of the function to be optimized and T its *intensity* (without changing its structural aspects).

The intensity (T) is only relevant for the optimization algorithms whose search depends on the values of $T(\mathbf{u}(\mathbf{x})) - T(\mathbf{u}(\mathbf{x}'))$. But all the optimization algorithms will be influenced by the structural aspects.

We will show that leximin FCSP as formulated above has globally an “irregular shape”.

Given a point \mathbf{x} in the search space, the single-point algorithms consider a neighborhood of \mathbf{x} , denoted as $N_\Upsilon(\mathbf{x})$, as the set of points \mathbf{x}' reachable from \mathbf{x} by means of the operator Υ , which is a transformation on some components of \mathbf{x} . On of the most common used operators is changing the value of one component, i. e., $\Upsilon(x_1, \dots, x_i, \dots, x_N) = (x_1, \dots, x_i', \dots, x_N)$, where $x_i' \in D_i - \{x_i\}$.

At each application of operator Υ , at least one constraint is involved and, therefore, at least one constraint may becomes completely unsatisfied; which correspond to saying that no matter what is the lexi-vector of the current point of the search (in the consistency region), the quality of the solution may drop (in its neighbourhood) to the inconsistency region. In terms of the optimization function this means that it may exist “holes” in the neighborhood of any point in the consistency region. The drop, if exists, may be greater when the current value is near to 1.0. In general, we can identify at least the following patterns of possible behavior on a neighborhood of \mathbf{x} , where $\mathbf{u} = \mathbf{u}(\mathbf{x})$:

- when $\min(\mathbf{u}) > 0$ an application of Υ may lead to points in the inconsistency region, i. e., with constraints not satisfied at all, or at least with $\min(\mathbf{u}') < \min(\mathbf{u})$. This possibility is more likely when $\min(\mathbf{u})$ increases. The difference $T(\mathbf{u}(\mathbf{x})) - T(\mathbf{u}(\mathbf{x}'))$ may be greater when $\min(\mathbf{u})$ increases
- when $\mathbf{u} = (\alpha \dots \alpha)$ (or with a large number of constraints whose satisfaction degree is equal to the min of \mathbf{u}), only small improvements are allowed (supposing a small number of constraints are affected by Υ);
- when $\mathbf{u} = (\alpha \beta \dots \beta)$, the application of Υ may produce good improvements (if Υ focused on the variables related by the least satisfied constraint); more generally, whenever there are a very small number of constraints whose satisfaction degrees are equal to $\min(\mathbf{u})$ the search of good improvements should be done by focusing Υ on the variables related by those constraints;

- a leximin optimal solution may be near (in terms of operator Υ) to points with very small quality and in general “good” points can have neighbours with “poor” quality.

It should be noted that whenever the variable domains are linearly ordered and the fuzzy relations take that domain order into account, this local perturbation on the optimization function will be smoothed in realistic problems.

Let’s consider an additive version of a FCSP P , denoted as P_{Σ} and stated as follows: find a $\mathbf{x}^* \in X$ such that

$$\mathbf{x}^* = \max_{\mathbf{x} \in X} \sum_{k=1, K} \mu_{C_k}(\mathbf{x}).$$

P_{Σ} is not equivalent to the min or leximin version of P , in the sense that the order of solution is not the same. Meanwhile, we refer to it only for comparison purposes in terms of optimization function defined in the same space. P_{Σ} does not have the local perturbation observed in the leximin version of P .

4. Influence on Optimisation Problems of the Leximin Shape

The main theoretical conclusion arrived in the previous sections is that the leximin optimization function, for FCSP’s with non-ordered variable domains, usually displays very irregular “shapes” on the neighborhood of many points of the solution space. This property affects the quality of results obtained with non-enumerative algorithms (such as SA, HC, GA) when applied without further information to guide the search with special codification. Some preliminary experimental evidences are available, but are not explicitly presented in this paper, and they are as follows.

For a set of binary FCSP’s, randomly generated with the parameters $N = 10$, $|D_i| = 10$ for $i = 1$ to N , $K = 27$ and $L = 10$) [21] the SA and GA fail to find good quality solutions. The same occurs with a multi-run version of HC. In both cases, the starting points used to run the problems were generated randomly.

Conversely, when the same problems were tested with starting points generated with many different heuristics on a multi-run HC, the results obtained were quite good [20].

Considering the leximin structural problems and the preliminary results obtained, we clearly see that further information is needed to enable non-enumerative algorithms to perform well. The main information that can be included in the algorithms to improve the search process is:

- use of problem information to generate good starting points [20] (or a starting population for GA);
- a dynamic selection of the operator Υ by using problem information and the pattern quality of the current solution;
- choosing more suitable mappings, T for the optimisation function.

5. Future research

In this paper, we discussed the complex structure and properties of the leximin FCSP. Due to that complex structure, solving these problems with non-enumerative

algorithms is not a suitable tool, without including further information to guide the search.

In order to fully support our claim we need further research on the topics:

- **Shape:** Study the variability on the leximin rank at the neighbourhood of a solution. For randomly chosen points calculate the average, standard deviation, percentiles etc. to extract information about that might be included in the optimisation function.
- **T influence:** Systematic study of the influence of the optimisation function upon the solution performance (quality, number of iterations, CPU time). This study should be done for SA, TS, and GA using the same set of benchmarks problems.

Other aspects that we also think worthwhile considering are:

- Testing the use of different optimisation functions at different stages of the search (depending on, for instance, the quality of current solution, pattern of u in the current solution, number of current iteration).
- Tuning of the algorithm control parameters depending on the pattern of u in the current solution. (see [11]).

6. References

- [1] Bellman R.E, and L. A. Zadeh. (1970). "Decision-making in a fuzzy environment" *Management Science*, 17, 141-164.
- [2] Connoly, D. (1992). "General Purpose Simulated Annealing",. *Journal of the Operational Research Society*, 43(5), 495-505.
- [3] Dubois, D., H. Fargier, and H. Prade (1994) "Propagation and satisfaction of flexible constraints". In R. Yager and L. Zadeh (eds.) , *Fuzzy Sets, Neural Networks and Soft Computing.*, Kluwer Academic Publ., Dordrecht, 266-187
- [4] Dubois, D., Fargier, H. and H. Prade, (1996a) "Possibility Theory in Constraint Satisfaction Problems: Handling Priority. Preference and Uncertainty", *Applied Intelligence*, 6 pp. 287-307.
- [5] Dubois, D., Fargier, H. and H. Prade, (1996b) "Refinements of the maximin approach to decision making in a fuzzy environment", *Fuzzy Sets and Systems*, Vol. 81, pp. 103-122.
- [6] Dubois D., H. Fargier, and H. Prade (1996). "Ordered weighted operations, "discrimin" and leximin in multicriteria decision" *Soft Computing with Industrial Applications* (Proc. of the 2nd World Automation Congress ,WAC'96), Montpellier, France, May 27-30, 1996. (M. Jamshidi, M. Fathi, F. Pierrot, (eds.), TSI Press Series, Albuquerque, NM, 167-172.
- [7] Dubois, D. and Fortemps, P. (1997). "Improved solutions to fuzzy constraint satisfaction problems", *Proc. of the 5th Europ. Congress on Intelligent Techniques and Soft Computing (EUFIT'97)*, Aachen, Germany, 1997, pp. 967-969.
- [8] Fargier, H. (1994). "Problèmes de satisfaction de contraintes flexibles: application à l'ordonnement de production". Ph. D. Thesis., Université Paul Sabatier, Toulouse.
- [9] Fargier, H. (1997). "Fuzzy scheduling: principles and experiments", in: *Fuzzy Information Engineering: A Guided Tour of Applications* (D. Dubois et al., eds.), Wiley, New York, 1997, pp. 655-668.

- [10] Fargier, H., J. Lang, and T. Schiex (1993). "Selecting preferred solutions in fuzzy constraint satisfaction problems". *Proc. of the 1st Europ. Congress on Fuzzy and Intelligent Technologies (EUFIT'93)*, Aachen, Germany, Sept. 7-10, 1128-1134.
- [11] Fortemps, P., Moura-Pires, J. and Prade, H. (1997). "Towards expressing heuristics with fuzzy logic", submitted.
- [12] Freuder, E. C., and Wallace, R. J. (1992). Partial constraint satisfaction. *AI* - 58, 21-70
- [13] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison Wesley.
- [14] Glover, F. (1989) "Tabu Search - Part 1", *ORSA Journal on Computing* 1(3), pp. 190-206.
- [15] Glover, F. (1990) "Tabu Search - Part 2", *ORSA Journal on Computing* 2(1), pp.4-32.
- [16] Guan, Q. and Friedrich, G. (1992). "Extending constraint satisfaction problem solving in structural design", *Proc. of the 5th Inter. Conf. IEA/AIE*, Paderborn, Germany, June, 1992, pp. 341-350.
- [17] Mackworth, A. (1977). "Consistency in networks of relations". *Artificial Intelligence* , 8, 99-121.
- [18] Montanari, H. (1974). "Networks of constraints: Fundamental properties and applications to picture processing". *Information Sciences* 7, 95-142.
- [19] Moulin H. (1988). *Axioms of Cooperative Decision Making*. Cambridge University Press, Cambridge, UK.
- [20] Moura-Pires, J. (1997) "Starting points in optimization algorithms for solving discrimin and leximin Fuzzy-CSP", *Actes de la 3ème Conf. Nationale de Résolution Pratique de Problèmes NP-Complets (JNPC'97)*, Rennes, France, 1997, pp. 95-101.
- [21] Moura-pires, J. (1997) "Random generation of binary CSPs and FCSPs". Internal report DI-97, Departamento de Informática, Universidade Nova de Lisboa.
- [22] Yager, R. R. (1988) "On Ordered Weighted Averaging Aggregation Operators in Multicriteria Decision Making". *IEEE Transactions on Systems, Man, and Cybernetics*. 18, 183-190.

Appendix: Combinatorial aspects of leximin

For K constraints and a satisfaction scale with $L + 1$ levels we can calculate the number of lexi-vectors between (including the limits):

- $(0 \dots 0)$ and $(L \dots L)$ by

$$\text{lexid}(K, L, 0, L) = \sum_{i=0}^L \text{lexi-levels}(K-1, L-i)$$

- $(j \dots j)$ and $(L \dots L)$ by

$$\text{lexid}(K, L, j, L) = \sum_{i=j}^L \text{lexi-levels}(K-1, L-i)$$

- $(j \dots j)$ and $(l L \dots L)$ by

$$\text{lexid}(K, L, j, L) = \sum_{i=j}^l \text{lexi-levels}(K-1, L-i)$$

Therefore the number of lexi-vectors such that the least satisfied constraint is α , i. e., the number of lexi-vectors that refines a min level of α is calculated by

$$\text{lexid}(K, L, \alpha, \alpha)$$

Further more the function *lexi-level* verify the following equations:

- $\text{lexi-levels}(K, L) = \text{lexi-levels}(K - 1, L) + \text{lexi-levels}(K, L - 1)$
- $\text{lexi-levels}(K, L) = \text{lexi-level}(L, K)$

Given a lexi-vector $\mathbf{u} = (u_1, \dots, u_k, \dots, u_K)$, where $u_k \leq u_{k+1}$, is easy to calculate its position, denoted by $\text{Pos}(\mathbf{u})$, in the leximin ordering, such that $\text{Pos}((0 \dots 0)) = 1$ and $\text{Pos}((L \dots L)) = |\mathcal{L}_{K,L}|$, by:

$$\text{lexid}(K, L, 0, u_1 - 1) + \sum_{i=2}^K \text{lexid}(i, L, u_{k-i}, u_{k-i+1} - 1),$$

supposing that $u_0 \neq 0$. If its not the case the leading zeros must be stripped and the K lowered by the same number of stripped zeros.